Steiner Connectivity Augmentation and Splitting-off in Poly-logarithmic Maximum Flows*

Ruoxu Cen† William He‡ Jason Li§ Debmalya Panigrahi¶

Abstract

We give an almost-linear time algorithm for the Steiner connectivity augmentation problem: given an undirected graph, find a smallest (or minimum weight) set of edges whose addition makes a given set of terminals τ-connected (for any given $\tau > 0$). The running time of our algorithm is dominated by polylogarithmic calls to *any* maximum flow subroutine; using the recent almost-linear time maximum flow algorithm (Chen et al., FOCS 2022), we get an almost-linear running time for our algorithm as well. This is tight up to the polylogarithmic factor even for just two terminals. Prior to our work, an almost-linear (in fact, near-linear) running time was known only for the special case of global connectivity augmentation, i.e., when all vertices are terminals (Cen et al., STOC 2022).

We also extend our algorithm to the closely related Steiner splitting-off problem, where the edges incident on a vertex have to be *split-off* while maintaining the (Steiner) connectivity of a given set of terminals. Prior to our work, a nearly-linear time algorithm was known only for the special case of global connectivity (Cen et al., STOC 2022). The only known generalization beyond global connectivity was to preserve all pairwise connectivities using a much slower algorithm that makes n calls to an all-pairs maximum flow (or Gomory-Hu tree) subroutine (Lau and Yung, SICOMP 2013), as against polylog(n) calls to a (single-pair) maximum flow subroutine in this work.

1 Introduction

Given an undirected graph $G = (V, E)$ with a set of terminals $T \subseteq V$ and a target connectivity $\tau > 0$, the *Steiner connectivity augmentation* problem asks for an edge set of minimum cardinality whose addition to G makes T τ-connected, where a set of vertices T is τ-connected if for every pair of vertices $s, t \in T$, the (s, t)-connectivity is $\geq \tau$. In the weighted version of the problem, edges in G can have nonnegative integer weights and the goal is to minimize the total weight of edges added to G to make T τ-connected. All edge weights and the target connectivity are assumed to be polynomially bounded in the size of the graph. We distinguish between adding edges of nonnegative *weight* and nonnegative *cost*. An edge of *weight* w is equivalent to w parallel edges; adding it increases the value of every cut that it belongs to by w and adds w to the objective that we seek to minimize. In contrast, if edges have nonuniform *costs* and the goal is to minimize the total cost of the added edges, then an edge only increases the value of a cut it belongs to by 1, but incurs a given nonnegative cost in the objective. This version of the problem with nonuniform costs is known to be NP-hard. We do not consider this version of the problem in this paper.

In this paper, we give an algorithm for Steiner connectivity augmentation in weighted graphs that runs in polylog(n) times $F(m, n)$ time, where $F(m, n)$ is the running time of any maximum flow algorithm on a graph with m edges and n vertices. This is optimal up to the polylog(n) term since a Steiner connectivity augmentation algorithm can be used to determine (s, t) connectivity by setting $T = \{s, t\}$ and (binary) searching for the maximum τ that admits an empty solution. If we use the recent breakthrough $m^{1+o(1)}$-time maximum flow algorithm of Chen et al. [16], then the running time of our algorithm becomes $m^{1+o(1)}$.

The special case of this problem where $T = V$, i.e., the goal is to increase the *global* connectivity of the graph to a target τ, has been extensively studied in the literature. Watanabe and Nakamura [48] were the first to give a polynomial-time algorithm for this problem in unweighted graphs, while the first strongly polynomial algorithm for weighted graphs was obtained by Frank [18]. Since then, several algorithms [9, 44, 24, 23, 43, 5, 11]

*Ruoxu Cen and Debmalya Panigrahi were supported in part by NSF grants CCF-1750140 (CAREER Award) and CCF-1955703.

†Department of Computer Science, Duke University. Email: ruoxu.cen@duke.edu

‡Duke University. Email: william.he@duke.edu

§Simons Institute for the Theory of Computing, UC Berkeley. Email: jmli@alumni.cmu.edu

¶Department of Computer Science, Duke University. Email: debmalya@cs.duke.edu

have progressively improved the running time for this problem until it was recently shown to be solvable in $\tilde{O}(m)$ time [12].[1] Another natural special case is when $T = \{s, t\}$. In this case, the optimal solution is to add an edge of weight $\tau - \lambda(s, t)$ between s and t, where $\lambda(s, t)$ is the (s, t)-connectivity. Thus, for $|T| = 2$, the problem is solvable in $F(m, n)$ time. But for general T not equal to V or $\{s, t\}$, it is not a priori obvious if the problem is polynomial-time tractable. In fact, by changing the problem slightly where only a given set of edges can be added, the problem becomes NP-hard. Therefore, it is somewhat surprising that the Steiner connectivity augmentation problem can be solved for general T not just in polynomial-time, but with only a polylog(n) overhead over the trivial case of $|T| = 2$.

A related problem to connectivity augmentation is that of *splitting-off*. In this problem, we are given a graph $G = (V, E)$ and a vertex $x \in V$ that has to be split-off. A splitting-off operation at x pairs two edges (u, x) and (x, v) and replaces them by the shortcut edge (u, v) in the graph. The goal is to split-off all edges incident on x while preserving some connectivity property of the remaining graph. Lovász [39] introduced the splitting-off operation and showed that any vertex can be split-off while preserving the *global* connectivity of the remaining graph. Since then, many splitting-off theorems have been shown (e.g., [40, 45]) that preserve a rich set of connectivity properties. The splitting-off operation has been an influential inductive tool for both extremal graph theory and graph algorithms with applications in network design [15, 13], tree packing [35, 7, 14], tree decomposition [42], graph orientation [19, 21, 6], and metric TSP [25].

Preserving a rich set of connectivity properties, however, comes at the cost of slower algorithmic implementation. While an $\tilde{O}(m)$-time implementation of Lovász's theorem has recently been shown [12], the best implementations of stronger splitting-off theorems are much slower. In this paper, we extend our Steiner connectivity augmentation algorithm to obtain an algorithm for Steiner splitting-off, namely split-off a vertex x while preserving the Steiner connectivity of any set of terminals $T \subseteq V$. (The Steiner connectivity of a set of terminals T is the minimum (s, t) connectivity for any terminals $s, t \in T$.) Our Steiner splitting-off algorithm has the same running time as the Steiner connectivity augmentation algorithm, namely polylog(n) calls to any maximum flow algorithm. To the best of our knowledge, prior to our work, the only known splitting-off algorithms that preserve a more general property than global connectivity actually preserve all pairwise connectivities at a much slower running time. The current best algorithm for this latter problem is due to Lau and Yung [36], whose running time is dominated by n calls to a Gomory-Hu tree subroutine. The paper describes the running time of the algorithm as is $\tilde{O}(m + n\tau^3)$ for unweighted graphs where τ is the maximum pairwise connectivity. This is based on the best known (partial) Gomory-Hu tree algorithm at the time of the Lau-Yung paper [28, 7]. Using the more recent $\tilde{O}(n^2)$-time Gomory-Hu tree algorithm for weighted graphs or the $m^{1+o(1)}$-time Gomory-Hu tree algorithm for unweighted graphs [1] improves the running time of the Lau-Yung algorithm to $\tilde{O}(n^3)$ in weighted graphs and $mn^{1+o(1)}$ in unweighted graphs. In this paper, we improve the running time for both weighted and unweighted graphs further from $\tilde{O}(n^3)$ and $mn^{1+o(1)}$ respectively to $m^{1+o(1)}$, but only for the Steiner splitting-off problem.

THEOREM 1.1. *There are $m^{1+o(1)}$-time algorithms for the Steiner connectivity augmentation and Steiner splitting-off problems on undirected graphs with m edges. The algorithms are randomized and succeed with high probability.*[2]

1.1 Our Techniques In this overview, we will only consider unweighted graphs; the same ideas also extend to weighted graphs. We will use $d(X)$ to denote the *degree* of any set of vertices X, i.e., the number of edges with exactly one endpoint in X.

Supreme Sets. A key role in (global) connectivity augmentation (e.g., [48, 44, 24, 4, 5, 11, 12]) is played by *extreme sets* that are defined as follows: a set of vertices X is extreme if for every proper subset $Y \subset X$, we have $d(Y) > d(X)$. Intuitively, the extreme sets represent the "bottlenecks" in connectivity augmentation, and we want to ensure that satisfying their deficit also satisfies that of all other sets. To this end, the algorithms first perform a step called *external augmentation*. In this step, an external vertex x is added to the graph and the algorithm finds a minimum set of edges with one endpoint at x that satisfies the deficits for all the extreme sets. It is not difficult to see that these edges also satisfy the connectivity requirements for all other sets of vertices. Now, the second step of the algorithm replaces these edges by a new set of edges only on the actual vertices (i.e.,

[1] $\tilde{O}(\cdot)$ hides polylogarithmic factors.

[2] Following standard convention, *with high probability* is used to mean with probability at least $1 - n^{-c}$ for an arbitrarily large constant $c > 0$, in this paper.

not incident to x) in a careful manner so as to preserve the augmented connectivity. The prior algorithms differ in how these two steps are implemented.

For Steiner connectivity augmentation, we can similarly define extreme sets, although we now need to restrict to sets that contain at least one terminal vertex (since sets not containing terminals do not have any connectivity requirement). But, this creates more extreme sets than in the global case. Consider a set X containing terminals such that all its proper subsets Y satisfying $d(Y) \leq d(X)$ do not contain any terminals. In the global case, X would not be an extreme set but now we need to declare X as extreme since its requirement will not be met by any of its subsets. In fact, the number of extreme sets can now be exponentially large (see Fig. 3 in Appendix A for an example). In contrast, one of the key properties of extreme sets in global connectivity augmentation is that they are laminar in structure (and hence linear in number), which allows the design of efficient algorithms that find all the extreme sets.

Since we cannot find all extreme sets explicitly, we first partition them into equivalence classes based on their projection onto the terminals. Now, consider all the extreme sets with the same terminal projection. We show that the extreme sets are closed under union (but not intersection!) and therefore, we can define a unique maximum extreme set in this class. We call these maximum extreme sets the *supreme sets* and show that they form a laminar family. This also means that the terminal projections themselves also form a laminar family, and our first goal is to find this tree of terminal projections. The key property that we prove is the following:

LEMMA 1.1. *A supreme set with terminal projection R is also the* earliest *minimum-value cut that bipartitions the terminals T as R and $T \setminus R$.*

This connection allows us to design a divide and conquer algorithm that uses polylogarithmic calls to a maximum flow subroutine and returns a laminar family of vertex sets that includes all the supreme sets. Using a sequence of postprocessing steps, we eliminate the spurious sets from this laminar family and extract the tree of terminal projections. Since extreme sets (for global connectivity) are an important concept in their own right that has been studied extensively in the literature, we hope this structural and algorithmic exploration of extreme sets for Steiner connectivity and the definition of supreme sets is of independent value. We present this in Section 3.

The augmentation algorithm (Section 4) has three parts. The first part performs external augmentation on the tree of terminal projections (Section 4.1). Note that this means that all the edges we add are incident to the terminals. (This is the key property that makes the Steiner connectivity augmentation problem simpler than the Steiner splitting-off problem that we will consider later; this property will not hold in the latter problem.) We now need to replace the edges added by external augmentation with edges between terminals. We view this as a splitting-off process (Section 4.2). The splitting-off proceeds in two steps. The first step (Section 4.3) achieves Steiner connectivity of $\tau - 1$ by adapting an iterative framework of adding *augmentation chains* while respecting the vertex degrees (on the terminals) given by external augmentation [5]. To implement this framework efficiently, we adapt the use of *lazy* data structures from [11], and show that the overall implementation of this step takes only near-linear time.

Augmenting Connectivity by Random Matchings. Finally, we need to augment Steiner connectivity by one, from $\tau - 1$ to τ (Section 4.4). For global connectivity, augmenting connectivity from $\tau - 1$ to τ uses the cactus data structure for all minimum cuts which can be computed in $\tilde{O}(m)$ time [33]. For minimum Steiner cuts, however, there is no data structure that can be computed as efficiently (the only known data structure is called a *connectivity carcass* [17] whose computation is highly inefficient). To remedy this, we revisit the algorithm for augmenting global connectivity from $\tau - 1$ to τ. Consider the simple case of augmenting connectivity of a cycle on n vertices from 2 to 3. Now, suppose we iteratively add edges that form a matching with the following property: when adding an edge (u, v), ensure that they are non-adjacent on the cycle, and after adding the edge, shortcut the two neighbors of u and the two neighbors of v. We show this in Figure 4a. It is not difficult to show that this gives an optimal augmentation for the cycle.

Crucially, this algorithm can be implemented even without knowing the structure of the cycle, since a random pair of vertices is unlikely to be adjacent (except at the end). Indeed, even if we add a random matching on half the vertices instead of single edge to the cycle, it will not contain any adjacent pair with constant probability. Thus, we can implement this algorithm using $O(\log n)$ random matchings, as long as we can certify correctness of a single matching (i.e., whether it contains an adjacent pair or not). We use this intuition to design an algorithm for augmenting the (Steiner) connectivity by one. The algorithm first partitions the set of terminals based on the

minimum Steiner cuts (of value $\tau - 1$), and then iteratively adds a sequence of random matchings between the terminal sets in this partition. We also give an algorithm that can certify whether a random matching can be extended to an optimal solution, and if not, we simply discard the matching and repeat the step again. Given the interest in the literature in the special case of augmenting the connectivity by one and the simplicity of our algorithm, we believe this algorithm is of independent interest.

Steiner Splitting-Off. The Steiner splitting off algorithm is presented in Section 5. This problem can be viewed as Steiner connectivity augmentation, but where the degree of every vertex in terms of the added edges is constrained by their original degree to the vertex being split-off. This creates additional complications because the connectivity augmentation algorithm that we just sketched always adds edges to terminals whereas the splitting-off algorithm might require us to add edges to nonterminals as well. Suppose we are augmenting the connectivity of a set of extreme sets that share the same terminal projection. Earlier, we could add edges to the terminals in the projection and this benefits all the extreme sets; so we did not need the explicit extreme sets structure (and it is too expensive to compute it in any case). Now, we need to add edges incident to nonterminals, but these edges will benefit some but not all of the extreme sets with the same terminal projection. To solve the problem of adding edges to nonterminals optimally, we encode this problem as a sequence of maximum flow problems on suitably defined auxiliary graphs for each heavy path in a *heavy-light decomposition* of the supreme sets forest. The external edges are tied to the heavy paths, and this allows us to track the contribution of edges incident to nonterminals. The details of this encoding and the consequent algorithm are given in Section 5.1.

Another difficulty arises in the maintenance of the (lazy) data structures for tracking the cut values of the supreme sets while adding augmentation chains. Our data structures do not explicitly track the supreme sets, but rather their intersections with the terminals. This was not a problem in augmentation since the edges we add are always between terminals. But, now that we are forced to add edges incident to nonterminals, we do not immediately know whether a supreme set's cut value was increased by the addition of an edge. We bypass this problem by showing that there is a small subset of *representative* supreme sets such that it is sufficient for our data structures to only maintain the cut values of the representatives. The details appear in Section 5.2. Finally, we adapt our algorithm for augmenting Steiner connectivity by one using random matchings to the degree-constrained case (Section 5.3).

1.2 Related Work Connectivity augmentation problems are well-studied in many settings. Since we already mentioned the line of work for global connectivity augmentation earlier, we describe some other connectivity augmentation problems here. Jackson and Jordán [31] proposed a polynomial time algorithm for vertex connectivity augmentation in undirected graphs when the connectivity target is a constant. This result was generalized to directed graphs by Frank and Jordán [20] who gave a polynomial-time algorithm, and was further improved by Végh and Benczúr [47] to a strongly polynomial-time algorithm. Another variant of connectivity augmentation considered in the literature is that of inclusion-wise monotone connectivity targets on vertex sets (Ishii and Hagiwara [30], Ishii [29]). They showed that this problem is NP-hard in general, but gave polynomial-time algorithms when the targets are strictly greater than 1.

A popular line of work in connectivity augmentation involves restricting the set of edges to a given set, or more generally, allowing the edges to have nonuniform costs. This makes the problem NP-hard to even augment (global) connectivity by one [22]. Nevertheless, Marx and Végh [41] showed that with edge costs, it is fixed parameter tractable (FPT) to increment connectivity with the number of new edges as the parameter. In a similar vein, Klinkby et al. [34] showed an FPT result for the problem of augmenting a directed graph to make it strongly connected even with edge costs. There is also a long line of work in approximation algorithms for adding a minimum number of edges to a graph to increase the connectivity to 2. Jain's iterative rounding framework [32] yields a 2-approximation for this problem. The approximation factor of 2 was subsequently improved in a series of works for the case where the input graph is a tree [27, 8, 10, 46]. Recently, Grandoni et al. [26] gave an algorithm breaking the approximation factor of 2 even when the input graph is a forest (which is equivalent to general input graphs for this problem).

2 Preliminaries

For any nonempty vertex set $X \subsetneq V$, the cut $(X, V \setminus X)$ (or cut X in short) is the set of edges connecting X and $V \setminus X$. Denote $d(X)$ to be the cut value of X, defined as the sum of weights of edges in cut X. In particular, $d(\emptyset) = d(V) = 0$.

For any $s, t \in V$, $s \neq t$, an s-t cut is a cut (S, T) separating s and t. An s-t min cut is an s-t cut with minimum value among all s-t cuts. Denote $\lambda(s, t)$ to be the value of the s-t min cut. Among the (possibly many) s-t min cuts, the one whose s-side is minimal (as a vertex set) is called the earliest s-t min cut, and the one whose t-side is minimal (or equivalently, maximal in s-side) is called the latest s-t min cut. (They are unique because the intersection and union of two s-t min cuts are also s-t min cuts.)

For disjoint vertex sets X and Y, similarly define an X-Y min cut to be the minimum cut separating X and Y, and $\lambda(X, Y)$ be the value of X-Y min cut. The earliest (resp. latest) X-Y min cut is the X-Y min cut whose X-side (resp. Y-side) is minimal. These notations about X-Y cuts can also be understood as the corresponding notations of s-t cuts by contracting X and Y to be s and t.

We now define two key terms that we use throughout the paper: crossing and submodularity.

Definition 2.1. (Crossing sets) *We say that vertex sets X and Y* cross *if $X \cap Y$, $X \setminus Y$ and $Y \setminus X$ are all non-empty.*

A family of sets is laminar *if no two sets in the family cross. Such a family can be represented by a collection of disjoint rooted trees, where each node corresponds to a set in the family, so that (the set corresponding to) an ancestor node of a tree contains (the set corresponding to) a descendant node, and two nodes that are incomparable in a single tree (or are in different trees) are disjoint sets. Formally, for each set R in the family, the node of R is a root iff it is maximal in the family, and R is a child of X iff $R \subseteq X$ and there is no other set Y in the family such that $R \subseteq Y \subseteq X$.*

Fact 2.1. (Submodularity and posi-modularity of cuts) *For any vertex sets X and Y,*

$$d(X \cap Y) + d(X \cup Y) \leq d(X) + d(Y) \tag{2.1}$$

$$d(X \setminus Y) + d(Y \setminus X) \leq d(X) + d(Y) \tag{2.2}$$

The following immediate corollaries of (2.1) *will be useful.*

$$d(X \cap Y) \geq d(X) \implies d(X \cup Y) \leq d(Y) \tag{2.3}$$

$$d(X \cap Y) > d(X) \implies d(X \cup Y) < d(Y) \tag{2.4}$$

2.1 Proof of Tractability Before discussing our efficient algorithm, we first observe that polynomial-time tractability of the Steiner connectivity augmentation and splitting-off problems follow easily from previous works. Indeed, these problems are special cases of pairwise connectivity augmentation and splitting off problems, for which we give short proofs of polynomial tractability based on previous work below.

For the splitting off problem preserving pairwise connectivities, the running time of the current best algorithm by Lau and Yung [36] is dominated by n calls to a Gomory-Hu tree subroutine. Although the result stated in Lemma 2.1 is for unweighted multigraphs, the same algorithm works for integer-weighted graphs if we conceptually regard an edge of weight w as w parallel edges.

Lemma 2.1. ([36]) *There exists an algorithm that, given an unweighted undirected graph with a special vertex x to be split off where $d(x)$ is even and there is no cut edge incident to x, solves splitting off preserving all-pairs connectivities of $V \setminus \{x\}$ in $O(n)$ attempts. Each attempt picks a pair of vertices (u, v) and splits (x, u) and (x, v) for some amount, and the amount can be calculated by a Gomory-Hu tree subroutine.*

Using the $\tilde{O}(n^2)$ time Gomory-Hu tree algorithm [1], we can solve Steiner splitting-off as a special case of splitting-off preserving all pairwise connectivities in $\tilde{O}(n^3)$ time.

For the Steiner connectivity augmentation problem, we first perform external augmentation, and then split-off the external augmentation solution to get a solution to the original problem. This transformation is discussed in Section 4.2. Here, we show in Lemma 2.4 that the external augmentation solution can be computed in $\tilde{O}(n^3)$ time using prior work.

We introduce a greedy external augmentation algorithm proposed by Frank [18]. It works for the all-pairs connectivity setting, where every pair of vertices $u, v \in V \setminus \{x\}$ has a u-v connectivity requirement. In external

augmentation, we add a new vertex x. First, we add sufficiently many edges (called external edges) from every vertex to the external vertex x, so that the connectivity requirement is satisfied. Then, we repeatedly remove any external edge as long as the connectivity requirement is satisfied, until no external edge can be removed. The remaining external edges form a feasible external augmentation solution.

LEMMA 2.2. (LEMMA 5.6 OF [18]) *If the optimal value of an all-pairs connectivity augmentation instance is γ, and no vertex pair has connectivity requirement 1, then the greedy external augmentation algorithm outputs a feasible external augmentation solution with total weight at most 2γ.*

LEMMA 2.3. *The greedy external augmentation algorithm can be implemented in $\tilde{O}(n^3)$ time.*

Proof. In the removal stage, we process one vertex at a time. Each iteration picks a vertex v, and removes the maximum number of (v, x) edges such that no connectivity requirement is violated. After vertex v has been processed, none of the (v, x) edges can be removed without violating a connectivity requirement in the future. This property holds irrespective of what other edges are removed in the future, because removing other edges cannot increase cut values. Therefore, it suffices to process every vertex v exactly once, and the algorithm terminates in n iterations.

In a single iteration, for a vertex v, we need to decide the maximum number of removable (v, x) edges. To this end, we compute all-pairs min-cuts in the graph after removing all (v, x) edges. Suppose there are w parallel (v, x) edges. Let Δ be the maximum violation of a connectivity requirement after removing these w edges. That is, Δ is the maximum difference between the (s, t)-connectivity requirement and the value of the (s, t) min-cut after removing the (v, x) edges, among all vertex pairs s, t. Since all cut values satisfy the connectivity requirements before removing the (v, x) edges, the (v, x) edges must cross all violated cuts. Therefore, by adding back Δ parallel (v, x) edges, all violations can be resolved. In other words, removing $w - \Delta$ edges is feasible. Moreover, removing more than $w - \Delta$ edges is infeasible because adding back less than Δ edges cannot resolve the maximum violation of Δ. Therefore the maximum number of removable edges is $w - \Delta$.

The algorithm runs n iterations. Each iteration calls an all-pairs min-cuts subroutine, which takes $\tilde{O}(n^2)$ time [1]. The total running time is $\tilde{O}(n^3)$. ∎

LEMMA 2.4. *The Steiner connectivity augmentation problem can be solved in $\tilde{O}(n^3)$ time.*

Proof. When $\tau = 1$ the optimal solution is simply a tree over the connected components containing terminals. The main case is $\tau \geq 2$. Assume the optimal solution has total weight γ.

Run the greedy external augmentation algorithm to get an external augmentation solution F^{ext}. (To convert Steiner connectivity augmentation into all-pairs connectivity augmentation, set the connectivity requirement for all pairs of terminals to τ and that for all other pairs to 0.) By Lemma 2.2, the total weight of F^{ext} is at most 2γ. If the total weight is odd, we increase the weight of an arbitrary edge in the solution by 1 so that the total weight is even and still at most 2γ. Adding these edges increases the Steiner connectivity to at least τ.

Then, we split-off the external vertex using Lemma 2.1 to get an edge set F with total weight at most γ. (There is no cut edge incident to the external vertex because the Steiner connectivity is at least 2.) Splitting-off preserves pairwise connectivity; therefore, the Steiner connectivity is still at least τ. Thus, F is a feasible solution to the Steiner connectivity augmentation problem, and its total weight is no more than the optimal value γ. Hence, F it is an optimal solution.

The running time is $\tilde{O}(n^3)$ in both steps by Lemmas 2.1 and 2.3. Therefore, the total running time is $\tilde{O}(n^3)$. ∎

3 Extreme Sets for Steiner Connectivity

For a graph G on vertex set V and a set $T \subseteq V$ of terminals, we define (one side of) a *Steiner cut* as a vertex set X such that $X \cap T \neq \emptyset$ and $T \not\subseteq X$. The concept of *extreme sets* in global connectivity setting can be generalized in the following natural way.

DEFINITION 3.1. (EXTREME SETS) *A Steiner cut X is* extreme *if and only if for any Steiner cut Y that is also a proper subset of X, $d(Y) > d(X)$.*

If a Steiner cut X is not extreme, by definition there exists a Steiner cut $Z \subsetneqq X$ with $d(Z) \leq d(X)$. Such a set Z is called a *violator*.

Let $\mathcal{X}$ be the family of all extreme sets in G. The following facts are immediate from the definition.

FACT 3.1. *If X is an extreme set and $Y \subseteq X$ is a Steiner cut, then $d(Y) \geq d(X)$, and equality only holds when $Y = X$.*

FACT 3.2. *If a Steiner cut X is not extreme, then there exists a violator of X that is extreme.*

Proof. Let Z be the violator of X of minimum size. By definition, $d(Z) \leq d(X)$. We prove that Z is extreme, which implies the statement.

Let W be any Steiner cut that is a proper subset of Z. Because $|W| < |Z|$ and Z is the minimum-size violator of X, W is not a violator of X. That is, $d(W) > d(X) \geq d(Z)$. Therefore Z is extreme by Definition 3.1. ∎

Finally, we define the projection of a Steiner cut to be the set of terminals in it.

DEFINITION 3.2. *Define ρ to be the* projection *from vertex sets to terminal sets. For any $X \subseteq V$, $\rho(X) = X \cap T$.*

REMARK 3.1. *X is a Steiner cut if and only if $\rho(X) \neq \emptyset$ and $\rho(X) \neq T$.*

3.1 Structure of Extreme Sets and Supreme Sets In the Steiner setting, the structure of extreme sets is much weaker than in the global connectivity setting. Most importantly, we are no longer guaranteed that extreme sets never cross. Fortunately, the extreme sets are still closed under set union, which we establish in the following chain of lemmas.

LEMMA 3.1. *If extreme sets X and Y cross, then at least one of $X \setminus Y$ or $Y \setminus X$ contains no terminal.*

Proof. Assume for contradiction that both $X \setminus Y$ and $Y \setminus X$ contain terminals. Then they are Steiner cuts. Because X and Y cross, $X \setminus Y$ is a proper subset of X, and $Y \setminus X$ is a proper subset of Y. Then $d(X \setminus Y) > d(X)$ and $d(Y \setminus X) > d(Y)$ since X and Y are extreme (Definition 3.1). Adding these two inequalities gives

$$d(X \setminus Y) + d(Y \setminus X) > d(X) + d(Y)$$

which contradicts (2.2) of Fact 2.1. ∎

LEMMA 3.2. *For any extreme sets X and Y, if $X \cap Y$ is nonempty, then $X \cap Y$ is a Steiner cut.*

Proof. When $X \subseteq Y$ (resp. $Y \subseteq X$), $X \cap Y$ is a Steiner cut because it is X (resp. Y). The remaining case is that both $X \setminus Y$ and $Y \setminus X$ are nonempty. The statement also assumes $X \cap Y$ is nonempty, so X and Y cross. By Lemma 3.1, one of $X \setminus Y$ or $Y \setminus X$ contains no terminal. Without loss of generality, assume $\rho(X \setminus Y) = \emptyset$. Then

$$\rho(X \cap Y) = \rho(X \setminus (X \setminus Y)) = \rho(X) \setminus \rho(X \setminus Y) = \rho(X)$$

Because X is a Steiner cut and $\rho(X \cap Y) = \rho(X)$, $X \cap Y$ is also a Steiner cut. ∎

LEMMA 3.3. *If two extreme sets X and Y cross and $X \cup Y$ is a Steiner cut, then $X \cup Y$ is extreme.*

Proof. Assume for contradiction that there exist crossing extreme sets X and Y such that $X \cup Y$ is a Steiner cut but not extreme. By Fact 3.2, there exists an extreme violator of $X \cup Y$. Let Z be the one with minimum cut value among all extreme violators of $X \cup Y$.

$$d(Z) \leq d(X \cup Y) \tag{3.5}$$

Because X, Y cross, $X \cap Y \neq \emptyset$. By Lemma 3.2, $X \cap Y$ is a Steiner cut. Again because X, Y cross, both $X \setminus Y$ and $Y \setminus X$ are non-empty, so $X \cap Y$ is a proper subset of X and Y. Since X and Y are extreme (Definition 3.1),

$$d(X \cap Y) > \max\{d(X), d(Y)\}$$

Applying (2.4) of Fact 2.1 on X and Y, we have

$$d(X \cup Y) < \min\{d(X), d(Y)\} \tag{3.6}$$

(3.5) and (3.6) imply $d(Z) < \min\{d(X), d(Y)\}$, which means Z cannot be a subset of X or Y by Fact 3.1. Therefore $X \cap Z \neq \emptyset,\ Y \cap Z \neq \emptyset$. By Lemma 3.2, $X \cap Z$ and $Y \cap Z$ are Steiner cuts, so

$$d(X \cap Z) \geq d(X), d(Y \cap Z) \geq d(Y)$$

by Fact 3.1. Moreover, one of the two inequalities must be strict. Because Z is a proper subset of $X \cup Y$, $X \cap Z = X$ and $Y \cap Z = Y$ cannot hold simultaneously. Without loss of generality, assume $Y \cap Z \subsetneq Y$; then since Y is extreme,

$$d(Y \cap Z) > d(Y)$$

Applying (2.3) of Fact 2.1 on X and Z, and (2.4) on Y and Z, we have

$$d(X \cup Z) \leq d(Z) \tag{3.7}$$

$$d(Y \cup Z) < d(Z) \tag{3.8}$$

Adding (3.7) and (3.8) and using (3.5) give

$$d(X \cup Z) + d(Y \cup Z) < d(Z) + d(Z) \leq d(Z) + d(X \cup Y) \tag{3.9}$$

(2.1) of Fact 2.1 on $X \cup Z$ and $Y \cup Z$ says

$$d(X \cup Z) + d(Y \cup Z) \geq d((X \cup Z) \cap (Y \cup Z)) + d((X \cup Z) \cup (Y \cup Z)) = d((X \cap Y) \cup Z) + d(X \cup Y) \tag{3.10}$$

Finally, (3.9) and (3.10) imply $d((X \cap Y) \cup Z) < d(Z)$.

If $(X \cap Y) \cup Z$ is extreme, let $Z' = (X \cap Y) \cup Z$; else let Z' be an extreme violator of $(X \cap Y) \cup Z$ (which exists by Fact 3.2). Then $d(Z') \leq d((X \cap Y) \cup Z) < d(Z) \leq d(X \cup Y)$, $Z' \subseteq (X \cap Y) \cup Z \subseteq X \cup Y$, so Z' is an extreme violator of $X \cup Y$ with cut value less than Z, which contradicts the assumption that Z has minimum cut value among all extreme violators of $X \cup Y$. ∎

Remark 3.2. *We note that $X \cap Y$ is not necessarily extreme, unlike in the global connectivity setting.*

Since extreme sets are closed under set union, the concept of maximal extreme sets is well-defined. This motivates one of the key new concepts of the paper, the supreme sets, which we show form a laminar family.

Definition 3.3. (Supreme sets) *Partition the extreme sets into equivalence classes according to projection: two extreme sets X and Y are equivalent if their projections $X \cap T$ and $Y \cap T$ are equal.*

For each equivalence class with a common terminal set R, define the supreme set *of R, denoted $\mu(R)$, to be the union of all extreme sets in the equivalence class, i.e. $\mu(R) = \bigcup_{X \in \mathcal{X}: \rho(X) = R} X$. The set $\mu(R)$ is defined only if $R \subseteq T$ is the projection of some extreme set; otherwise we say $\mu(R)$ is undefined. In particular, $\mu(\emptyset)$ and $\mu(T)$ are undefined.*

Fact 3.3. *If $\mu(R)$ is defined, then $\mu(R)$ is an extreme set and $\rho(\mu(R)) = R$.*

Proof. $R \subsetneq T$ because $\mu(R)$ is defined. By definition, $\mu(R)$ is the union of extreme sets $\{X_1, \ldots, X_r\}$, where $\rho(X_i) = R$ for each X_i. For any $1 \leq k \leq r$, let $Y_k = \bigcup_{i=1}^{k} X_i$. By a simple inductive argument, $\rho(Y_k) = R$ and Y_k is a Steiner cut for each k.

We prove by induction that Y_k is extreme. The base case $k = 1$ is trivial. As an induction step, assume Y_k is extreme. If $Y_k \subseteq X_{k+1}$ (resp. $X_{k+1} \subseteq Y_k$), $Y_{k+1} = X_{k+1}$ (resp. $Y_{k+1} = Y_k$) is an extreme set. The remaining case is that both $Y_k \setminus X_{k+1}$ and $X_{k+1} \setminus Y_k$ are nonempty. $Y_k \cap X_{k+1} \supseteq R$ is also nonempty, so Y_k and X_{k+1} cross. We have shown Y_{k+1} is a Steiner cut. By Lemma 3.3, $Y_{k+1} = Y_k \cup X_{k+1}$ is extreme.

In conclusion, $\mu(R) = Y_r$ is extreme. ∎

LEMMA 3.4. *The supreme sets form a laminar family.*

Proof. Assume for contradiction that two supreme sets $X_1 = \mu(R_1)$ and $X_2 = \mu(R_2)$ cross. By Fact 3.3, X_1 and X_2 are extreme sets. By Lemma 3.1, one of $X_1 \setminus X_2$ and $X_2 \setminus X_1$ contains no terminal. Therefore $R_1 \subseteq R_2$ or $R_2 \subseteq R_1$. Without loss of generality, assume $R_1 \subseteq R_2$. Then

$$\rho(X_1 \cup X_2) = \rho(X_1) \cup \rho(X_2) = R_1 \cup R_2 = R_2.$$

By Definition 3.3 of $\mu(R_2)$, $X_1 \cup X_2 \subseteq \mu(R_2) = X_2$, which contradicts the assumption that X_1 crosses X_2.

There is no crossing supreme sets, so they form a laminar family. ∎

Finally, we show that supreme sets can be efficiently computed as long as we know their projections, a key ingredient in our supreme sets algorithm.

LEMMA 3.5. *For any supreme set $X = \mu(R)$, X is the earliest R-$(T \setminus R)$ mincut.*

Proof. Because $\mu(R)$ is defined, $R \neq \emptyset$ and $R \neq T$. Let S be the R-side of the earliest R-$(T \setminus R)$ min cut, which is a Steiner cut because $\rho(S) = R$. Because X is an R-$(T \setminus R)$ cut and S is an R-$(T \setminus R)$ min cut,

$$d(S) \leq d(X) \tag{3.11}$$

First we prove that S is extreme. Assume otherwise; then by Fact 3.2, there exists an extreme violator $Z \subsetneqq S$ such that

$$d(Z) \leq d(S) \tag{3.12}$$

Because $Z \subsetneqq S$, $\rho(Z) \subseteq \rho(S) = R$. We divide into two cases $\rho(Z) = R$ and $\rho(Z) \subsetneqq R$ and derive a contradiction for both cases.

When $\rho(Z) = R$, Z is an R-$(T \setminus R)$ cut, and also a proper subset of the earliest R-$(T \setminus R)$ min cut S. So $d(Z) > d(S)$, which contradicts (3.12).

The remaining case is $\rho(Z) \subsetneqq R$. Let $Y = \mu(\rho(Z))$. Because Z is an extreme set, $Z \subseteq Y$ by Definition 3.3. Using Fact 3.1,

$$d(Z) \geq d(Y) \tag{3.13}$$

By Lemma 3.4, $Y = \mu(\rho(Z))$ and $X = \mu(R)$ cannot cross. Because $X \setminus Y \supseteq R \setminus \rho(Z) \neq \emptyset$ and $X \cap Y \supseteq \rho(Z) \neq \emptyset$, we have $Y \setminus X = \emptyset$ and $Y \subsetneqq X$. Since X is extreme,

$$d(Y) > d(X) \tag{3.14}$$

(3.13) and (3.14) give $d(Z) > d(X)$, but (3.11) and (3.12) give $d(Z) \leq d(X)$, a contradiction. In conclusion, S is extreme.

By Definition 3.3, $S \subseteq \mu(R) = X$. $S \subsetneqq X$ would imply $d(S) > d(X)$ since X is extreme, which contradicts (3.11). Therefore $S = X$. ∎

LEMMA 3.6. *For any supreme set $X = \mu(R)$ in G, and any set K which is either a subset of X or disjoint from X, consider a graph H by contracting K from G. Then X's image X' is still a supreme set in H.*

Proof. The contraction homomorphism φ maps vertices in K to a contracted node k, and maps vertices outside K to themselves. Define its inverse homomorphism φ^{-1} that maps k to K and vertices outside K to themselves. We have $d_H(Z) = d(\varphi^{-1}(Z))$ for any vertices set Z in H. In particular, the assumption means $X = \varphi^{-1}(X')$, so $d_H(X') = d(X)$.

Define extreme sets and supreme sets in H with terminal set $\varphi(T)$. Let $R' = X' \cap \varphi(T)$. Our goal is to prove that $X' = \mu_H(R')$.

First we prove that X' is extreme in H. Intuitively, by contracting a subset or disjoint set, the extreme condition of X can only become weaker. Formally, for any Steiner cut $Z' \subsetneqq X'$, let $Z = \varphi^{-1}(Z')$, so that $Z \subsetneqq X$.

Because Z' is a Steiner cut in H, $\rho_H(Z')$ is not empty nor $\varphi(T)$, so $\rho(Z) = \varphi^{-1}(\rho_H(Z'))$ is not empty nor T. That is, Z is a Steiner cut and also a proper subset of X. Because X is extreme,

$$d_H(Z') = d(Z) > d(X) = d_H(X')$$

So X' is extreme in H by Definition 3.1. By Definition 3.3 of $\mu_H(R')$, $X' \subseteq \mu_H(R')$.

Assume for contradiction that $X' \subsetneq \mu_H(R')$. Let $Y = \varphi^{-1}(\mu_H(R'))$, so that $X \subsetneq Y$. If there exists a terminal $t \in Y \setminus X$, then t is mapped to R', which means t and an vertex in X have the same image. However, this is impossible because K is either contained in X or disjoint from X. Therefore $\rho(Y) = \rho(X) = R$.

Because $\mu_H(R')$ is extreme in H,

$$d(X) = d_H(X') > d_H(\mu_H(R')) = d(Y) \tag{3.15}$$

If Y is extreme, then Y is a subset of $X = \mu(R)$ and $d(Y) \leq d(X)$, which contradicts (3.15). Therefore Y is not extreme and there exists a violator Z of Y with

$$d(Z) \leq d(Y) \tag{3.16}$$

Because $\rho(Z) \subseteq \rho(Y) = R$ and Z is extreme, $Z \subseteq \mu(\rho(Z)) \subseteq X$. Then $d(Z) \leq d(X)$ by Fact 3.1. However, (3.15) and (3.16) give $d(X) > d(Z)$, a contradiction. In conclusion, $X' = \mu_H(R')$. ∎

3.2 Algorithm for Finding Supreme Sets This section presents an algorithm that constructs a laminar forest L representing the terminal sets of all extreme sets, and calculates $c(T_i) = d(\mu(T_i))$ for each $T_i \in L$ as a byproduct. This is a little weaker than the laminar forest of supreme sets guaranteed by Lemma 3.4. Instead of finding all supreme sets $\mu(T_i)$, we only find their terminal sets T_i and cut values $c(T_i)$, which are enough for our augmentation algorithm.

The algorithm consists of three phases. Phase 1 performs perturbation on edge weights to make cut values distinct, but also distorts the supreme sets. Phase 2 applies a divide and conquer algorithm to find a laminar forest containing all supreme sets of the perturbed graph. Finally in Phase 3, several rounds of post-processing extract the laminar forest on terminals of original graph from the laminar forest of supreme sets of perturbed graph.

3.2.1 Phase 1: Perturbation We apply the same perturbation used in [11]. That is, for each edge (u, v), let the new weight $\widetilde{w}(u, v) = mN \cdot w(u, v) + r(u, v)$, where N is an integer larger than the sum of all edge weights but still polynomial in n, and $r(u, v)$ are uniformly and independently drawn from $\{1, 2, \ldots, N\}$. Let $\widetilde{G}$ be the graph after perturbation. Use $\widetilde{d}(\cdot)$ and $\widetilde{\lambda}(\cdot, \cdot)$ to denote cut values under edge weights $\widetilde{w}$.

Phase 1 takes $O(m)$ time to generate the random numbers.

FACT 3.4. ([11]) *Perturbation does not change order of two unequal cut values.*

FACT 3.5. *If X is an extreme set under edge weights w, then X remains extreme under edge weights $\widetilde{w}$.*

Proof. For every Steiner cut Y that is a proper subset of X, we have $d(Y) > d(X)$ by Definition 3.1. By Fact 3.4, $\widetilde{d}(Y) > \widetilde{d}(X)$. Therefore X satisfies Definition 3.1 under edge weights $\widetilde{w}$. ∎

LEMMA 3.7. ([11]) *After the perturbation, the following holds with high probability. Fix any vertex s. For every vertex $t \neq s$, the minimum s-t cut under $\widetilde{w}$ is unique. Moreover, for every vertex pair $t, t' \in V \setminus \{s\}$, either $\widetilde{\lambda}(s, t) \neq \widetilde{\lambda}(s, t')$, or the unique minimum s-t cut is identical to the unique minimum s-t' cut under $\widetilde{w}$.*

3.2.2 Phase 2: Divide and Conquer This section works on the perturbed graph $\widetilde{G}$ assuming the properties of Lemma 3.7 hold. We remark that for simplicity, the reader can assume that the original graph G has all s-t min cuts unique, and simply pretend that no perturbation exists, i.e. $\widetilde{G} = G$.

We present in Algorithm 1 a divide and conquer algorithm that outputs a laminar family containing all supreme sets of $\widetilde{G}$. The algorithm is based on uncrossing property about cut thresholds (Lemma 3.8).

Algorithm 1: Find the laminar forest of supreme sets.

Input : Graph $\widetilde{G} = (V, E)$ with edge weights $\widetilde{w}$, terminal set T.

1 **if** $|T| \leq 16$ **then**
2 Calculate the earliest T_1-T_2 min cut for every bipartition (T_1, T_2) of T.
3 For any pair of crossing sets (X, Y) in the family, either $d(X \setminus Y) \leq d(X)$ or $d(Y \setminus X) \leq d(Y)$ by (2.2). In the former case remove X, and in the latter case remove Y.
4 Return the laminar forest of remaining sets.
5 **else**
6 **repeat**
7 Uniformly sample two terminals $s, t \in T$. Let $\phi = \tilde{\lambda}(s, t)$.
8 Compute $T_1 = ct(s, \phi) \cap T$. Let $T_2 = T \setminus T_1$.
9 **until** $|T_1| \geq \frac{1}{16}|T|, |T_2| \geq \frac{1}{16}|T|$;
10 Compute the earliest T_1-T_2 min cut. Let the two sides of the min cut be S_1 and S_2 where $T_1 \subseteq S_1$, $T_2 \subseteq S_2$.
11 Form $\widetilde{G}_1$ by contracting S_1 in $\widetilde{G}$. Let t_1 be the contracted node of S_1. Recursively run the algorithm on input $(\widetilde{G}_1, T_2 \cup \{t_1\})$. Let L_1 be the returned laminar forest.
12 Form $\widetilde{G}_2$ by contracting S_2 into t_2. Recurse on $(\widetilde{G}_2, T_1 \cup \{t_2\})$ to get laminar forest L_2.
13 After deleting the leaf of $\{t_1\}$ in the forest L_1, attach all trees in L_1 to be children of the leaf of $\{t_2\}$ in L_2.
14 Output the merged laminar forest.

DEFINITION 3.4. *The cut threshold of source s and threshold ϕ is defined as*

$$ct(s, \phi) = \{t \in V : \widetilde{\lambda}(s, t) \geq \phi\}.$$

We also use the convention that $s \in ct(s, \phi)$.

Given graph $\widetilde{G}$, the algorithm samples two terminals $s, t \in T$ uniformly, and uses cut threshold $ct(s, \tilde{\lambda}(s, t))$ to partition the terminals into T_1 and T_2. By Lemma 3.13 proved later on, this partition is balanced on terminals with constant probability, which is crucial for efficiency. Let (S_1, S_2) be the earliest T_1-T_2 min cut. We form two subproblems by contracting S_1 in one and S_2 in the other. Lemma 3.9 guarantees that the cut (S_1, S_2) does not cross any supreme set. Then for any supreme set X, one of S_1 and S_2 is a subset of X or disjoint from X, and X remains supreme in one of the recursive calls by Lemma 3.6. Therefore, all supreme sets can be found in the recursive calls.

The recursion reaches the base case when $|T| \leq 16$, which is still nontrivial because the number of non-terminals is unbounded. By Lemma 3.5, every supreme set is the earliest min cut of a subset of T, so finding all $2^{|T|-1}$ such earliest min cuts produces a family containing all supreme sets. However, this family may not be laminar. Fortunately, for any pair of crossing sets (X, Y), either $d(X \setminus Y) \leq d(X)$ or $d(Y \setminus X) \leq d(Y)$ by (2.2), which means one of X and Y is violated by its subset and hence not extreme. By checking all pairs of sets in the family and remove a non-extreme set in each crossing pair, we get a laminar family containing all supreme sets.

Finally, we merge the two laminar forests returned by the recursive calls in Line 13, without losing any supreme sets. By attaching the forest L_1 to the leaf of $\{t_2\}$ in L_2 and deleting the leaf of $\{t_1\}$, we preserve all sets in L_1 and L_2, except the sets in L_1 that contain t_1. Lemma 3.9 guarantees that these sets cannot be supreme, so we do not lose any supreme set in the merging step, and the final output is a laminar forest containing all supreme sets of $\widetilde{G}$.

LEMMA 3.8. *For any cut threshold $X = ct(s, \phi)$ and any extreme set Y, either $Y \setminus X$ contains no terminal, or at least one of $X \cap Y$ and $X \setminus Y$ is empty.*

Proof. Assume for contradiction that there exists $X = ct(s, \phi)$ and extreme Y such that $\rho(Y \setminus X) \neq \emptyset$, $X \cap Y \neq \emptyset$, $X \setminus Y \neq \emptyset$. Pick $u \in X \setminus Y$ if $s \in Y$, and pick $u \in X \cap Y$ if $s \notin Y$. Then Y separates s from $u \in X$. Also pick

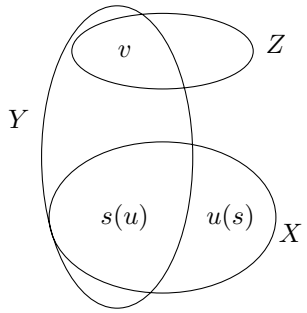


Figure 1: Proof of Lemma 3.8

terminal $v \in \rho(Y \setminus X)$. By Definition 3.4 on X,

$$\widetilde{\lambda}(s,u) \geq \phi, \tag{3.17}$$

$$\widetilde{\lambda}(s,v) < \phi. \tag{3.18}$$

Let Z be the v-side of the s-v min cut (which is unique by Lemma 3.7).

$$\widetilde{d}(Z) = \widetilde{\lambda}(s,v) < \phi \tag{3.19}$$

Then for all $z \in Z$, $\widetilde{\lambda}(s,z) \leq \widetilde{d}(Z) < \phi$ and $z \notin X$ by Definition 3.4 on X, so $Z \cap X = \emptyset$. Therefore, $Y \cup Z$ is a s-u cut, and

$$\widetilde{d}(Y \cup Z) \geq \widetilde{\lambda}(s,u) \tag{3.20}$$

$Y \cap Z$ is a proper subset of Y containing terminal v, and since Y is extreme, $\widetilde{d}(Y \cap Z) > \widetilde{d}(Y)$. Use (2.4) of Fact 2.1 and then (3.19),

$$\widetilde{d}(Y \cup Z) < \widetilde{d}(Z) < \phi \tag{3.21}$$

But (3.17) and (3.20) give $\widetilde{d}(Y \cup Z) \geq \phi$, which contradicts (3.21). ∎

Corollary 3.1. *For any cut threshold X and any extreme set Y, $\rho(X)$ does not cross $\rho(Y)$.*

Remark 3.3. *Recall that in the setting of global min cuts, a cut threshold does not cross any extreme set. However, in Steiner min cuts setting, it is possible that a cut threshold crosses an extreme set in non-terminals, so Corollary 3.1 only gives the weaker property of uncrossing in terminal.*

Lemma 3.9. *Suppose $ct(s,\phi)$ partitions the terminals into two nonempty sides $T_1 = \{t \in T : \widetilde{\lambda}(s,t) \geq \phi\}$ and $T_2 = \{t \in T : \widetilde{\lambda}(s,t) < \phi\}$. Let S be the T_1-side of the earliest T_1-T_2 mincut. Then S does not cross any supreme set.*

Proof. Assume for contradiction that S crosses some supreme set $X = \widetilde{\mu}(R)$. By Corollary 3.1, R does not cross T_1, which is the terminal set of a cut threshold. Then there are three cases of the relation between R and T_1:

(1) $R \subseteq T_1$. Then $\rho(X \cap S) = R$ and $X \cap S$ is a Steiner cut. $X \cap S$ is also a proper subset of X because X and S cross. Since X is extreme, $\widetilde{d}(X \cap S) > \widetilde{d}(X)$. Using (2.4) of Fact 2.1, $\widetilde{d}(X \cup S) < \widetilde{d}(S)$. This contradicts the the fact that S is a T_1-T_2 min cut because $X \cup S$ is a T_1-T_2 cut.

(2) $R \cap T_1 = \emptyset$. Then $\rho(X \setminus S) = R$ and $X \setminus S$ is a Steiner cut. $X \setminus S$ is also a proper subset of X because X and S cross. Since X is extreme, $\widetilde{d}(X \setminus S) > \widetilde{d}(X)$. Using (2.2) of Fact 2.1, $\widetilde{d}(S \setminus X) < \widetilde{d}(S)$. This contradicts the fact that S is a T_1-T_2 min cut because $S \setminus X$ is a T_1-T_2 cut.

(3) $T_1 \subsetneqq R$. $X \cap S$ is a proper subset of S because X and S cross, and also a T_1-T_2 cut. Because S is the earliest T_1-T_2 cut, $\widetilde{d}(X \cap S) > \widetilde{d}(S)$. Using (2.4) of Fact 2.1,

$$\widetilde{d}(X \cup S) < \widetilde{d}(X) \tag{3.22}$$

By Lemma 3.5, X is an R-$(T \setminus R)$ min cut. Notice that $X \cup S$ is a R-$(T \setminus R)$ cut. Therefore, $\widetilde{d}(X \cup S) \geq \widetilde{d}(X)$, which contradicts (3.22).

All three cases derive contradiction, so S does not cross any supreme set. ∎

LEMMA 3.10. *Algorithm 1 outputs a laminar forest of Steiner cuts and the family contains all supreme sets of* $\widetilde{G}$.

Proof. First we prove that all sets in the output family are Steiner cuts. In the base case, we only calculate min cuts between terminal sets. In the recursive case, the output is merged from the output of subproblems. So the algorithm only outputs Steiner cuts.

Next we prove that every supreme set is in the output by induction on recursion depth. In the base case, we find all sets in the form of T_1-T_2 earliest min cut, which contains all supreme sets by Lemma 3.5. Line 3 removes a set X only when X crosses Y and $d(X \setminus Y) \leq d(X)$. If $X \setminus Y$ is a Steiner cut, then it is a violator of X. If $\rho(X \setminus Y) = \emptyset$, then $\rho(X \cup Y) = \rho(Y)$, and $d(X \cup Y) \geq d(Y)$ because Y is a min cut between terminals. By (2.1), $d(X \cap Y) \leq d(X)$, and $X \cap Y$ is a violator of X. In both cases X does not satisfy the extreme property, so the supreme sets are not removed.

As an inductive step, consider any supreme set X and its relation to the cut S_1 in Line 10 of Algorithm 1. By Lemma 3.9, one of the following holds: (a) $X \subseteq S_1$, (b) $X \supseteq S_1$, or (c) $X \cap S_1 = \emptyset$. By Lemma 3.6, in cases (b) and (c) X is a supreme set in the subproblem contracting S_1, and in case (a) X is a supreme set in the subproblem contracting S_2. The two subproblems return two laminar forests L_1 and L_2. Using the induction hypothesis, X is found in L_1 for cases (b) and (c), and found in L_2 for case (a).

In the merge step (Line 13), we form a merged laminar forest L by attaching L_1 to the leaf of contracted node in L_2, and deleting the leaf of contracted node of L_1. For case (a), X is a set in L_2 disjoint from the contracted S_2, so X is preserved in L. For case (b), X is a set in L_1 containing the contracted S_1, and is preserved in L because we expand the contracted node of L_1 to be S_1. Finally for case (c), X is a set in L_2 disjoint from the contracted S_1, so X is preserved in L. In conclusion the merge step does not lose supreme sets. The merged forest represents a laminar family because what we do is just expanding the contracted S_1 by a subtree of subsets of S_1.

By induction, the overall merged laminar forest contains all supreme sets of $\widetilde{G}$. ∎

Next we analyze the running time of Algorithm 1. The analysis has the same spirit as [11]. We first show the recursion depth is $O(\log n)$, then bound the running time of each recursive call to be poly-logarithmic many max flows on the contracted graph of the subproblem, and finally prove that the total size of subproblems in each recursion layer is $\tilde{O}(m)$ edges and $\tilde{O}(n)$ vertices. It follows that the total running time is $\tilde{O}(F(m,n))$.

LEMMA 3.11. ([38]) *Given a graph* $G = (V, E)$*, a source vertex* $s \in V$ *and an integer* $\phi > 0$*, cut threshold* $ct(s, \phi)$ *can be computed in* $\tilde{O}(F(m,n))$ *time with high probability.*

LEMMA 3.12. ([2]) *For any graph* $G = (V, E)$ *and terminal set* $T \subseteq V$*, there exists* $|T|/2$ *nodes* $u \in T$ *such that at least* $|T|/4$ *other nodes* $w \in T \setminus \{u\}$ *satisfy* $|U \cap T| > |W \cap T|$*, where* (U, W) *is the u-w min cut. (If not unique, choose the same cut for u-w min cut and w-u min cut.)*

LEMMA 3.13. *Uniformly sample two terminals* $s, t \in T$ *and let* $\phi = \widetilde{\lambda}(s,t)$*. After perturbation, use cut threshold* $ct(s, \phi)$ *to partition terminals into two sides* $T_1 = \{t \in T : \widetilde{\lambda}(s,t) \geq \phi\}$ *and* $T_2 = \{t \in T : \widetilde{\lambda}(s,t) < \phi\}$*. Then with constant probability,* $|T_1| \geq \frac{1}{16}|T|, |T_2| \geq \frac{1}{16}|T|$.

Proof. Uniformly sample $s \in T$. Assume the property of Lemma 3.7 holds, which happens with high probability. Then the s-t min cut for any other terminal t is unique. Further assume s satisfies the property of u in Lemma 3.12, which holds with probability $\frac{1}{2}$.

We claim that there does not exist $\frac{3}{4}|T|$ terminals $t \in T \setminus \{s\}$ with the same $\widetilde{\lambda}(s,t)$ value. Assume for contradiction that there exists such a subset R_1. By Lemma 3.7, the unique s-t min cuts are the same for all $t \in R_1$. Then for each $t \in R_1$, the t-side of s-t min cut has more than $\frac{1}{2}|T|$ terminals. By property of Lemma 3.12, there exist $\frac{1}{4}|T|$ terminals $R_2 \subseteq T \setminus \{s\}$ such that for each $t \in R_2$, the t-side of s-t min cut has less than $\frac{1}{2}|T|$ terminals. R_1 and R_2 are disjoint, so $|T \setminus \{s\}| \geq |R_1| + |R_2| = \frac{3}{4}|T| + \frac{1}{4}|T| = |T|$, which is impossible.

List all other terminals $t \in T \setminus \{s\}$ in non-increasing order of $\widetilde{\lambda}(s,t)$. Sample t uniformly from the list. Further assume that t has index between $\frac{1}{16}|T|$ and $\frac{1}{8}|T|$ in the list, which holds with probability $\frac{1}{16}$. Then $|T_1| = |\{t \in T : \widetilde{\lambda}(s,t) \geq \phi\}| \geq \frac{1}{16}|T|$. Also $|T_1| = |\{t : \widetilde{\lambda}(s,t) > \phi\}| + |\{t : \widetilde{\lambda}(s,t) = \phi\}| \leq \frac{1}{8}|T| + \frac{3}{4}|T| = \frac{7}{8}|T|$ by the claim. So $|T_2| = |T| - |T_1| \geq \frac{1}{16}|T|$. The probability that all three assumptions hold is at least $\frac{1}{32}(1 - n^{-D})$ for a large enough constant D. ∎

LEMMA 3.14. *Algorithm 1 runs in $\tilde{O}(F(m,n))$ time with high probability.*

Proof. The recursion depth is $O(\log n)$ because each recursive call reduces the size of T by a constant factor, and the recursion terminates when $|T| \leq 16$. Note that each recursive call only introduces one contracted node to a subproblem, so in each subproblem, there are $O(\log n)$ contracted nodes.

Let (m', n') be the size of a subproblem. Let m_i and n_i be the total edge and vertex size of the depth-i recursion layer respectively. Let k_i be the number of subproblems in the depth-i recursion layer. We first prove that $m_i = O(m \log^2 n)$, $n_i = O(n \log n)$ for every layer, and then bound the running time of a subproblem to be $\tilde{O}(F(m', n'))$. It follows that the total running time is $\tilde{O}(F(m,n))$.

To prove the bounds on m_i and n_i, we begin by bounding $k_i \leq n$. We first claim that each instance has at least 3 vertices. It holds for recursive case $|T| > 16$. For base case $|T| \leq 16$, consider its parent instance with terminal set T'. (Assuming the original graph is sufficiently large, so the root is not in base case.) $|T'| > 16$, so $|T'_1|, |T'_2| \geq \frac{1}{16}|T'| > 1$, and $|T'_1|, |T'_2| \geq 2$ as integers. The recursive calls have terminal sets $T'_1 \cup \{t_2\}$ or $T'_2 \cup \{t_1\}$, which contain at least 3 vertices.

Let $\alpha_i = n_i - 2k_i$. Initially $\alpha_0 = n - 2$. Consider the difference of α_{i+1} from α_i. A base case contributes to α_i (positively by the claim) but not to α_{i+1}. A recursive call creates two new contracted nodes, so n_{i+1} increases by 2, but k_{i+1} also increases by 1. In conclusion $\alpha_{i+1} \leq \alpha_i$. Therefore for every depth i, $\alpha_i = n_i - 2k_i \leq n$. $n_i \geq 3k_i$ by the claim, so $k_i \leq n$.

The uncontracted nodes are disjointly partitioned in a recursion layer. For contracted nodes, each of the $O(n)$ subproblems in a layer contains $O(\log n)$ contracted nodes, so the total number is $O(n \log n)$. Therefore, $n_i = O(n \log n)$.

Edges between uncontracted nodes are disjointedly partitioned. Edges between contracted and uncontracted nodes are repeated at most twice in a recursion layer because the uncontracted endpoint only occurs in one subproblem. For edges between contracted nodes, there are $O(\log^2 n)$ in each subproblem and $O(n \log^2 n)$ in a layer. In conclusion, $m_i = O(m + n \log^2 n)$.

Next we bound the running time of a recursive call. In the base case, $|T| \leq 16$. The algorithm runs $2^{|T|-1} = O(1)$ max flows to calculate the min cuts. In the checking step of Line 3, the algorithm calculates the cut values of $O(1)$ cuts, which can be done by enumerating all edges. In conclusion, the running time is $\tilde{O}(F(m', n'))$.

In the recursive case, the bottleneck of running time is calculating cut threshold, and calculating earliest T_1-T_2 min cut. By Lemma 3.11, each cut threshold call costs $\tilde{O}(F(m', n'))$. The condition of Line 9 is satisfied with constant probability by Lemma 3.13, so with high probability the **repeat** loop terminates in $O(\log n)$ rounds for all $O(n \log n)$ recursive calls. In conclusion, the running time is $\tilde{O}(F(m', n'))$. ∎

3.2.3 Phase 3: Post-processing In Phase 2, we get a laminar forest $\widetilde{L}$ containing all supreme sets of $\widetilde{G}$. This forest differs from our goal in two ways. First, $\widetilde{L}$ may contain non-supreme sets of $\widetilde{G}$, because the contraction in divide and conquer may generate new extreme sets; second, the supreme sets of $\widetilde{G}$ may differ from the supreme sets of G, because perturbation may distort the relation of equal-valued cuts and generate new extreme sets. In post-processing, we apply several post-order traversal on each tree in the forest to remove redundant sets, and only keep the weaker forest structure of terminals and cut values.

With some abuse of notation, we refer to a tree node and its corresponding set interchangeably. When we remove a node R during the algorithm, we assign its children to its parent if R is not a root, and we designate its children to be new roots of the forest if R is a root. In this way the new forest represents the laminar family after removing set R.

The algorithm works as follows. Compute the cut values under w as well as $\widetilde{w}$ of each set in $\widetilde{L}$. Record the number of terminals inside each set in $\widetilde{L}$. Then run three post-order traversals:

1. In the first post-order traversal, when visiting a node of set R, compare its cut value under $\widetilde{w}$ to all its children. If a child W has $\widetilde{d}(W) \leq \widetilde{d}(R)$, remove R.

2. In the second post-order traversal, when visiting a node of set R, compare its number of terminals to its parent W (if exists). If the numbers are equal, i.e. all terminals in W are contained in R, then remove R.

3. In the third post-order traversal, when visiting a node of set R, compare its cut value under w to all its children. If a child W has $d(W) \leq d(R)$, remove R.

The algorithm only removes sets from the forest, so all queried cut values are recorded. It is not hard to combine the three steps into one post-order traversal; we keep them separated for clarity.

After the second post-order traversal, we get a laminar family of all supreme sets under $\widetilde{w}$ (Lemma 3.15). This is very close to supreme sets under w, but not identical. We show in Lemma 3.16 that after the third post-order traversal, the laminar forest is the same as the family of all supreme sets of G in terms of terminal sets and cut values. We discard the information about non-terminals, and only record the projection on terminals and the cut value of each set. The output is a laminar family of terminal sets, and the cut value $c(R) = d(\mu(R))$ of each terminal set R.

FACT 3.6. *If W is an ancestor of U on the forest, and $\widetilde{d}(W) \geq \widetilde{d}(U)$, then W will be removed in the first post-order traversal.*

Proof. Let the tree path from U to root be $(W_0 = U, W_1, W_2, \ldots)$ after the first post-order traversal. We claim that $\widetilde{d}(U) > \widetilde{d}(W_1) > \widetilde{d}(W_2) > \ldots$. Assume otherwise; then for some k, $\widetilde{d}(W_k) \leq \widetilde{d}(W_{k+1})$. When visiting W_{k+1}, W_k is a child, so W_{k+1} should be removed according to the description of the first post-order traversal. This contradicts the assumption of tree path.

The claim implies that any ancestor W with $\widetilde{d}(W) \geq \widetilde{d}(U)$ is removed. ∎

FACT 3.7. *If W is an ancestor of U on the forest after the second post-order traversal, and $d(W) \geq d(U)$, then W will be removed in the third post-order traversal.*

Proof. Identical to Fact 3.6. ∎

LEMMA 3.15. *After the second post-order traversal, the laminar forest represents the family of all supreme sets of $\widetilde{G}$.*

Proof. We first prove that after the second post-order traversal, all supreme sets are in the forest. By Lemma 3.10, this holds before the post-processing, so only need to show that any supreme set $X = \mu(R)$ cannot be removed. Because X is extreme (Fact 3.3) and all sets in the forest are Steiner cuts (Lemma 3.10), no descendant of X can be a violator of X, so X is not removed in the first post-order traversal. By Lemma 3.5, X is a R-$(T \setminus R)$ min cut. If $\rho(W) = R$, $\widetilde{d}(X) \leq \widetilde{d}(W)$. This implies X's parent W cannot have $\rho(W) = R$, otherwise W would be removed in the first post-order traversal. So X cannot be removed in the second post-order traversal.

It remains to prove that after the second post-order traversal, all sets in the laminar forest are supreme sets. We prove by bottom-up induction that after visiting a node U, all nodes in the subtree of U are supreme sets under $\widetilde{w}$. Consider any node U. By induction, before visiting U, all nodes except U in the subtree are supreme sets. We only need to prove that if U is not removed, it is supreme. Let $R = \rho(U)$.

We first prove that U is extreme. Suppose not; then there exists an extreme violator Z. Let $Z' = \mu(\rho(Z))$, which is defined because Z is extreme. As a supreme set, Z' is in the forest. If Z' is in the subtree of U, then Z' is a subset of U, and $\widetilde{d}(Z') \leq \widetilde{d}(Z) \leq \widetilde{d}(U)$, so U would be removed in the first post-order traversal by Fact 3.6. If Z' is outside the subtree of U, then Z' is an ancestor of U because $\rho(Z' \cap U) \neq \emptyset$, which means Z' is a superset of U and $\rho(Z') \supseteq R$. We also have $\rho(Z') = \rho(Z) \subseteq R$, so $\rho(Z') = R$. This means all sets on the path from U to Z' have projection R, and U will be removed in the second post-order traversal. In conclusion, both cases contradict the assumption that U is not removed.

Now that U is extreme, let $W = \mu(R)$. As a supreme set, W is in the forest. Because $W \supseteq U$, W is an ancestor of U. Then all sets on the path from U to W have projection R. The only case when U is not removed in the second post-order traversal is $U = W$.

In conclusion, after the second post-order traversal, the laminar forest is exactly the family of all supreme sets in $\widetilde{G}$. ∎

LEMMA 3.16. *Let $\mathcal{R}$ be the family of supreme sets in G projected onto terminals. After the third post-order traversal, the laminar forest is $\{R \in \mathcal{R} : \widetilde{\mu}(R)\}$. Moreover, $d(\mu(R)) = d(\widetilde{\mu}(R))$ for all $R \in \mathcal{R}$. Note that the family of supreme sets in G is $\{R \in \mathcal{R} : \mu(R)\}$.*

Proof. First we prove that $d(\widetilde{\mu}(R)) = d(\mu(R))$ if $\mu(R)$ is defined. By Fact 3.5, perturbation cannot destroy extreme sets, so $\mu(R)$ is also extreme under $\widetilde{w}$, and $\mu(R) \subseteq \widetilde{\mu}(R)$. If $\widetilde{\mu}(R)$ is extreme before perturbation, $\widetilde{\mu}(R) \subseteq \mu(R)$, so $\widetilde{\mu}(R) = \mu(R)$ and $d(\widetilde{\mu}(R)) = d(\mu(R))$. The remaining case is that $\widetilde{\mu}(R)$ is not extreme before perturbation, so there exists an extreme violator Z of $\widetilde{\mu}(R)$ by Fact 3.2. Because $Z \subseteq \widetilde{\mu}(R)$, $\rho(Z) \subseteq R$, and $Z \subseteq \mu(\rho(Z)) \subseteq \mu(R)$ by laminarity of supreme sets (Lemma 3.4). So

$$d(\mu(R)) \leq d(Z) \leq d(\widetilde{\mu}(R))$$

(The first inequality uses Fact 3.1 on $\mu(R)$, and the second follows that Z is a violator.) Because $\widetilde{\mu}(R)$ is extreme after perturbation,

$$\widetilde{d}(\mu(R)) > \widetilde{d}(\widetilde{\mu}(R)).$$

By Fact 3.4, perturbation does not change order of unequal cut values, so the two inequalities imply $d(\mu(R)) = d(\widetilde{\mu}(R))$.

Next we prove that if $X = \widetilde{\mu}(R)$ is defined but $\mu(R)$ is not defined, X will be removed during the third post-order traversal. Because $\mu(R)$ is not defined, there is no extreme set with terminal set R, and X is not extreme under w. So there exists an extreme violator Z such that $Z \subsetneqq X$ and $d(Z) \leq d(X)$. Because X is extreme under $\widetilde{w}$, $\widetilde{d}(Z) > \widetilde{d}(X)$. This is impossible if $d(Z) < d(X)$ because perturbation does not change order of unequal cut values (Fact 3.4), so $d(Z) = d(X)$. Because $\mu(R)$ is not defined, $\rho(Z) \subsetneqq R$. Because Z is extreme, $\mu(\rho(Z))$ is defined and $Y = \widetilde{\mu}(\rho(Z))$ is in the forest after the second post-order traversal. Because $X \setminus Y \neq \emptyset$, and X, Y are in the laminar family, $Y \subsetneqq X$. We have $d(Y) \leq d(Z) = d(X)$, so X will be removed in the third post-order traversal by Fact 3.7.

Third we prove that for all $R \in \mathcal{R}$, $\widetilde{\mu}(R)$ is in the family after the third post-order traversal. $\mu(R)$ is defined because $R \in \mathcal{R}$. By Fact 3.5, $\mu(R)$ is extreme under $\widetilde{w}$, and $X = \widetilde{\mu}(R)$ is defined. By Lemma 3.15, X is in the laminar forest before the third post-order traversal. It remains to show that X is not removed in the third post-order traversal. Assume otherwise; then there exists a child $W \subsetneqq X$ such that $d(W) \leq d(X)$. Because X is extreme under $\widetilde{w}$, $\widetilde{d}(W) > \widetilde{d}(X)$, so $d(W) = d(X)$. By Lemma 3.15, W is another supreme set under $\widetilde{w}$. Let $W = \widetilde{\mu}(P)$, then $P \subsetneqq R$. We proved $d(W) = d(\mu(P)), d(X) = d(\mu(R))$, so $d(\mu(P)) = d(\mu(R))$. Because the supreme sets under w are laminar, $\mu(P) \subsetneqq \mu(R)$, so $\mu(P)$ violates the extreme property of $\mu(X)$. This contradicts the fact that $\mu(X)$ is extreme under w. ∎

LEMMA 3.17. *The post-processing takes $\tilde{O}(m)$ time.*

Proof. Given the cut values, the post-order traversal can be executed in $O(n)$ time. Next show that cut values can be computed in $\tilde{O}(m)$ time. We describe the process under w, and $\widetilde{w}$ is the same.

We use a dynamic tree data structure on the laminar forest. For the sake of analysis, add a dummy root representing V and let all trees in the forest be its children. In this laminar tree, every vertex is contained in a path from some node to the root. Attach each vertex u to the lowest node $l(u)$ of the path. For each edge u, v, add $w(u, v)$ to the path from the node $l(u)$ to $l(v)$, excluding the lowest common ancestor (LCA) of $l(u)$ and $l(v)$. This costs $\tilde{O}(m)$ time.

To see the correctness of cut values, we prove that for any set X in the tree and any edge (u, v), (u, v) is in cut X if and only if X is on the path from $l(u)$ to $l(v)$ excluding the LCA. The edge (u, v) is in cut X iff $u \in X$, $v \notin X$ or $v \in X$, $u \notin X$, which is equivalent to that exactly one of $l(u)$ and $l(v)$ is in the subtree of X. This is further equivalent to X being on the path from $l(u)$ to $l(v)$ excluding the LCA. ∎

Algorithm 2: External augmentation.

Input : A laminar forest L of supreme sets on terminals, a function $c(R) = d(\mu(R))$ for every node $u \in L$.

foreach $u \in L$ *in post-order traversal* **do**
  Calculate $\text{rdem}(R) = \max\{\tau - c(R), \sum_{W \in ch(R)} \text{rdem}(W)\}$.
  Add an edge of weight $\text{rdem}(R) - \sum_{W \in ch(R)} \text{rdem}(W)$ from x to any terminal in R.
Output the new edges.

4 Algorithm for Steiner Connectivity Augmentation

In this section, we give the algorithm for Steiner connectivity augmentation. We will assume throughout that the target connectivity $\tau \geq 2$. If not, an optimal solution is a spanning tree on the components containing terminals. This algorithm has three parts which we present in different subsections below:

(a) An algorithm for external augmentation. (Section 4.1)

(b) An algorithm for adding augmentation chains to achieve Steiner connectivity of $\tau - 1$. (Section 4.3)

(c) An algorithm to augment the Steiner connectivity from $\tau - 1$ to τ using random matchings. (Section 4.4)

4.1 External Augmentation This section considers the external augmentation problem. That is, given an undirected graph $G = (V, E)$ with integer edge weights $w \geq 0$, terminal set $T \subseteq V$ and a connectivity target τ, insert a new node x, and add edges connecting x and vertices in V with minimum total weight (the weights of new edges are chosen by the algorithm), such that the Steiner connectivity of T is increased to τ.

Given the laminar forest L of supreme sets on terminals, external augmentation can be solved by a simple greedy algorithm in Algorithm 2. For a rooted forest containing a node R, define $ch(R)$ as the set of children of R. Define the recursive demand $\text{rdem}(R)$ of terminal set R to be the minimum total weight of edges across the cut $\mu(R)$ to satisfy all supreme sets in the subtree of R in external augmentation. To satisfy $\mu(R)$, the demand is $\tau - c(R)$; to satisfy other supreme sets in the subtree, take the sum of recursive demands of R's children. Therefore $\text{rdem}(R) = \max\{\tau - c(R), \sum_{P \in ch(R)} \text{rdem}(P)\}$ and can be computed by tree DP. The algorithm performs a post-order traversal on each tree of L. When visiting each node R, add edges from x to any terminal in R such that the new edges crossing R reach total weight of $\text{rdem}(R)$. That is, add a new edge of weight $\text{rdem}(R) - \sum_{P \in ch(R)} \text{rdem}(P)$.

LEMMA 4.1. *For any node $R \in L$, at the time Algorithm 2 finishes visiting R, the algorithm adds edges of total weight* $\text{rdem}(R)$ *across R.*

Proof. We prove by induction on decreasing depth of R in the rooted forest. As a base case, when R is a leaf, the sum on children is empty, so the algorithm adds an edge of weight $\text{rdem}(R)$ across R. As an inductive step, consider an internal node R. When visiting R, every child $P \in ch(R)$ does not get new edges after visiting it because the algorithm only add edges to terminals in the visited node during post-order traversal. Therefore the new edges crossing each child P has a total weight of $\text{rdem}(P)$ when visiting R. We add edges of weight $\text{rdem}(R) - \sum_{P \in ch(R)} \text{rdem}(P) \geq 0$ across R, so the total weight of new edges across R is $\text{rdem}(R)$ after visiting R. ∎

LEMMA 4.2. *Algorithm 2 outputs a feasible solution to the external augmentation problem.*

Proof. Consider any Steiner cut X. If X is not extreme, by Fact 3.2 there exists an extreme violating set $Z \subsetneq X$ with $d(Z) \leq d(X)$. If Z is augmented to τ, so is X because any new edge across Z also crosses X. Therefore we only need to show that every extreme set X is augmented to value τ.

Let $R = \rho(X)$. $R \in L$. By Lemma 4.1, when visiting R, we add $\text{rdem}(R)$ edges across R. The algorithm only adds edges, so the final cut value of X is at least $d(X) + \text{rdem}(R) \geq d(\mu(R)) + \tau - c(R) = \tau$. ∎

LEMMA 4.3. *Algorithm 2 outputs an optimal solution of external augmentation. The optimal value is $\sum_{R \in L_1} \text{rdem}(R)$, where L_1 is the set of all roots in L.*

Proof. We claim that for each $R \in L$, we need to add at least $\text{rdem}(R)$ edges across $\mu(R)$. First, $\mu(R)$ requires $\tau - c(R)$ edges because $c(R) = d(\mu(R))$. Second, by induction, for each child $W \in ch(R)$, $\mu(W)$ requires $\text{rdem}(W)$ edges, so $\mu(R)$ requires $\sum_{W \in ch(R)} \text{rdem}(W)$ edges as a super set of all $\mu(W)$'s.

The overall lower bound on the total number of new edges is $\sum_{R \in L_1} \text{rdem}(R)$. Algorithm 2 adds $\text{rdem}(R)$ edges across R when visiting R by Lemma 4.1, and does not add any new edge across R after visiting R if $R \in L_1$. Therefore the output of Algorithm 2 has size $\sum_{R \in L_1} \text{rdem}(R)$. As a feasible solution by Lemma 4.2, the output is an optimal solution. ∎

LEMMA 4.4. *Algorithm 2 takes $O(n)$ time.*

Proof. The size of forest L is $O(n)$. Calculating $\text{rdem}(R)$ takes $|ch(R)|$ time, so Line 2 takes $O(n)$ time in total. When visiting each node, we only add edge to one terminal, so inserting edges takes $O(n)$ time in total. ∎

4.2 Splitting off View The external augmentation problem is closely related to Steiner connectivity augmentation problem. In our algorithm for Steiner connectivity augmentation problem, assume the input is graph G, terminals T and target τ. Construct the corresponding external augmentation problem with the same graph G, terminals T and target τ. After finding the structure of supreme sets in Section 3, Algorithm 2 produces the optimal external augmentation solution F^{ext}.

Assume the total weight of F^{ext} is k. For the sake of analysis, regard the weighted edges as parallel unweighted edges. If k is odd, add an arbitrary external edge to make the degree of x even. By Mader's splitting off theorem 4.1, we can split x off to get $\lceil \frac{k}{2} \rceil$ edges F on V which preserves the Steiner connectivity to be τ. (There is no cut edge incident on x because the Steiner connectivity is at least 2 after external augmentation and the degree of x is always even during the splitting off.) This means F is a feasible solution to the Steiner connectivity augmentation problem. On the other hand, $\lceil \frac{k}{2} \rceil$ is a lower bound on the value of any feasible solution by Lemma 4.5. Therefore, F is an optimal solution to Steiner connectivity augmentation problem. This means we can get an optimal solution to augmentation problem by splitting off the external augmentation output.

In general, splitting off is not easier than augmentation. However, our external augmentation solution only consists of edges connecting x and T, which makes splitting off easier. Going through this reduction also provides useful information. In our augmentation problem, we now know how an optimal solution looks like, in terms of the degree of every vertex in the new edges. We call it degree constraint of vertices, and say a terminal u has vacant degree if the number of new edges incident to u has not reached the degree constraint. Our algorithm will only add edges to terminals with vacant degree.

THEOREM 4.1. (MADER'S THEOREM [40]) *If a vertex x is not incident to a cut edge and $d(x) \neq 3$, then there exist a pair of edges incident to x such that splitting them preserves the s-t minimum cut value for each pair $s, t \in V \setminus \{x\}$.*

LEMMA 4.5. *If the optimal external augmentation solution has value k, then any feasible solution to the corresponding Steiner augmentation problem has value at least $\lceil \frac{k}{2} \rceil$.*

Proof. Let F be any feasible solution to Steiner augmentation problem. Let p be the total weight of F. Construct F' by replacing every edge $(u, v) \in F$ by two edges (u, x) and (v, x) with the same weight. Then F' is a feasible solution to external augmentation with total weight $2p$. Because k is the optimal value of external augmentation, $k \leq 2p$, which implies $p \geq \frac{k}{2}$. Because we work on integer weights, p and k are integers, and $p \geq \lceil \frac{k}{2} \rceil$. ∎

4.3 Greedily Adding Augmentation Chains In this section, we split off the external augmentation output.

For a terminal set R of some extreme set, define its demand $\text{dem}(R) = \tau - d(\mu(R))$, and the cut value will change as we add edges. List the maximal supreme sets $(X_1, \ldots, X_r)$ with demands at least 2, such that X_1 and X_r have the minimum cut value. Let their terminal sets be $(R_1, \ldots, R_r)$. Because a Steiner min cut of minimum size is an extreme set with no extreme superset, X_1 and X_r are Steiner min cuts. Define an augmentation chain to be a set of edges $F = \{(a_i, b_{i+1})\}_{1 \leq i \leq r-1}$ such that $a_i, b_i \in R_i$, and every vertex's degree does not exceed its degree in the external augmentation solution.

A basic algorithm is to repetitively add augmentation chains until all demands of extreme sets are reduced to at most 1. We solve the case of demand 1 in Section 4.4. During the algorithm, the extreme sets structure

may change and need to be maintained. Our key observation Theorem 4.3 guarantees that the algorithm does not create new extreme sets, so we only need to handle the original supreme sets in L. To maintain the value $c(R) = d(\mu(R))$ for every $R \in L$, notice that the effect of adding an edge (a, b) in augmentation chain is increasing the cut value of all supreme sets on the L tree path from a to b excluding the LCA, so we can maintain $c(R)$ by a dynamic tree data structure. For a node $R \in L$, if $c(R)$ is increased to make $c(R) \geq c(P)$ for some child P of R, then R is no longer the terminal set of any extreme set and need to be removed from L.

The running time of the basic algorithm is $O(n\tau)$, that is $O(n)$ for each augmentation chain. To speed it up, we lazily use the same edges to form the chain until they become invalid. We use a priority queue to maintain the time when each edge will become invalid, which is further maintained by efficient data structures. Then we can add the same chain while repeatedly replacing the first invalid edge. This speedup is identical to [11] and discussed in Appendix C.

THEOREM 4.2. *Given the supreme sets forest and the cut values of the supreme sets, we can add all augmentation chains so that all supreme sets have demand at most 1 in time $\tilde{O}(n)$.*

4.3.1 Property of Augmentation Chains We now prove the key property of augmentation chains, stated below.

THEOREM 4.3. *Adding an augmentation chain does not create new extreme set.*

Proof. Assume for contradiction that U is a new extreme set after adding $F = \{(a_i, b_{i+1})\}_{1 \leq i \leq r-1}$, and W is an extreme violating set of U before augmentation.

Let $d(\cdot)$ and $d'(\cdot)$ be the cut values before and after augmentation respectively. Then $d(W) \leq d(U)$ but $d'(W) > d'(U)$. So $d_F(W) > d_F(U)$. Because we add at most 2 edges to any extreme set, $2 \geq d_F(W) > d_F(U) \geq 0$. So $d_F(U)$ can only be 0 or 1. $d_F(W) > d_F(U)$ also implies that there must be a new edge crossing W but not crossing U. Such an edge $e_0 = (a_i, b_{i+1})$ must have $a_i, b_{i+1} \in U$. Also, no extreme set can contain all terminals in U because e_0 crosses two maximal supreme sets.

Starting from $e_0 = (a_i, b_{i+1})$, we have b_i and a_{i+1} also belong to U by Lemma 4.7. Because $d_F(U) \leq 1$, one of a_{i-1} and b_{i+1} is contained in U. By repeating this step, we can always walk along F in one of the two directions, until reaching a_1 or b_r. By Lemma 4.6, U cannot cross X_1 or X_r in terminal because they only have one new edge. Therefore $\rho(X_1) \subsetneqq U$ or $\rho(X_r) \subsetneqq U$. Assume wlog $\rho(X_1) \subsetneqq U$.

$X_1 \cap U$ is a proper subset of U that contains terminals, so $d'(X_1 \cap U) > d'(U)$. Then

$$d(X_1 \cap U) \geq d'(X_1 \cap U) - 1 \geq d'(U) = d(U) + d_F(U) \geq d(U)$$

By (2.3) of Fact 2.1, $d(X_1 \cup U) \leq d(X_1)$. $X_1 \cup U$ has the same terminal set as U, so it is a Steiner cut and $d(X_1 \cup U) \geq d(X_1)$. ($X_1$ is a Steiner min cut.) Therefore $d(X_1 \cup U) = d(X_1)$ and $X_1 \cup U$ is also a Steiner min cut. Now the chain of inequality becomes equality, so $d_F(U) = 0$.

When $d_F(U) = 0$, we can continue the walking on both directions of F, so U contains all edges of the chain F. By Lemma 4.8, because $X_1 \cup U$ is a Steiner min cut, it cannot cross any extreme set. Therefore U must contain all terminals which is in an extreme set with demand at least 2. However, $X_1 \cup U$ is a Steiner min cut, so there is terminal in $\overline{X_1 \cup U}$ which has demand $\mathrm{dem}(X_1) \geq 2$, a contradiction. ∎

LEMMA 4.6. *Assume U is a new extreme set after adding F. If U crosses an extreme set X in terminal, then $d_F(X) = 2$, and the two new edges across X are both contained in U.*

Proof. Because U crosses X in terminal, $U \cap X, X \setminus U$ and $U \setminus X$ are Steiner cuts. Because X is extreme, $d(X \setminus U) > d(X)$. By (2.4) of Fact 2.1, $d(U \setminus X) < d(U)$. Because U is a new extreme set, $d'(U \setminus X) > d'(U)$. Therefore

$$d(U \setminus X) + d_F(U \setminus X) = d'(U \setminus X) \geq d'(U) + 1 = d(U) + d_F(U) + 1 \geq d(U \setminus X) + d_F(U) + 2$$

$$d_F(U \setminus X) \leq d_F(U) + d_F(U \setminus X, U \cap X) \leq d_F(U) + d_F(X) = d_F(U) + 2$$

Comparing these two inequalities, we have $d_F(U \setminus X) = d_F(U) + 2$, and all inequalities are equality. Therefore $d_F(X) = 2$, and the two new edges incident to X must both connect $U \setminus X$ and $U \cap X$. ∎

LEMMA 4.7. *Assume U is a new extreme set after adding F. If X_i has two new edges incident on a_i and b_i, and $a_i \in U$, then $b_i \in U$ as well.*

Proof. Assume for contradiction that $b_i \notin U$. Then U crosses X_i. (a_i, b_i are terminals in $U \cap X_i$ and $X_i \setminus U$. $U \setminus X_i$ is also a Steiner cut because no extreme set can contain all terminals in U.) By Lemma 4.6, $b_i \in U$, a contradiction. ∎

LEMMA 4.8. *A Steiner min cut cannot cross any extreme set.*

Proof. Assume for contradiction that a Steiner min cut Y crosses an extreme set X.

If $\rho(X \setminus Y) \neq \emptyset$ and $\rho(Y \setminus X) \neq \emptyset$, we have $d(X \setminus Y) > d(X)$ by extreme property of X, and $d(Y \setminus X) \geq d(Y)$ because Y is a Steiner min cut. Adding them contradicts (2.2) of Fact 2.1.

The remaining case is that $X \setminus Y$ or $Y \setminus X$ contains no terminal. Because X and Y are Steiner cuts, $\rho(X \cap Y) \neq \emptyset$. $d(X \cap Y) > d(X)$ by extreme property of X. By submodularity $d(X \cup Y) < d(Y)$, so $X \cup Y$ is not a Steiner cut and $\rho(X \cup Y) = T$. Because X and Y are Steiner cuts, they cannot contain all the terminals. So $\rho(X \setminus Y) \neq \emptyset, \rho(Y \setminus X) \neq \emptyset$, which contradicts the assumption. ∎

4.4 Augmenting Connectivity by One via Random Matchings After adding as many augmentation chains as possible, there is no extreme set with demand at least 2. If all demands are at most 0, the solution is complete. The remaining case is that there are extreme sets with demand 1, or equivalently cut value $\tau - 1$. This means the Steiner minimum cut is now $\tau - 1$. This section solves the subproblem that we augment Steiner connectivity by 1, from $\tau - 1$ to τ. The new edges need to augment or cross every Steiner mincut.

The optimal external augmentation is adding a star connecting x and every demand-1 extreme sets. (These extreme sets must be maximal and disjoint.) Contract these extreme sets, and let K be the set of contracted extreme sets. In the splitting off view, the optimal solution is a perfect matching of K. (If $k = |K|$ is odd, we match an arbitrary node twice.)

4.4.1 Cactus Structure In global connectivity setting, previous algorithms construct the matching using the cactus structure of global minimum cuts. A cactus is a graph where any two simple cycles intersect in at most one node. A min cut cactus is a weighted cactus where the nodes correspond to a partition of the vertices of the original graph, and every min cut of the original graph is represented by a min cut with the same vertex partition in the cactus. The demand-1 extreme sets are minimal min cuts, so they are represented by all degree-2 nodes of the cactus. Given the min cut cactus, one can construct in $O(n)$ time a matching on all leaves to augment every min cut [44].

In contrast, Steiner min cuts do not form such a neat structure. In the special case of $T = \{s, t\}$, the Steiner min cuts are just s-t min cuts, which can only be represented by a directed acyclic graph (DAG). Fortunately, when projected onto terminals, the Steiner min cuts also admit a cactus structure. This is known as the 'skeleton' part of connectivity carcass structure [17].

Calculating the cactus structure of Steiner min cuts on terminals can be time-consuming. We circumvent the construction, and only exploit its existence to bound the success probability of our random matching algorithm. The key lemma is as follows.

LEMMA 4.9. *There exists a cyclic order of K such that any Steiner min cut partitions the order into two intervals.*

Proof. Take any Euler tour[3] of a cactus structure of Steiner min cuts on terminals. Take the subsequence of all degree-2 nodes. Any Steiner min cut has value 2 so it cuts the Euler tour in two edges, breaking it into two intervals. Restricted to the degree-2 nodes, it also partitions the cyclic order into two intervals. ∎

4.4.2 Algorithm Description and Correctness As presented in Algorithm 3, the algorithm runs in phases, each phase adding a partial matching. A matching of K is feasible if it augments every Steiner min cut. Call a partial matching feasible if it is contained in a feasible matching of K. Because we always end a phase with

[3] An Euler tour is a (not necessarily simple) cycle that contains every edge of the graph exactly once. Here, the graph is the underlying cactus of the cactus structure.

Algorithm 3: Augment Steiner connectivity by 1.

Input : Graph G, terminal set T, demand-1 extreme sets family K as a partition of T.

1 Contract every set in K to be a node. (When adding edge to a node, add to an arbitrary terminal in the node.)
2 **if** $|K|$ *is odd* **then**
3 Pick arbitrary $u, v \in K$, add edge $\{u, v\}$, and remove u from K.
4 **while** $|K| \geq 4$ **do**
5 **repeat**
6 Randomly match $\lfloor \frac{|K|}{4} \rfloor$ pairs of nodes in K. Let M be the partial matching.
7 **until** M *is a feasible partial solution*;
8 Remove endpoints in M from K.
9 Match the last two nodes of K.

a feasible partial solution, there always exists a feasible matching containing the current partial solution. This invariant holds when $|K|$ is reduced to 2, in which case matching the last two nodes is the only solution and must be a feasible solution.

The ignored detail about checking whether M is a feasible partial solution in Line 7 is explained in Lemma 4.10.

LEMMA 4.10. *Form a new graph G' by adding a node x and adding a star connecting x and $K \setminus V(M)$. M is a feasible partial solution if and only if the Steiner min cut of G' is τ.*

Proof. If the Steiner min cut of G' is τ, by Mader's splitting off theorem 4.1 we can split off x to get a feasible matching of K.

If the Steiner min cut is not τ, it must be $\tau - 1$. (The Steiner min cut of G' is at least $\tau - 1$ because the Steiner min cut of G is $\tau - 1$. The Steiner min cut of G' is at most τ because the degree cut of any $u \in K$ is τ.) Let $(S, \overline{S})$ be a Steiner cut of value $\tau - 1$. Without loss of generality, assume $x \in S$. Because S has cut value at least $\tau - 1$ in G, there can be no star edge across the cut. Therefore all unmatched nodes are in S, and the cut S will not be augmented for any matching containing M. ∎

4.4.3 Running Time The algorithm runs in phases, each phase reduce $|K|$ from $2t$ to $2t - 2\lfloor \frac{2t}{4} \rfloor \leq t + 1$ ($|K|$ is always even), so the algorithm runs $O(\log n)$ phases. Each phase makes several trials until getting a feasible partial solution. By Lemma 4.11, the success probability is at least $\frac{1}{3}$, so each phase ends in $O(\log n)$ trials whp. In each trial, sampling the random matching takes $O(n)$ time. Checking the feasibility calls a Steiner min cut according to Lemma 4.10, which runs in poly-logarithmic many max flows [37]. In conclusion, the total running time is $\tilde{O}(F(m, n))$.

LEMMA 4.11. *The success probability that M is a feasible partial matching is at least $\frac{1}{3}$.*

Proof. Generate M by picking one uniformly random unmatched pair of nodes at each step. In each step, match the picked pair, and remove them from the cyclic order given by Lemma 4.9. Let $|K| = k$ initially, then $k - 2i$ unmatched nodes remain after the i-th step.

Let A_i be the event that in the i-th step, the picked pair (u, v) is not adjacent in the cyclic order. $\Pr[A_i] = \frac{k-2i-1}{k-2i+1}$. Because each step is independent,

$$\Pr[A_1 \wedge A_2 \wedge \ldots \wedge A_{\lfloor k/4 \rfloor}] = \frac{k-3}{k-1} \times \frac{k-5}{k-3} \times \ldots \times \frac{k - 2\lfloor k/4 \rfloor - 1}{k - 2\lfloor k/4 \rfloor + 1} = \frac{k - 2\lfloor k/4 \rfloor - 1}{k-1} \geq \frac{1}{3}$$

when $k \geq 4$.

Finally, we prove that when no step matches adjacent nodes, the partial solution is feasible. Assume for contradiction that all events A_i happen, but the output partial matching is not feasible. Then by Lemma 4.10, there exists a Steiner min cut $(S, \overline{S})$ such that all leaves in its one side S are matched. Consider the last matched leaf u in S. Assume u is matched with v. Because cut $(S, \overline{S})$ is not augmented, $v \in S$. S forms an interval in the cyclic order by Lemma 4.9. u, v are the last two matched nodes in S, so they are adjacent in cyclic order when matched, which contradicts the assumption. ∎

5 Algorithm for Steiner Splitting-Off

In this section, we generalize our Steiner augmentation algorithm to solve the Steiner splitting-off problem. We assume there is no cut edge incident to x and the degree of x is even, which a standard assumption for splitting-off theorems including Lovász's theorem and Mader's theorem.

The splitting-off problem reduces to a degree-constrained external augmentation problem by removing the external vertex x and setting the degree constraint $\beta(u)$ to be the weight of removed (u, x) edge for each vertex u. The connectivity target τ of the degree-constrained problem is the Steiner connectivity in the original graph.

The algorithm follows the same framework as Section 4. However, degree constraints introduce a significant new complication: that we now need to add edges incident to nonterminals. (Recall that in the Steiner connectivity augmentation algorithm, all edges added to the graph were incident to terminals only.) As before, we have three parts:

(a) For external augmentation, we use maximum flow calls on suitably defined auxiliary graphs for each heavy path in a heavy-light decomposition of the trees of the supreme sets forest. The external edges are tied to the heavy paths, and this allows us to track the contribution of edges connecting nonterminals. (Section 5.1)

(b) Next, we adapt the definition of augmentation chains to incorporate degree constraints in a way that retains its key properties and allows us to design a greedy algorithm. (Section 5.2)

(c) Finally, for augmenting Steiner connectivity from $\tau - 1$ to τ, we incorporate the notion of *surrogates* in our random matching algorithm. (Section 5.3)

5.1 Degree-Constrained External Augmentation In degree-constrained augmentation, we have a degree constraint $\beta(u)$ on each vertex u, and the aim is to augment Steiner connectivity to τ while maintaining that the degree of u in the new edges is at most $\beta(u)$. Although our unconstrained augmentation algorithm only adds edges to terminals, in degree-constrained setting we may have to add to non-terminals as well because of the terminals' degree constraints. This makes the problem much harder because now a new edge does not augment all extreme sets in a supreme set, and there may be exponentially many crossing extreme sets in a supreme set.

In degree-constrained setting, we also define external augmentation, that is inserting a new vertex x and adding edges from V to x, such that each vertex $u \in V$ has at most $\beta(u)$ new edges, and the Steiner connectivity of T is augmented to at least τ. A degree-constrained external augmentation instance is feasible if there exists such an edge set. Equivalently, the instance is feasible if adding a (u, x) edge of capacity $\beta(u)$ for each vertex u augments the Steiner connectivity to at least τ. Because our degree constraints are generated from the splitting-off problem, we can assume the problem is feasible. The rest of the section constructs an optimal solution with minimum total weight.

Recall that in Section 4.1, for external augmentation without degree constraint, the optimal total weight is $\sum_{R \in L_1} \text{rdem}(R)$, where the sets in L_1 are the terminal projection of maximal supreme sets. This is a lower bound on the value of degree-constrained external augmentation solution, because any feasible degree-constrained solution is also feasible for the unconstrained problem. Surprisingly, Lemma 5.3 shows that the optimal degree-constrained solution matches this lower bound regardless of the constraints, as long as the problem is feasible.

We restate the related notations here. A node $R \in L$ represents the terminal set of a supreme set. $\mu(R)$ is the supreme set of terminal set R. The recursive demand of $R \in L$ is $\text{rdem}(R) = \max\{\tau - d(\mu(R)), \sum_{W \in ch(R)} \text{rdem}(W)\}$.

If some $R \in L$ has $d(\mu(R)) \geq \tau$, then all extreme sets with terminal set R are already satisfied by extreme property of $\mu(R)$, and R can be contracted to its parent in L. After this step, we always have $\text{rdem}(R) > 0$ for $R \in L$. Call a node R *critical* if

$$\text{rdem}(R) = \tau - d(\mu(R)) > \sum_{W \in ch(R)} \text{rdem}(W).$$

In particular, leaves are critical because $\text{rdem}(R) > 0$.

Next we describe the external augmentation algorithm. Apply heavy-light decomposition separately on each tree of the forest L. This partitions the forest into a vertex-disjoint collection of paths (called heavy paths). Define the depth of a heavy path P to be the number of heavy paths intersecting the path from P to the root of the tree containing P. A key property of heavy-light decomposition is that the maximum depth of heavy paths

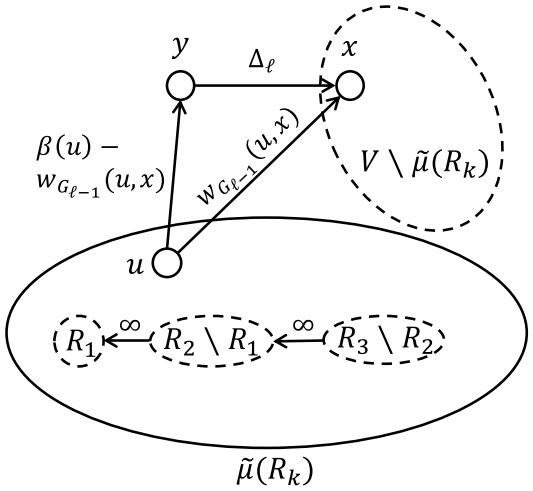


Figure 2: Construction of the flow network H_ℓ. Dashed ovals are contracted sets.

is $O(\log n)$. Process the heavy paths in a bottom-up manner by indexing the paths in non-increasing order of depth, and treating the paths one by one in that order.

Let G_ℓ be the external augmentation graph after processing the first ℓ heavy paths. Initially $G_0 = (V \cup \{x\}, E)$ is the original graph after adding an external vertex x. In the ℓ-th iteration, the algorithm adds external edges into $G_{\ell-1}$ to form a graph G_ℓ, and the aim is to satisfy all extreme sets with terminal sets on the ℓ-th heavy path P_ℓ. To determine these new external edges, we run a max flow on an auxiliary graph H_ℓ. The algorithm only adds edges, so all extreme sets will be satisfied after processing all heavy paths.

We construct a directed graph H_ℓ based on $G_{\ell-1}$ and run R_1-x max flow on H_ℓ, where R_1 is the leaf of the ℓ-th heavy path P_ℓ. In the construction, all extreme sets with terminal sets on P_ℓ are R_1-x cuts, so their cut values on H_ℓ are at least the R_1-x max flow value, which turns out to be τ by Lemma 5.1. We construct G_ℓ from H_ℓ such that these extreme sets have the same cut values in G_ℓ as their flow values in H_ℓ for the R_1-x max flow, so these extreme sets are augmented to at least τ. Let $P_\ell = (R_1^{(\ell)}, R_2^{(\ell)}, \ldots, R_{k_\ell}^{(\ell)})$, $R_1^{(\ell)} \subsetneqq R_2^{(\ell)} \subsetneqq \ldots \subsetneqq R_{k_\ell}^{(\ell)}$. For simplicity, we will drop the superscripts of $R_i^{(\ell)}$ in the rest of the section, as long as the path is clear. The following are the detailed steps for constructing H_ℓ from $G_{\ell-1}$, illustrated in Figure 2.

1. Replace every undirected edge (u, v) by two directed edges (u, v) and (v, u).
2. Add an external vertex y.
3. Add a directed edge from u to y with capacity $\beta(u) - w_{G_{\ell-1}}(u, x)$ for every $u \in V$, where $w_{G_{\ell-1}}$ is the edge capacity in $G_{\ell-1}$.
4. For each $1 \le i \le k$, contract $R_i \setminus R_{i-1}$. Also contract $V \cup \{x\} \setminus \tilde{\mu}(R_k)$. ($\tilde{\mu}(\cdot)$ is the supreme sets in $\tilde{G}$, the graph after perturbation.) For the sake of analysis, we explain contraction as adding a cycle of infinite capacity connecting the contracted set, so that the vertices set of H_ℓ is $V \cup \{x, y\}$.
5. Add a directed edge of infinite capacity from s_{i+1} to s_i for each $1 \le i \le k-1$, where s_i is an arbitrary vertex in the contracted $R_i \setminus R_{i-1}$.
6. Add a directed edge from y to x of capacity $\Delta_\ell = \mathrm{rdem}(R_k) - \sum_{W \in B(R_k)} \mathrm{rdem}(W)$, where $B(R_k)$ is the set of roots of heavy paths in the subtree of R_k except R_k itself.

Run an R_1-x max flow on H_ℓ to get an integer max flow f_ℓ. Form G_ℓ by adding the flow values on (u, y) edges to the capacity of (u, x) edges in $G_{\ell-1}$ for each $u \in V$. That is, $w_{G_\ell}(u, x) = w_{G_{\ell-1}}(u, x) + f_\ell(u, y)$ for each $u \in V$.

After processing all p heavy paths, the resulting graph G_p consists of the original graph G and external edges connecting V and x. Return these external edges as the output. We prove in Lemma 5.3 that this output is feasible and optimal. The proof is inductive and based on analysis of the flow f_ℓ.

LEMMA 5.1. *Assume the external augmentation is feasible. Then in each H_ℓ, the $R_1^{(\ell)}$-x min cut value is τ.*

We defer the proof of Lemma 5.1 and use it to show the optimality of the external augmentation algorithm. Use $\delta_{H_\ell}(\cdot)$ to denote the outgoing cut value in H_ℓ, emphasizing that H_ℓ is directed.

LEMMA 5.2. *If $S \subseteq \tilde{\mu}(R_{k_\ell}^{(\ell)})$ and $\rho(S) = R_i^{(\ell)}$ is a node in path P_ℓ, then*

- $\delta_{H_\ell}(S) = d(S) + \beta(S)$, *where* $\beta(S) = \sum_{u \in S} \beta(u)$.
- $\delta_{H_\ell}(S \cup \{y\}) = d_{G_{\ell-1}}(S) + \Delta_\ell$.

Proof. Because $S \subseteq \tilde{\mu}(R_{k_\ell})$ and $\rho(S) = R_i$, S does not cross the contracted $R_j \setminus R_{j-1}$ and $V \cup \{x\} \setminus \tilde{\mu}(R_k)$, and the infinite capacity edges in step 5 do not contribute to $\delta_{H_\ell}(S)$ or $\delta_{H_\ell}(S \cup \{y\})$.

In $\delta_{H_\ell}(S)$, the edges on V contribute $d(S)$. For each $u \in V$, (u, x) and (u, y) contribute $w_{G_{\ell-1}}(u, x) + \beta(u) - w_{G_{\ell-1}}(u, x) = \beta(u)$ by step 3. So $\delta_{H_\ell}(S) = d(S) + \beta(S)$.

In $\delta_{H_\ell}(S \cup \{y\})$, the edges on V and (u, x) edges contribute $d_{G_{\ell-1}}(S)$ in total. The (y, t) edge contributes Δ_ℓ. The (u, y) edges do not cross the cut. So $\delta_{H_\ell}(S \cup \{y\}) = d_{G_{\ell-1}}(S) + \Delta_\ell$. ∎

LEMMA 5.3. *The degree-constraint external augmentation algorithm outputs a feasible solution of optimal value $\sum_{R \in L_1} \mathrm{rdem}(R)$.*

Proof. First we prove that every Steiner cut is augmented to at least τ. We only need to prove this for extreme sets. For other Steiner cuts U, consider their extreme violators W in Fact 3.2. If W is satisfied, we add $\tau - d(W) \geq \tau - d(U)$ edges connecting W and x, which also cross U because $W \subseteq U$. So satisfying all extreme sets implies satisfying all Steiner cuts.

For any extreme set X with terminal set R, suppose R is processed in the ℓ-th heavy path $P_\ell = (R_1, R_2, \ldots, R_{k_\ell})$ and $R = R_i$. We have $R_1 \subseteq R \subseteq X$. By definition of supreme sets, $X \subseteq \mu(R) \subseteq \mu(R_k)$. By Fact 3.5, $\mu(R_k) \subseteq \tilde{\mu}(R_k)$. So $R_1 \subseteq X \subseteq \tilde{\mu}(R_k)$ and X is an R_1-x cut that is not contributed by infinite capacity edges in H_ℓ. Let λ_ℓ be the flow value of f_ℓ. Then the flow value on cut X is

$$\lambda_\ell = \sum_{u \in X, v \in V \cup \{x,y\} \setminus X} f_\ell(u, v) - f_\ell(v, u).$$

In this flow value, the flow in edges on V and (u, x) edges is upper bounded by their total capacity $d_{G_{\ell-1}}(X)$. Combined with the flow in (u, y) edges,

$$\lambda_\ell \leq \sum_{u \in X, v \in V \cup \{x\} \setminus X} f_\ell(u, v) + \sum_{u \in X} f_\ell(u, y) \leq d_{G_{\ell-1}}(X) + \sum_{u \in X} f_\ell(u, y).$$

Because we define $w_{G_\ell}(u, x) = w_{G_{\ell-1}}(u, x) + f_\ell(u, y)$, we have $d_{G_\ell}(X) = d_{G_{\ell-1}}(X) + \sum_{u \in X} f_\ell(u, y) \geq \lambda_\ell$. Since $d_{G_\ell}(X) \geq \lambda_\ell = \tau$ by Lemma 5.1, we conclude that X is satisfied.

Next we prove that the output satisfies degree constraints, so it is a feasible solution. Fix any vertex $u \in V$. For each ℓ, $w_{G_\ell}(u, x) = w_{G_{\ell-1}}(u, x) + f_\ell(u, y) \leq w_{G_{\ell-1}}(u, x) + \beta(u) - w_{G_{\ell-1}}(u, x) = \beta(u)$. The output uses $w_{G_p}(u, x)$ degrees after processing the last heavy path p. By induction, $w_{G_p}(u, x) \leq \beta(u)$, so the edge weight does not exceed $\beta(u)$.

By Lemma 4.3, the output value is lower bounded by $\sum_{R \in L_1} \mathrm{rdem}(R)$, which is the optimal value if we remove the degree constraint. Because the (y, x) edge in H_ℓ has capacity Δ_ℓ and we only add new edge for flows through y, each heavy path adds at most $\Delta_\ell = \mathrm{rdem}(R_k) - \sum_{W \in B(R_k)} \mathrm{rdem}(W)$. So the total value of output is upper bounded by the sum $\sum_{R \in L_1} \mathrm{rdem}(R)$. ($L_1$ nodes have rdem added once, and other nodes have rdem added once and subtracted once.) In conclusion the output value is $\sum_{R \in L_1} \mathrm{rdem}(R)$ and is optimal. ∎

Instead of proving Lemma 5.1 by itself, we prove it in combination with the following statement.

LEMMA 5.4. *For each ℓ, let $R_m^{(\ell)}$ be the highest critical node on the ℓ-th heavy path. Then $\mu(R_m^{(\ell)}) \cup \{y\}$ is an R_1-x min cut in H_ℓ.*

We give corollaries of the statements of Lemma 5.1 and Lemma 5.4 to facilitate the inductive proof.

LEMMA 5.5. *If the statement of Lemma 5.4 holds for ℓ, then G_ℓ is constructed from $G_{\ell-1}$ by adding Δ_ℓ edges from $\mu(R_m^{(\ell)})$ to x.*

Proof. By the statement, $\mu(R_m) \cup \{y\}$ is an R_1-x min cut in H_ℓ. Also, $f_\ell(y, x) = \Delta_\ell$ because (y, x) is a min cut edge. By flow conservation, $f_\ell(y, x) = \sum_{u \in V} f_\ell(u, y)$. The algorithm adds the integer flow on (u, y) edges as new edges to G_ℓ.

In an R_1-x max flow, no flow goes from sink side to source side of an R_1-x min cut. Because y is in the source side, all flow through y are from the source side, so all new edges are from $\mu(R_m)$. ∎

LEMMA 5.6. *If the statement of Lemma 5.1 and Lemma 5.4 hold for ℓ, then for any $R \subsetneqq T$ such that $R \supseteq R_k^{(\ell)}$, the R-$(T \setminus R)$ min cut value increases by Δ_ℓ from $G_{\ell-1}$ to G_ℓ.*

Proof. Let the R-$(T \setminus R)$ min cut values in $G_{\ell-1}$ and G_ℓ be $\lambda_{\ell-1}$ and λ_ℓ respectively. Let U be the R-side of the latest R-$(T \setminus R)$ min cut in G_ℓ. By Lemma 5.5, the algorithm adds Δ_ℓ edges to $\mu(R_m)$ in G_ℓ. We claim that $U \supseteq \mu(R_m)$, which implies $\lambda_\ell = d_{G_\ell}(U) = d_{G_{\ell-1}}(U) + \Delta_\ell \geq \lambda_{\ell-1} + \Delta_\ell$. The increment is exactly Δ_ℓ because iteration ℓ only adds Δ_ℓ units of edges.

To prove the claim, assume for contradiction that $\mu(R_m) \setminus U \neq \emptyset$. Because U is the latest R-$(T \setminus R)$ min cut and $R_m \subseteq R_k \subseteq R$, $d_{G_\ell}(U \cup \mu(R_m)) > d_{G_\ell}(U)$. By submodularity,

$$d_{G_\ell}(U \cap \mu(R_m)) < d_{G_\ell}(\mu(R_m)) \tag{5.23}$$

By the statement of Lemma 5.4 for ℓ, $\mu(R_m) \cup \{y\}$ is an R_1-x min cut in H_ℓ, so the statement of Lemma 5.1 for ℓ gives

$$\tau = \delta_{H_\ell}(\mu(R_m) \cup \{y\}) = d_{G_{\ell-1}}(\mu(R_m)) + \Delta_\ell = d_{G_\ell}(\mu(R_m)) \tag{5.24}$$

(The second step uses Lemma 5.2, and the third step uses Lemma 5.5.)

Because $R_m \subseteq U \cap \mu(R_m) \subseteq \mu(R_m)$, $U \cap \mu(R_m)$ is an R_1-x cut that is not contributed by infinite capacity edges in H_ℓ. We have $d_{G_\ell}(U \cap \mu(R_m)) = d_{G_{\ell-1}}(U \cap \mu(R_m)) + \sum_{u \in U \cap \mu(R_m)} f_\ell(u, y)$, which is at least the flow value of f_ℓ on cut $U \cap \mu(R_m)$. The R_1-x max flow value on H_ℓ is τ by the statement of Lemma 5.1 for ℓ, so $d_{G_\ell}(U \cap \mu(R_m)) \geq \tau$. But (5.23) and (5.24) gives $d_{G_\ell}(U \cap \mu(R_m)) < \tau$, a contradiction. ∎

Before proving Lemma 5.1, we first introduce two helper statements below.

FACT 5.1. *In any path P_ℓ, for each $i > 1$,*

$$\mathrm{rdem}(R_i) - \sum_{W \in B(R_i)} \mathrm{rdem}(W) \geq \mathrm{rdem}(R_{i-1}) - \sum_{W \in B(R_{i-1})} \mathrm{rdem}(W)$$

Moreover, if R_i is not critical, then the inequality can be replaced by equality.

Proof. By definition of rdem,

$$\mathrm{rdem}(R_i) \geq \sum_{W \in ch(R_i)} \mathrm{rdem}(W) = \mathrm{rdem}(R_{i-1}) + \sum_{W \in ch(R_i) \setminus \{R_{i-1}\}} \mathrm{rdem}(W).$$

When R_i is not critical, the inequality can be replaced by equality. Note that $ch(R_i) \setminus \{R_{i-1}\} = B(R_i) \setminus B(R_{i-1})$, so the statement holds. ∎

COROLLARY 5.1. *Let R_m be the highest critical node in a path P_ℓ. Then*

$$\mathrm{rdem}(R_{k_\ell}) - \sum_{W \in B(R_{k_\ell})} \mathrm{rdem}(W) = \mathrm{rdem}(R_m) - \sum_{W \in B(R_m)} \mathrm{rdem}(W).$$

Now we are prepared to prove Lemma 5.1 in combination with Lemma 5.4.

Proof. [Proof of Lemma 5.1 and Lemma 5.4] Because every heavy path stops at a leaf, R_1 is critical. So the highest critical node R_m exists.

We prove by induction on ℓ. Fix the path P_ℓ, and assume the statement holds for all paths P_j with $j < \ell$.

Let S be the R_1-side of the latest R_1-x min cut in H_ℓ. Because of the infinite capacity edges, S contains a prefix of $(R_1, R_2 \setminus R_1, \ldots, R_{k_\ell} \setminus R_{k_\ell - 1})$, and $S \subseteq \tilde{\mu}(R_{k_\ell})$. That is, $\rho(S) = R_i$ for some node R_i in P_ℓ.

First we prove that $\delta_{H_\ell}(S) \geq \tau$. If $y \notin S$, $\delta_{H_\ell}(S) = d(S) + \beta(S)$ by Lemma 5.2, which is at least τ because the problem is feasible. The remaining case is $y \in S$. Let $S' = S \setminus \{y\}$. Now $\delta_{H_\ell}(S) = d_{G_{\ell-1}}(S') + \Delta_\ell$ by Lemma 5.2.

S' is an R_i-$(T \setminus R_i)$ cut. Initially, the R_i-$(T \setminus R_i)$ min cut value is $d(\mu(R_i))$ by Lemma 3.5. For every other heavy path P_j in the subtree of R_i, iteration j adds Δ_j to the min cut value by Lemma 5.6. Before iteration ℓ, all other paths in the subtree of R_i are already processed, adding a total of $\sum_{W \in B(R_i)} \operatorname{rdem}(W)$ to the min cut value. So $d_{G_{\ell-1}}(S') \geq d(\mu(R_i)) + \sum_{W \in B(R_i)} \operatorname{rdem}(W)$. Also, by applying Fact 5.1 iteratively from k_ℓ to i, $\Delta_\ell = \operatorname{rdem}(R_{k_\ell}) - \sum_{W \in B(R_{k_\ell})} \operatorname{rdem}(W) \geq \operatorname{rdem}(R_i) - \sum_{W \in B(R_i)} \operatorname{rdem}(W)$. Therefore,

$$\begin{aligned}
\delta_{H_\ell}(S) &= d_{G_{\ell-1}}(S') + \Delta_\ell \\
&\geq d(\mu(R_i)) + \sum_{W \in B(R_i)} \operatorname{rdem}(W) + \operatorname{rdem}(R_i) - \sum_{W \in B(R_i)} \operatorname{rdem}(W) \\
&\geq d(\mu(R_i)) + \tau - d(\mu(R_i)) = \tau
\end{aligned}$$

Next we prove that $\delta_{H_\ell}(\mu(R_m) \cup \{y\}) = \tau$, so it is an R_1-x min cut.

$$\begin{aligned}
\delta_{H_\ell}(\mu(R_m) \cup \{y\}) &= d_{G_{\ell-1}}(\mu(R_m)) + \Delta_\ell \\
&= d_{G_{\ell-1}}(\mu(R_m)) + \operatorname{rdem}(R_m) - \sum_{W \in B(R_m)} \operatorname{rdem}(W) \\
&= d(\mu(R_m)) + \sum_{W \in B(R_m)} \operatorname{rdem}(W) + \operatorname{rdem}(R_m) - \sum_{W \in B(R_m)} \operatorname{rdem}(W) \\
&= d(\mu(R_m)) + \tau - d(\mu(R_m)) = \tau
\end{aligned}$$

The second equality uses Corollary 5.1. The third equality is because each path P_j in the subtree of R_m adds Δ_j new edges (in $G_{\ell-1}$) to a supreme set in path P_j by Lemma 5.5, the supreme sets are contained in $\mu(R_m)$, and the total value is $\sum_{W \in B(R_m)} \operatorname{rdem}(W)$. The fourth equality is because R_m is critical. ∎

LEMMA 5.7. *The running time of degree-constrained external augmentation algorithm is $\tilde{O}(F(m,n))$ for all heavy paths.*

Proof. If two heavy paths P_1, P_2 have the same depth, then they are disjoint, and all supreme sets of the nodes on P_1 are disjoint from all supreme sets on P_2. By Lemma 3.15, we computed the supreme sets in the perturbed graph $\tilde{G}$. The supreme sets in $\tilde{G}$ form a laminar family, so the heavy paths P_ℓ of the same depth have disjoint $\tilde{\mu}(R_{k_\ell}^{(\ell)})$. We contract $V \setminus \tilde{\mu}(R_{k_\ell}^{(\ell)})$ when constructing H_ℓ, so all max flow calls for heavy paths of the same depth can run in parallel with total graph size $O(m)$. The maximum depth in heavy-light decomposition is $O(\log n)$, so the total running time is $O(F(m,n) \log n)$. ∎

5.2 Degree-Constrained Augmentation Given an external augmentation solution, we split it off to get new edges on V such that the Steiner connectivity is preserved to be at least τ. They form a degree-constrained augmentation solution. Equivalently, we can first remove all external edges, use the degrees in the external edges to be degree constraints, and then add new edges under these degree constraints. Note that these degree constraints generated by external augmentation solution are tighter then the original degree constraints. With some abuse of notation, we forget about the original degree constraints and refer to these tight degree constraints as $\beta(\cdot)$.

We slightly modify the definition of augmentation chains in degree-constrained setting. List all maximal supreme sets $Q = (X_1, \ldots, X_r)$ with demands at least 2, such that X_1 and X_r have the minimum cut value. (Supreme sets are defined in the graph before adding chain.) Define an augmentation chain as a set of edges $F = \{(a_i, b_{i+1})\}_{1 \leq i \leq r-1}, a_i, b_i \in X_i$, such that adding these edges satisfies the degree constraint. In contrast to the unconstrained setting, now we allow the endpoints to be non-terminals.

5.2.1 Property of Augmentation Chains We prove in Lemma 5.10 that adding an augmentation chain does not create new extreme sets, even if we allow nonterminal endpoints. This is a stronger version of Theorem 4.3. It follows that splitting off by augmentation chains preserves the Steiner min cut value to be at least τ.

FACT 5.2. *If Q is nonempty, then $r \geq 2$, and X_1 and X_r are Steiner min cuts.*

Proof. Let (S_1, S_2) be the two sides of a Steiner min cut and λ be the Steiner min cut value. If S_1 is extreme, let $U_1 = S_1$, otherwise let U_1 be the extreme violator of S_1 in Fact 3.2. Then U_1 is extreme and $d(U_1) \leq d(S_1) = \lambda$. Also $d(U_1) \geq \lambda$ because U_1 is a Steiner cut. So $d(U_1) = \lambda$. If an extreme set $U' \supsetneqq U_1$, then $d(U') < d(U_1)$, which is impossible. So U_1 is a maximal extreme set. Applying the same argument to S_2 gives another maximal extreme set U_2 the is a Steiner min cut.

There exists at least two maximal supreme sets that are Steiner min cuts. When Q is nonempty, they have minimum cut value $\lambda \leq \tau - 2$ among supreme sets in Q. So we choose X_1 and X_r to be Steiner min cuts. ∎

LEMMA 5.8. *Suppose U is a new extreme set after adding an augmentation chain F and U crosses some $X_i \in Q$. If $d_F(X_i) - d_F(X_i \setminus U) \leq 1$, then $\rho(U \cap X_i) \neq \emptyset$.*

Proof. Assume for contradiction that $\rho(U \cap X_i) = \emptyset$, then $\rho(U \setminus X_i) = \rho(U) \neq \emptyset, \rho(X_i \setminus U) = \rho(X_i) \neq \emptyset$. Also $U \cap X_i \neq \emptyset$ because U and X_i cross, and the two difference sets are proper subsets of U and X_i respectively. Then because U is extreme after adding F, and X_i is extreme,

$$d(U \setminus X_i) + d_F(U \setminus X_i) > d(U) + d_F(U) \tag{5.25}$$

$$d(X_i \setminus U) > d(X_i) \tag{5.26}$$

By posi-modularity (2.2),

$$d(U \setminus X_i) + d(X_i \setminus U) \leq d(X_i) + d(U)$$

$$d_F(U \setminus X_i) + d_F(X_i \setminus U) \leq d_F(X_i) + d_F(U) \tag{5.27}$$

(5.27) implies $d_F(U \setminus X_i) - d_F(U) \leq d_F(X_i) - d_F(X_i \setminus U) \leq 1$. So (5.25) implies

$$d(U \setminus X_i) \geq d(U) \tag{5.28}$$

Together, (5.26) and (5.28) violate posi-modularity (2.2). ∎

LEMMA 5.9. *If U is a new extreme set created by an augmentation chain F, then $U \cap X_1 = U \cap X_r = \emptyset$.*

Proof. X_1 is a Steiner min cut with one new edge endpoint a_1. Let $\lambda = d(X_1)$ be the Steiner min cut value. Assume for contradiction that $U \cap X_1 \neq \emptyset$. $U \neq X_1$ because U is not extreme before augmentation.

First we rule out $U \supsetneqq X_1$. If that happens, because U is extreme after augmentation,

$$d(X_1) + d_F(X_1) > d(U) + d_F(U)$$

$$\lambda + 1 = d(X_1) + d_F(X_1) \geq d(U) + d_F(U) + 1 \geq \lambda + 1$$

Therefore U is a Steiner min cut and $d_F(U) = 0$. A Steiner min cut cannot cross any supreme set by Lemma 4.8. U also does not cross any edge in F because $d_F(U) = 0$, so U contains all supreme sets in Q. But U is a Steiner cut, so there is a terminal outside U. U's demand is at least 2 because U is a Steiner min cut and $F \neq \emptyset$, so $V \setminus U$ has demand at least 2, and there exists a supreme set outside U with demand at least 2, which contradicts the definition of Q.

Next we rule out the case that U and X_1 cross. $d_F(X_1) - d_F(X_1 \setminus U) \leq d_F(X_1) = 1$, so $U \cap X_1$ is a Steiner cut by Lemma 5.8. Because U is extreme after augmentation,

$$d(X_1 \cap U) + d_F(X_1 \cap U) > d(U) + d_F(U).$$

Notice that $d_F(X_1 \cap U) \le d_F(X_1) = 1$. So $d(X_1 \cap U) \ge d(U) + d_F(U)$. By submodularity (2.1), $d(X_1 \cup U) \le d(X_1) - d_F(U)$. If $X_1 \cup U$ is a Steiner cut, then it is a Steiner min cut and $d_F(U) = 0$, so we can use the same argument as the former case. If $X_1 \cup U$ is not a Steiner cut, it must be $\rho(X_1 \cup U) = T$. Because X_1 and U are Steiner cuts, $\rho(X_1 \setminus U) \neq \emptyset, \rho(U \setminus X_1) \neq \emptyset$. Because X_1 is extreme, $d(X_1 \setminus U) > d(X_1)$. Because U is extreme after adding F, $d(U \setminus X_1) + d_F(U \setminus X_1) > d(U) + d_F(U)$. Along with $d_F(U \setminus X_1) \le d_F(U) + d_F(X_1) = d_F(U) + 1$, we obtain $d(U \setminus X_1) \ge d(U)$. Adding this inequality with $d(X_1 \setminus U) > d(X_1)$ contradicts posi-modularity (2.2).

Because U is not contained in any extreme set, the only remaining case is that $U \cap X_1 = \emptyset$. $U \cap X_r = \emptyset$ follows the same argument. ∎

LEMMA 5.10. *In degree constraint setting, adding an augmentation chain F does not create any new extreme set.*

Proof. In this proof, the cut value $d(\cdot)$ and supreme sets are defined in the graph before adding the augmentation chain F. Let $d_F(\cdot)$ be the cut value on the edges of F. Then after adding F, the cut value of U is $d(U) + d_F(U)$.

Assume for contradiction that U is a new extreme set. Because U is not extreme before augmentation, let $W \subsetneqq U$ be its extreme violator in Fact 3.2 with $d(W) \le d(U)$. Because U is extreme after adding F, $d(W) + d_F(W) > d(U) + d_F(U)$. Noticing that we add at most 2 edges to any extreme set, the above inequalities imply

$$0 \le d_F(U) < d_F(W) \le 2.$$

$d_F(W) > d_F(U)$ implies that there exists a new edge crossing W but not crossing U. Suppose $(a_i, b_{i+1}) \in F$ is such an edge, then $a_i, b_{i+1} \in U$. So U intersects two maximal supreme sets X_i, X_{i+1}, and U is not contained in any extreme set.

Let Q_U be the sub-family of Q that intersects $\rho(U)$. Because $d_F(W) > 0$, W is contained in a maximal supreme set X_w in Q. Because $X_w \cap U \supseteq W$ is a Steiner cut, $X_w \in Q_U$, and Q_U is nonempty. Let $X_t = \arg\min_{Y \in Q_U} d(Y)$ be the set with minimum cut value in Q_U. Because $X_t \in Q_U$, $U \cap X_t$ is a Steiner cut. Because U is extreme after augmentation and not contained in X_t,

$$d(X_t \cap U) + d_F(X_t \cap U) > d(U) + d_F(U)$$

Using submodularity $d(X_t \cap U) + d(X_t \cup U) \le d(U) + d(X_t)$,

$$d(U \cup X_t) < d(X_t) + d_F(X_t \cap U) - d_F(U) \tag{5.29}$$

By Lemma 5.9, $1 < t < r$. Divide F into two halves $F_1 = \{(a_i, b_{i+1})\}_{1 \le i \le t-1}$ and $F_2 = \{(a_i, b_{i+1})\}_{t \le i \le r-1}$. Then the extra term $d_F(X_t \cap U) - d_F(U)$ is

$$d_F(X_t \cap U) - d_F(U) = 1[b_t \in U] + 1[a_t \in U] - d_{F_1}(U) - d_{F_2}(U)$$

where $1[\cdot]$ is the indicator function that takes 1 if the event happens and 0 otherwise. Let $\theta_1 = 1[b_t \in U] - d_{F_1}(U)$, $\theta_2 = 1[a_t \in U] - d_{F_2}(U)$, so that $\theta_1, \theta_2 \le 1$. Let $U_1 = U \cup X_t$. (5.29) can be rewritten as

$$d(U_1) < d(X_t) + \theta_1 + \theta_2.$$

When $\theta_1 \le 0$, let $U_2 = U_1$. When $\theta_1 = 1$, we find a partner X_p below, and let $U_2 = U_1 \cup X_p$. Because $\theta_1 = 1$, $b_t \in U$ and $d_{F_1}(U) = 0$. Then for each $1 \le i \le t-1$, either $a_i, b_{i+1} \in U$ or $a_i, b_{i+1} \notin U$. The two ends are $a_1 \notin U$ by Lemma 5.9 and $b_t \in U$, so there exists some $2 \le p \le t-1$ such that $b_p \notin U, a_p \in U$. Then $d_F(X_p \setminus U) = 1, d_F(X_p) = 2$, so $U \cap X_p$ is a Steiner cut by Lemma 5.8 and $X_p \in Q_U$. Because X_p is extreme,

$$d(U \cap X_p) > d(X_p).$$

Applying submodularity on U_1 and X_p ($U_1 \cap X_p = U \cap X_p$ because X_t is disjoint from X_p),

$$d(U_1 \cup X_p) < d(U_1)$$

In both cases ($\theta_1 \le 0$ and $\theta_1 = 1$), we have

$$d(U_2) \le d(U_1) - \theta_1 < d(X_t) + \theta_2.$$

The other direction is symmetric, except we start with U_2, not U_1. When $\theta_2 \leq 0$, let $U_3 = U_2$. When $\theta_2 = 1$, we can find a partner X_q such that $t \leq q \leq r-1$ and $b_q \in U$ but $a_q \notin U$. Let $U_3 = U_2 \cup X_q$. Using the same argument, we have

$$d(U_3) \leq d(U_2) - \theta_2 < d(X_t) \tag{5.30}$$

U_3 is the union of U, X_t and X_p, X_q if they exist. Recall that X_t is the set with minimum cut value in Q_U, and $X_p, X_q \in Q_U$ if they exist. U_3 is a Steiner cut because X_1 is outside by Lemma 5.9. U_3 is not extreme because it contains U, which is not contained in any extreme set. So there exists an extreme violator W' of U_3 with $d(W') \leq d(U_3)$ by Fact 3.2. Because W' is extreme, it is contained in a maximal supreme set $X_{w'}$. Because $X_{w'} \cap U_3 \supseteq W'$ is a Steiner cut, either $X_{w'} \cap U$ is a Steiner cut, or $X_{w'}$ is one of X_t, X_p, X_q if they exist. So $X_{w'} \in Q_U$, and $d(X_{w'}) \geq d(X_t)$. Because $W' \subseteq X_{w'}$ and $X_{w'}$ is extreme, $d(W') \geq d(X_{w'})$. In conclusion

$$d(U_3) \geq d(W') \geq d(X_{w'}) \geq d(X_t)$$

which contradicts (5.30). ∎

The algorithm is to simply keep adding augmentation chains. By Lemma 5.11, the partial solution is always feasible. Moreover, because we split off the optimal external augmentation solution, the partial solution is contained in an optimal solution. The algorithm stops when there is no supreme sets with demand at least 2, or equivalently, when the Steiner connectivity reaches at least $\tau - 1$. This situation is solved in Section 5.3.

Let G_ℓ be the graph after adding ℓ chains, and G_ℓ^{ext} be the external graph after splitting off ℓ chains. (The definition of G_ℓ differs from Section 5.1.) Initially $G_0 = G$ is the original graph, and $G_0^{ext} = (V \cup \{x\}, E \cup F^{ext})$ is the original graph after adding the external augmentation solution. Let $\beta_\ell(u)$ be the remaining external degrees of u in G_ℓ, or equivalently the capacity of (u, x) edge in G_ℓ^{ext}. Let $\beta_\ell(S) = \sum_{u \in S} \beta_\ell(u)$. Then for any vertex set S, $d_{G_\ell^{ext}}(S) = d_{G_\ell}(S) + \beta_\ell(S)$.

LEMMA 5.11. *The external graph G_ℓ^{ext} always has Steiner connectivity at least τ.*

Proof. We prove by induction on ℓ that the Steiner connectivity in G_ℓ^{ext} is at least τ. The base case $\ell = 0$ holds because the external augmentation solution is feasible by Lemma 5.3.

Assume the statement holds up to ℓ. Consider a minimal Steiner min cut X in $G_{\ell+1}^{ext}$, taking the side $x \notin X$. By inductive hypothesis, $d_{G_\ell^{ext}}(X) \geq \tau$. If no edges of chain $\ell + 1$ are inside X, then the splitting off does not decrease the cut value of X, and $d_{G_{\ell+1}^{ext}}(X) \geq d_{G_\ell^{ext}}(X) \geq \tau$. The remaining case is that there exists a chain edge inside X. A chain edge connects two maximal supreme sets (defined in G_ℓ), so X is not extreme in G_ℓ and hence not extreme in $G_{\ell+1}$ by Lemma 5.10, and there exists an extreme violator W of X in $G_{\ell+1}$, $d_{G_{\ell+1}}(W) \leq d_{G_{\ell+1}}(X)$. Because $W \subsetneq X$, $\beta_{\ell+1}(W) \leq \beta_{\ell+1}(X)$. So

$$d_{G_{\ell+1}^{ext}}(W) = d_{G_{\ell+1}}(W) + \beta_{\ell+1}(W) \leq d_{G_{\ell+1}}(X) + \beta_{\ell+1}(X) = d_{G_{\ell+1}^{ext}}(X)$$

which contradicts the assumption that X is a minimal Steiner min cut in $G_{\ell+1}^{ext}$. So this case is impossible, and the Steiner connectivity is at least τ. ∎

5.2.2 Finding Augmentation Chains Next we describe the detailed steps to find an augmentation chain efficiently. We first establish the algorithm, with a slight modification in the heavy-light decomposition of external augmentation, then prove it always finds augmentation chains in Lemma 5.19.

Let L_2 be the collection of $R \in L$ such that R is critical and all ancestors of R are not critical. Let L_{high} be the union of paths from nodes in L_2 to the roots, that is a sub-forest of L with leaves L_2. In the external augmentation algorithm, modify the heavy-light decomposition step such that the heavy paths stop at nodes in L_2. More precisely, apply heavy-light decomposition on L_{high}, then separately apply heavy-light decomposition on each subtree rooted at a node in L_2, and then merge the heavy-light decompositions. This way, a leaf to root path still passes $O(\log n)$ heavy paths, so all proofs in Section 5.1 work for the new definition. But now, each heavy path stops at a leaf or a node in L_2, which is critical. We say that the heavy-light decomposition on L_{high} is *upper level*, and the ones on the subtrees rooted at L_2 are *lower level*.

We maintain the forest L_{high}. For lower level heavy paths, regard them as contracted into their ancestors in L_2. For each $R \in L_{high}$, maintain the cut value $c(R)$, which is initialized to $d(\mu(R))$. We assign an external edge to a heavy path if the edge is added by the external augmentation algorithm when processing the heavy path. In each iteration, we list all roots of L_{high}, and pick the roots R_i with $c(R_i) \leq \tau - 2$ to form a list $Q_T = (R_1, \ldots, R_r)$, such that $c(R_1)$ and $c(R_r)$ are the minimum two in Q_T. Pick a chain $F = \{(a_i, b_{i+1})\}_{1 \leq i \leq r-1}$, such that a_i, b_i are vacant vertices (having unused external edges) assigned to heavy paths in the tree rooted at R_i. (When $a_i = b_i$, it needs two vacant degrees.) We use the external edges assigned to lower level paths first. The external edges assigned to upper level paths are used only when all lower level degrees in the tree are used.

After picking a chain, we add it to the solution, and update the vacant degrees. Next we update $c(R)$ values. For each endpoint u in the chain, define its *representative leaf* $U \in L_2$ as follows. If u is assigned to an upper level path, let U be the leaf of the heavy path to which u is assigned. If u is assigned to a lower level path, let U be the node to which that path is contracted. Add 1 to the values $c(R)$ for each R on the path from U to the root. Next we update the structure of L_{high}. If some node $R \in L_{high} \setminus L_2$ and its child W have $c(R) \geq c(W)$, delete R in L_{high} and assign its children to its parent (when R is a root, its children become new roots).

For efficiency, we use the same chain edges until they become invalid. This occurs when either some endpoint uses up its vacant degree, or adding the chain causes some forest node R to be deleted from Q_T. This second event occurs either when the supreme set corresponding to that forest node R has demand at most 1 or when one of its children W satisfies $c(R) \geq c(W)$. The $c(R)$ values are maintained by a dynamic tree data structure. We can use priority queues to maintain the expiration time of each endpoint and each forest node. This speedup is identical to [11] and discussed in Appendix C.

THEOREM 5.1. *Given the supreme sets forest and the cut values of the supreme sets, we can add all augmentation chains so that all supreme sets have demand at most 1 in time $\tilde{O}(n)$.*

The rest of this section establishes correctness by proving the chain found by the algorithm is an augmentation chain. One difficulty of the degree-constrained setting is that the chain edges may connect nonterminals, so for a fixed terminal set R, the supreme set whose projection is R may change as we add augmentation chains. This does not happen in the unconstrained setting where we only add edges to terminals.

In the proofs, $\mu(\cdot)$ is the supreme set defined in the original graph, and $\mu_\ell(\cdot)$ is the supreme set defined in G_ℓ. That is, for any $R \subseteq T$, if R is the terminal projection of an extreme set in G_ℓ, then $\mu_\ell(R)$ is defined to be the supreme set of terminals R in G_ℓ; otherwise, $\mu_\ell(R)$ is undefined. In particular, $\mu_0(\cdot) = \mu(\cdot)$ because $G_0 = G$. Recall that $\beta_\ell(u)$ is the remaining external degree after adding ℓ augmentation chain, so the cut value in G_ℓ^{ext} can be expressed as $d_{G_\ell^{ext}}(U) = d_{G_\ell}(U) + \beta_\ell(U)$, where $\beta_\ell(U) = \sum_{u \in U} \beta_\ell(u)$.

LEMMA 5.12. *For $R \in L_{high}$, the $c(R)$ value maintained by the algorithm is equal to $d_{G_\ell}(\mu(R))$ at every time ℓ.*

Proof. Consider the contribution of an edge (u, w) in the ℓ-th augmentation chain. Let U be the representative leaf of u. If u is assigned to a upper level heavy path, U is the leaf of the path, and by Lemma 5.5 $u \in \mu(U)$. If u is assigned to a lower level heavy path, U is the parent of the path, and we also have $u \in \mu(R_m) \subseteq \mu(U)$ for some node R_m in the path. Therefore u is in $\mu(R)$ for every R on the path from U to the root. Also these supreme sets do not contain w, because w is assigned to a different tree from u, and (u, w) connects two maximal supreme sets. In the algorithm, $c(R)$ is increased for R on the path from U to the root, which correctly maintains $d_{G_\ell}(\mu(R))$. ∎

FACT 5.3. *If $\mu_i(R)$ is defined, then for any $0 \leq j < i$, $\mu_j(R)$ exists and $\mu_i(R) \subseteq \mu_j(R)$.*

Proof. $\mu_j(R)$ is extreme in G_i, so it is also extreme in G_j by repeatedly applying Lemma 5.10. The statement then follows the definition of $\mu_j(R)$. ∎

LEMMA 5.13. *For any $R \in L_2$, $d_{G_\ell^{ext}}(\mu(R)) = \tau$ holds for every ℓ.*

Proof. By Lemma 5.3, the external augmentation solution is feasible, so for each $R \in L_2$, $\beta(\mu(R)) \geq \tau - d(\mu(R))$. Moreover, Lemma 5.3 says the total external degree is $\sum_{R \in L_1} \mathrm{rdem}(R)$, which equals $\sum_{R \in L_2} \tau - d(\mu(R))$ because all internal nodes in $L_{high} \setminus L_2$ are not critical. Therefore $\beta(\mu(R)) = \tau - d(\mu(R))$ for each $R \in L_2$.

Initially, $d_{G_0^{ext}}(\mu(R)) = \beta(\mu(R)) + d(\mu(R)) = \tau$. $G_{\ell+1}^{ext}$ is formed by splitting off edges for an augmentation chain from G_ℓ^{ext}, so $d_{G_{\ell+1}^{ext}}(\mu(R)) \leq d_{G_\ell^{ext}}(\mu(R))$, and $d_{G_\ell^{ext}}(\mu(R)) \leq \tau$ for each ℓ by induction. Also $d_{G_\ell^{ext}}(\mu(R)) \geq \tau$ by Lemma 5.11, so the statement holds. ∎

LEMMA 5.14. *For any $R \in L_2$, if $\mu(R)$ has vacant degree after adding the ℓ-th chain, then $\mu_\ell(R)$ is defined, and $d_{G_\ell}(\mu(R)) = d_{G_\ell}(\mu_\ell(R))$.*

Proof. Fix some $R \in L_2$. Assume first that $\mu_\ell(R)$ is defined. By Fact 5.3, $\mu_\ell(R) \subseteq \mu(R)$, so

$$\beta_\ell(\mu_\ell(R)) \leq \beta_\ell(\mu(R)) \tag{5.31}$$

By Lemma 5.13, $\tau = d_{G_\ell^{ext}}(\mu(R)) = \beta_\ell(\mu(R)) + d_{G_\ell}(\mu(R))$. Combined with (5.31), we have

$$\beta_\ell(\mu_\ell(R)) \leq \tau - d_{G_\ell}(\mu(R)) \tag{5.32}$$

By Lemma 5.11, $d_{G_\ell^{ext}}(\mu_\ell(R)) \geq \tau$. Combined with (5.32), we have

$$\tau \leq d_{G_\ell^{ext}}(\mu_\ell(R)) = \beta_\ell(\mu_\ell(R)) + d_{G_\ell}(\mu_\ell(R)) \leq \tau - d_{G_\ell}(\mu(R)) + d_{G_\ell}(\mu_\ell(R))$$

So $d_{G_\ell}(\mu(R)) \leq d_{G_\ell}(\mu_\ell(R))$. By Lemma 3.5, $\mu_\ell(R)$ is an R-$(T \setminus R)$ min cut in G_ℓ, so $d_{G_\ell}(\mu(R)) = d_{G_\ell}(\mu_\ell(R))$.

It remains to prove that $\mu_\ell(R)$ is defined. We prove by induction on ℓ. For the base case $\ell = 0$, $\mu(R)$ is defined because $R \in L$. Next consider the inductive case $\ell > 0$.

Assume for contradiction that there is no extreme set with terminal projection R in G_ℓ. Let X be an extreme violator of $\mu(R)$ in G_ℓ; then $X \subsetneqq \mu(R)$ and

$$d_{G_\ell}(X) \leq d_{G_\ell}(\mu(R)) \tag{5.33}$$

Because X is extreme, $\rho(X) \neq R$, and $\rho(X) \subsetneqq R$ is a node in the subtree of R in L. In the proof of Lemma 5.3, we proved that for an extreme set X with terminal set $\rho(X) \in P_j$ for some heavy path P_j, the cut value of X is augmented to τ just after processing heavy path P_j. Because X is in the subtree of R, and we stop the heavy paths at nodes in L_2, external edges assigned to R are added after path P_j. Let $\beta'(\cdot)$ be the external degree in the external edges added for heavy paths up to P_j, and $\beta'_\ell(\cdot)$ be the remaining β' degree in G_ℓ, so that

$$\beta'_\ell(X) + d_{G_\ell}(X) \geq \tau \tag{5.34}$$

The algorithm uses lower level degrees first, and uses upper level external edges assigned to R only after all lower level degrees in the subtree of R are used. As long as $\mu(R)$ has vacant degree, we have $\beta_\ell(\mu(R)) > \beta'_\ell(\mu(R)) \geq \beta'_\ell(X)$. Combined with Lemma 5.13 and (5.33),

$$\tau = \beta_\ell(\mu(R)) + d_{G_\ell}(\mu(R)) > \beta'_\ell(X) + d_{G_\ell}(X),$$

which contradicts (5.34). ∎

LEMMA 5.15. *After adding the ℓ-th chain, let u be a vertex with vacant degree and $U \in L_2$ be its representative leaf, then $u \in \mu_\ell(U)$.*

Proof. $u \in \mu(U)$ by Lemma 5.5 and the definition of representative leaf. $\mu(U)$ has a vacant degree at u in G_ℓ, so $\mu_\ell(U)$ is defined by Lemma 5.14. Assume for contradiction that $u \notin \mu_\ell(U)$. Then by Fact 5.3, $\mu_\ell(U) \subsetneqq \mu(U)$. Because one of the vacant degrees in $\beta_\ell(\mu(U))$ is at $u \notin \mu_\ell(U)$, $\beta_\ell(\mu(U)) > \beta_\ell(\mu_\ell(U))$.

$\mu(U)$ is not extreme in G_ℓ, so there exists an extreme violator W. $\rho(W) \subseteq U$, so $W \subseteq \mu_\ell(U)$, and $d_{G_\ell}(\mu(U)) \geq d_{G_\ell}(W) \geq d_{G_\ell}(\mu_\ell(U))$. By Lemma 5.13, $d_{G_\ell}(\mu(U)) + \beta_\ell(\mu(U)) = \tau$. Therefore

$$d_{G_\ell^{ext}}(\mu_\ell(U)) = d_{G_\ell}(\mu_\ell(U)) + \beta_\ell(\mu_\ell(U)) < d_{G_\ell}(\mu(U)) + \beta_\ell(\mu(U)) = \tau,$$

which contradicts Lemma 5.11. ∎

LEMMA 5.16. *If $\mu_\ell(R)$ is defined for some $R \in L_{high} \setminus L_2$, then $d_{G_\ell}(\mu_\ell(R)) = d_{G_\ell}(\mu_{\ell-1}(R))$.*

Proof. If $\mu_\ell(R) = \mu_{\ell-1}(R)$, the statement is trivial. By Fact 5.3, the remaining case is $\mu_\ell(R) \subsetneqq \mu_{\ell-1}(R)$. $d_{G_{\ell-1}}(\mu_\ell(R)) > d_{G_{\ell-1}}(\mu_{\ell-1}(R))$ since $\mu_{\ell-1}(R)$ is extreme in $G_{\ell-1}$, but $d_{G_\ell}(\mu_\ell(R)) \leq d_{G_\ell}(\mu_{\ell-1}(R))$ by Lemma 3.5. It remains to prove that $d_{G_\ell}(\mu_\ell(R)) \geq d_{G_\ell}(\mu_{\ell-1}(R))$. An augmentation chain adds at most 2 edges to any supreme set. So the only nontrivial case is that the ℓ-th chain adds 2 edges to $\mu_{\ell-1}(R) \setminus \mu_\ell(R)$ and no edge to $\mu_\ell(R)$.

For each endpoint u of the ℓ-th chain, $u \in \mu_{\ell-1}(U)$ by Lemma 5.15, where $U \in L_2$ is u's representative leaf. Assume the two edges are added to $X_1 = \mu_{\ell-1}(U_1)$ and $X_2 = \mu_{\ell-1}(U_2)$, $U_1, U_2 \in L_2$. $U_1, U_2 \subsetneqq R$ because $R \notin L_2$ and the two edges are added to $\mu_{\ell-1}(R)$. Also the two endpoints are not in $\mu_\ell(R)$, so $\mu_\ell(R)$ crosses X_1 and X_2.

Case 1: $U_1 \neq U_2$. Since X_1 and X_2 are extreme in $G_{\ell-1}$,

$$d_{G_{\ell-1}}(X_1 \cap \mu_\ell(R)) > d_{G_{\ell-1}}(X_1), d_{G_{\ell-1}}(X_2 \cap \mu_\ell(R)) > d_{G_{\ell-1}}(X_2)$$

By submodularity (2.4), $d_{G_{\ell-1}}(\mu_\ell(R) \cup X_1) < d_{G_{\ell-1}}(\mu_\ell(R))$, $d_{G_{\ell-1}}(\mu_\ell(R) \cup X_1 \cup X_2) < d_{G_{\ell-1}}(\mu_\ell(R) \cup X_1)$ (X_1 and X_2 are disjoint by laminarity). So

$$d_{G_\ell}(\mu_\ell(R)) = d_{G_{\ell-1}}(\mu_\ell(R)) \geq d_{G_{\ell-1}}(\mu_\ell(R) \cup X_1 \cup X_2) + 2 \geq d_{G_{\ell-1}}(\mu_{\ell-1}(R)) + 2 = d_{G_\ell}(\mu_{\ell-1}(R))$$

where the equations use the assumption that the ℓ-th chain adds 2 edges to $\mu_{\ell-1}(R)$ but no edge to $\mu_\ell(R)$.

Case 2: $U_1 = U_2$. By Lemma 5.14, either $\mu_\ell(U_1)$ exists and $d_{G_\ell}(\mu_\ell(U_1)) = d_{G_\ell}(\mu(U_1))$, or $\mu(U_1)$ has no vacant degree at time ℓ. Notice that $\rho(X_1 \cap \mu_\ell(R)) = U_1$. In the former case,

$$d_{G_{\ell-1}}(X_1 \cap \mu_\ell(R)) = d_{G_\ell}(X_1 \cap \mu_\ell(R)) \geq d_{G_\ell}(\mu_\ell(U_1)) = d_{G_\ell}(\mu(U_1)) = d_{G_{\ell-1}}(\mu(U_1)) + 2 = d_{G_{\ell-1}}(X_1) + 2$$

In the latter case, $\mu(U_1)$ has no vacant degree in G_ℓ, so $d_{G_{\ell-1}}(X_1 \cap \mu_\ell(R)) = d_{G_\ell}(X_1 \cap \mu_\ell(R)) \geq \tau$. By Lemma 5.13, $\tau = d_{G_{\ell-1}}(\mu(U_1)) + \beta_{\ell-1}(\mu(U_1)) \geq d_{G_{\ell-1}}(\mu(U_1)) + 2 = d_{G_{\ell-1}}(X_1) + 2$, so we also have $d_{G_{\ell-1}}(X_1 \cap \mu_\ell(R)) \geq d_{G_{\ell-1}}(X_1) + 2$. By submodularity,

$$d_{G_{\ell-1}}(\mu_\ell(R) \cup X_1) + 2 \leq d_{G_{\ell-1}}(\mu_\ell(R))$$

$\mu_\ell(R) \subseteq \mu_{\ell-1}(R)$ by Fact 5.3, and $X_1 = \mu_{\ell-1}(U_1) \subseteq \mu_{\ell-1}(R)$ by laminarity. Therefore, since $\mu_{\ell-1}(R)$ is extreme,

$$d_{G_\ell}(\mu_\ell(R)) = d_{G_{\ell-1}}(\mu_\ell(R)) \geq d_{G_{\ell-1}}(\mu_\ell(R) \cup X_1) + 2 \geq d_{G_{\ell-1}}(\mu_{\ell-1}(R)) + 2 = d_{G_\ell}(\mu_{\ell-1}(R))$$

In conclusion we have $d_{G_\ell}(\mu_\ell(R)) \geq d_{G_\ell}(\mu_{\ell-1}(R))$ in both cases, so $d_{G_\ell}(\mu_\ell(R)) = d_{G_\ell}(\mu_{\ell-1}(R))$. ∎

LEMMA 5.17. *After processing the ℓ-th chain, if a node R in L_{high} is not deleted, then either $\mu_\ell(R)$ is defined and $d_{G_\ell}(\mu(R)) = d_{G_\ell}(\mu_\ell(R))$, or $R \in L_2$ and $\mu(R)$ has no vacant degree in G_ℓ.*

Proof. We prove by induction first on time ℓ, then bottom-up on the forest. The base case $\ell = 0$ holds trivially, and the base case $R \in L_2$ is handled by Lemma 5.14. Next consider the inductive case $\ell > 0$ and R is an internal node in $L_{high} \setminus L_2$.

By inductive hypothesis, $\mu_{\ell-1}(R)$ is defined in $G_{\ell-1}$, and $d_{G_{\ell-1}}(\mu(R)) = d_{G_{\ell-1}}(\mu_{\ell-1}(R))$. $\mu(R) \supseteq \mu_{\ell-1}(R)$ by Fact 5.3. If chain ℓ does not add edges to $\mu(R)$, then $d_{G_\ell}(\mu(R)) = d_{G_\ell}(\mu_{\ell-1}(R))$. Next assume chain ℓ adds edges to $\mu(R)$. Because R is a forest node, for each chain edge (u, w), at most one endpoint u is assigned to a leaf U in the subtree of R. $u \in \mu_{\ell-1}(U) \subseteq \mu_{\ell-1}(R)$ by Lemma 5.15 and laminarity of supreme sets in $G_{\ell-1}$. The other endpoint w is assigned to leaves outside the tree of R, so $w \notin \mu(R)$. Therefore we also have $d_{G_\ell}(\mu(R)) = d_{G_\ell}(\mu_{\ell-1}(R))$ because they increase by the same amount from $G_{\ell-1}$. In both cases we have $d_{G_\ell}(\mu(R)) = d_{G_\ell}(\mu_{\ell-1}(R))$. By Lemma 5.16, if $\mu_\ell(R)$ is defined, then $d_{G_\ell}(\mu(R)) = d_{G_\ell}(\mu_{\ell-1}(R)) = d_{G_\ell}(\mu_\ell(R))$. It remains to prove that $\mu_\ell(R)$ is defined.

Assume for contradiction that there is no extreme set with terminal set R in G_ℓ. Let $X \subsetneqq \mu(R)$ be an extreme violator of $\mu(R)$ in G_ℓ with $d_{G_\ell}(X) \leq d_{G_\ell}(\mu(R))$. Let $W = \rho(X)$, then $W \subsetneqq R$ because X is extreme. $d_{G_\ell}(\mu(R)) < d_{G_\ell}(\mu(W))$ because R is not deleted, so

$$d_{G_\ell}(X) < d_{G_\ell}(\mu(W))$$

By inductive assumption, either $\mu_\ell(W)$ exists and $d_{G_\ell}(\mu_\ell(W)) = d_{G_\ell}(\mu(W))$, or $\mu(W)$ has no vacant degree. In the former case, $d_{G_\ell}(X) < d_{G_\ell}(\mu_\ell(W))$, which contradicts the fact that $\mu_\ell(W)$ is extreme. In the latter case, $X \subseteq \mu(W)$ because X is extreme, so X has no vacant degree. But by Lemma 5.13,

$$\tau = d_{G_\ell^{ext}}(\mu(W)) \geq d_{G_\ell}(\mu(W)) > d_{G_\ell}(X)$$

So X should have vacant degree in a feasible external solution, a contradiction. ∎

LEMMA 5.18. *If a node R is deleted after processing the ℓ-th chain, then $\mu_\ell(R)$ is not defined in G_ℓ.*

Proof. The algorithm only deletes internal nodes of L_{high}, so $R \in L_{high} \setminus L_2$. Because R is not deleted at time $\ell-1$, by Lemma 5.17, $d_{\ell-1}(\mu(R)) = d_{\ell-1}(\mu_{\ell-1}(R))$. If (u, w) is a chain edge and $u \in \mu(R)$, then w is not assigned to the tree of R and $w \notin \mu(R)$. By Lemma 5.15 and laminarity of supreme sets in $G_{\ell-1}$, all endpoints in $\mu(R)$ in the ℓ-th chain are in $\mu_{\ell-1}(R)$. Therefore the cut values of $\mu(R)$ and $\mu_{\ell-1}(R)$ increase by the same amount from $G_{\ell-1}$ to G_ℓ, and $d_{G_\ell}(\mu(R)) = d_{G_\ell}(\mu_{\ell-1}(R))$.

Assume for contradiction that $\mu_\ell(R)$ is defined. Then by Lemma 5.16, $d_{G_\ell}(\mu_{\ell-1}(R)) = d_{G_\ell}(\mu_\ell(R))$, so $d_{G_\ell}(\mu(R)) = d_{G_\ell}(\mu_\ell(R))$. Because R is deleted at time ℓ, $d_{G_\ell}(\mu(R)) \geq d_{G_\ell}(\mu(W))$ for some $W \in ch(R)$. By Lemma 3.5, $\mu_\ell(R)$ is an R-$(T \setminus R)$ min cut in G_ℓ, so

$$d_{G_\ell}(\mu_\ell(R) \cup \mu(W)) \geq d_{G_\ell}(\mu_\ell(R)) = d_{G_\ell}(\mu(R)) \geq d_{G_\ell}(\mu(W))$$

Since $\mu_\ell(R)$ is extreme in G_ℓ ($\rho(\mu_\ell(R) \cap \mu(W)) = W \subsetneqq R$),

$$d_{G_\ell}(\mu_\ell(R) \cap \mu(W)) > d_{G_\ell}(\mu_\ell(R))$$

Adding these two inequalities contradicts submodularity (2.1). ∎

LEMMA 5.19. *The chains picked by the algorithm are augmentation chains.*

Proof. By Lemma 5.12, the $c(R)$ value maintained by the algorithm equals $d_{G_\ell}(\mu(R))$ for all $R \in L_{high}$ at each time ℓ. By Lemma 5.17 and Lemma 5.18, the algorithm keeps a node R in L_{high} if and only if R remains a projection of an extreme set, or $R \in L_2$ and $\mu(R)$ has no vacant degree. For the latter case, $c(R) = d_{G_\ell}(\mu(R)) = \tau$ at time ℓ by Lemma 5.13. Therefore the roots R_i with $c(R_i) \leq \tau - 2$ picked by the algorithm are exactly the terminal sets of all maximal supreme sets with demand at least 2.

The algorithm picks endpoints with vacant degrees. By Lemma 5.15, the endpoints are in the current supreme sets of their representative leaves, which are in corresponding maximal supreme sets by laminarity. In conclusion the algorithm picks an augmentation chain. ∎

5.3 Augment Connectivity by One in the Degree-Constrained Setting Similar to the unconstrained degree case, we are left with the problem of augmenting Steiner connectivity by 1 (from $\tau - 1$ to τ) after augmentation chains have been added. As input to this problem, we are given the graph G with Steiner connectivity $\tau - 1$, degree constraint β, and the family K of the terminal sets of all supreme sets. We solve this problem by a reduction to the unconstrained setting and a modification of Algorithm 3.

The sets in K with demand 1 correspond to the terminal sets of minimal Steiner minimum cuts in G. These are also the nodes in the skeleton structure described in Section 4.4. Let K' be the sub-family of K of sets with demand 1. That is, each $k \in K'$ is the terminal set of a minimal Steiner min cut. We do not know the supreme sets of an arbitrary terminal projection, but in this special case we can find the supreme sets efficiently for terminal sets in K' using isolating cuts as follows. Given a graph with terminals, isolating cuts framework finds the minimum cuts separating each terminal from all other terminals in $O(\log n)$ max flow calls [37, 3].

Contract each set $k \in K'$, and also contract $T \setminus K'$. Run isolating cuts with these contracted nodes as terminals. By Lemma 3.5, for each $k \in K'$, $\mu(k)$ is the earliest k-$(T \setminus k)$ min cut. Therefore, if we find minimal isolating cuts, we get the supreme sets $\mu(k)$ for each $k \in K'$.

LEMMA 5.20. *If the degree-constrained augmenting connectivity by 1 problem is feasible, then for each $k \in K'$, there exists $v_k \in \mu(k)$ such that $\beta(v_k) \geq 1$.*

Proof. Because the problem is feasible and $\mu(k)$ is a Steiner cut, $d(\mu(k)) + \beta(\mu(k)) \geq \tau$, where $\beta(\mu(k)) = \sum_{u \in \mu(k)} \beta(u)$. $d(\mu(k)) = \tau - 1$ by definition of K', so there exists a $v_k \in \mu(k)$ with $\beta(v_k) \geq 1$. ∎

Recall that in Algorithm 3, we randomly match a pair of demand-1 extreme sets $\mu(k), \mu(k')$, and add a matching edge connecting arbitrary terminals in k and k'. Consider modifying the algorithm in the following way. For each $k \in K'$, choose a surrogate $v_k \in \mu(k)$ according to Lemma 5.20. When matching a pair of extreme sets $\mu(k)$ and $\mu(k')$, instead of connecting arbitrary terminals, we add an edge $(v_k, v_{k'})$ connecting the surrogates. Lemma 5.21 shows that this new matching on surrogates is an optimal solution.

LEMMA 5.21. *Given an instance of degree-constrained augmentation by 1 problem, remove the degree constraint and run Algorithm 3 to get an output F. If we replace every edge in F by its surrogate edge to form a new matching F', then F' is an optimal solution to the degree-constrained augmentation by 1 problem.*

Proof. For any Steiner min cut S, we prove that it is augmented by F', so that F' is a feasible solution. Because S is augmented by F, there exists an edge $(u, v) \in F$ across S, for some $u \in \mu(k), v \in \mu(k')$, where $\mu(k), \mu(k')$ are demand-1 supreme sets. Then $k, k' \in K'$. By Lemma 4.8, S does not cross $\mu(k)$ and $\mu(k')$. So for surrogates $v_k \in \mu(k)$ and $v_{k'} \in \mu(k')$, edge $(v_k, v_{k'}) \in F'$ also augments S. The surrogates have vacant degree by Lemma 5.20. Therefore F' is a feasible solution.

Notice that removing the degree constraint cannot decrease the optimal value, because an optimal solution to degree-constrained augmentation by 1 problem is also feasible to the corresponding unconstrained problem. Because F is an optimal solution to the unconstrained problem, and F' has the same value as F, F' is an optimal solution. ∎

Next we analyze the running time. Finding the surrogates v_k takes time $\tilde{O}(F(m, n))$ for isolating cuts and $O(n)$ for picking the surrogates. Running Algorithm 3 takes $\tilde{O}(F(m, n))$. So the total running time is $\tilde{O}(F(m, n))$.

References

[1] 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. In *FOCS 2022: 63rd IEEE Symposium on Foundations of Computer Science*, 2022.

[2] Amir Abboud, Robert Krauthgamer, and Ohad Trabelsi. Cut-equivalent trees are optimal for min-cut queries. In *Proceedings of the 61st IEEE Annual Symposium on Foundations of Computer Science*, FOCS 2020, pages 105–118. IEEE, 2020.

[3] 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*, STOC 2021, page 1725–1737, 2021.

[4] András A. Benczúr. Augmenting undirected connectivity in rnc and in randomized $\tilde{O}(n^3)$ time. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, STOC '94, page 658–667. ACM, 1994.

[5] András A. Benczúr and David R. Karger. Augmenting undirected edge connectivity in $\tilde{O}(n^2)$ time. *J. Algorithms*, 37(1):2–36, October 2000.

[6] Attila Bernáth, Satoru Iwata, Tamás Király, Zoltán Király, and Zoltán Szigeti. Recent results on well-balanced orientations. *Discrete Optimization*, 5(4):663–676, 2008.

[7] Anand Bhalgat, Ramesh Hariharan, Telikepalli Kavitha, and Debmalya Panigrahi. Fast edge splitting and edmonds' arborescence construction for unweighted graphs. In *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '08, page 455–464. SIAM, 2008.

[8] Jarosław Byrka, Fabrizio Grandoni, and Afrouz Jabal Ameli. Breaching the 2-approximation barrier for connectivity augmentation: a reduction to steiner tree. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 815–825, 2020.

[9] Guo-Ray Cai and Yu-Geng Sun. The minimum augmentation of any graph to a k-edge-connected graph. *Networks*, 19(1):151–172, 1989.

[10] Federica Cecchetto, Vera Traub, and Rico Zenklusen. Bridging the gap between tree and connectivity augmentation: unified and stronger approaches. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 370–383, 2021.

[11] 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, 2022.

[12] Ruoxu Cen, Jason Li, and Debmalya Panigrahi. Edge connectivity augmentation in near-linear time. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2022, pages 137–150. ACM, 2022.

[13] Yuk Hei Chan, Wai Shing Fung, Lap Chi Lau, and Chun Kong Yung. Degree bounded network design with metric costs. *SIAM Journal on Computing*, 40(4):953–980, 2011.

[14] Chandra Chekuri and Nitish Korula. A graph reduction step preserving element-connectivity and packing steiner trees and forests. *SIAM Journal on Discrete Mathematics*, 28(2):577–597, 2014.

[15] Chandra Chekuri and F. Bruce Shepherd. Approximate integer decompositions for undirected network design problems. *SIAM Journal on Discrete Mathematics*, 23(1):163–177, 2009.

[16] 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. In *FOCS 2022: 63rd IEEE Symposium on Foundations of Computer Science*, 2022.

[17] Yefim Dinitz and Alek Vainshtein. The connectivity carcass of a vertex subset in a graph and its incremental maintenance. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, STOC '94, page 716–725. ACM, 1994.

[18] András Frank. Augmenting graphs to meet edge-connectivity requirements. *SIAM Journal on Discrete Mathematics*, 5(1):25–53, February 1992.

[19] András Frank and Zoltán Király. Graph orientations with edge-connection and parity constraints. *Combinatorica*, 22(1):47–70, 2002.

[20] András Frank and Tibor Jordán. Directed vertex-connectivity augmentation. *Mathematical Programming*, 84(3):537–553, 1999.

[21] András Frank and Tamás Király. Combined connectivity augmentation and orientation problems. *Discrete Applied Mathematics*, 131(2):401–419, 2003.

[22] Greg N. Frederickson and Joseph Ja'Ja'. Approximation algorithms for several graph augmentation problems. *SIAM Journal on Computing*, 10(2):270–283, 1981.

[23] Harold N. Gabow. Efficient splitting off algorithms for graphs. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, STOC '94, page 696–705. ACM, 1994.

[24] Harold N. Gabow. The minset-poset approach to representations of graph connectivity. *ACM Trans. Algorithms*, 12(2):24:1–24:73, 2016.

[25] Zhihan Gao. On the metric s–t path traveling salesman problem. *SIAM Review*, 60(2):409–426, 2018.

[26] Fabrizio Grandoni, Afrouz Jabal Ameli, and Vera Traub. Breaching the 2-approximation barrier for the forest augmentation problem. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1598–1611, 2022.

[27] Fabrizio Grandoni, Christos Kalaitzis, and Rico Zenklusen. Improved approximation for tree augmentation: saving by rewiring. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 632–645, 2018.

[28] Ramesh Hariharan, Telikepalli Kavitha, and Debmalya Panigrahi. Efficient algorithms for computing all low *s-t* edge connectivities and related problems. In Nikhil Bansal, Kirk Pruhs, and Clifford Stein, editors, *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA 2007, pages 127–136. SIAM, 2007.

[29] Toshimasa Ishii. Minimum augmentation of edge-connectivity with monotone requirements in undirected graphs. *Discrete Optimization*, 6(1):23–36, 2009.

[30] Toshimasa Ishii and Masayuki Hagiwara. Minimum augmentation of local edge-connectivity between vertices and vertex subsets in undirected graphs. *Discrete Applied Mathematics*, 154(16):2307–2329, 2006.

[31] Bill Jackson and Tibor Jordán. Independence free graphs and vertex connectivity augmentation. *Journal of Combinatorial Theory, Series B*, 94(1):31–77, 2005.

[32] Kamal Jain. A factor 2 approximation algorithm for the generalized steiner network problem. *Combinatorica*, 21(1):39–60, 2001.

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

[34] Kristine Vitting Klinkby, Pranabendu Misra, and Saket Saurabh. Strong connectivity augmentation is fpt. In *Proceedings of the 32nd Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '21, page 219–234, USA, 2021. SIAM.

[35] Lap Chi Lau. An approximate max-steiner-tree-packing min-steiner-cut theorem. *Combinatorica*, 27(1):71–90, 2007.

[36] Lap Chi Lau and Chun Kong Yung. Efficient edge splitting-off algorithms maintaining all-pairs edge-connectivities. *SIAM J. Comput.*, 42(3):1185–1200, 2013.

[37] Jason Li and Debmalya Panigrahi. Deterministic min-cut in poly-logarithmic max-flows. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 85–92. IEEE, 2020.

[38] 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*, STOC 2021, page 1738–1748. ACM, 2021.

[39] László Lovász. *Combinatorial Problems and Exercises*. North-Holland Publishing Company, Amsterdam, 1979.

[40] W. Mader. A reduction method for edge-connectivity in graphs. In B. Bollobás, editor, *Advances in Graph Theory*, volume 3 of *Annals of Discrete Mathematics*, pages 145–164. Elsevier, 1978.

[41] Dániel Marx and László A. Végh. Fixed-parameter algorithms for minimum-cost edge-connectivity augmentation. *ACM Trans. Algorithms*, 11(4), apr 2015.

[42] Martin Merker. Decomposing highly edge-connected graphs into homomorphic copies of a fixed tree. *Journal of Combinatorial Theory, Series B*, 122:91–108, 2017.

[43] Hiroshi Nagamochi and Toshihide Ibaraki. Deterministic õ(nm) time edge-splitting in undirected graphs. *Journal of Combinatorial Optimization*, 1(1):5–46, 1997.
[44] Dalit Naor, Dan Gusfield, and Charles U. Martel. A fast algorithm for optimally increasing the edge connectivity. *SIAM J. Comput.*, 26(4):1139–1165, 1997.
[45] Zoltán Szigeti. Edge-splittings preserving local edge-connectivity of graphs. *Discrete Applied Mathematics*, 156(7):1011–1018, 2008.
[46] Vera Traub and Rico Zenklusen. A better-than-2 approximation for weighted tree augmentation. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1–12. IEEE, 2022.
[47] László A. Végh and András A. Benczúr. Primal-dual approach for directed vertex connectivity augmentation and generalizations. *ACM Trans. Algorithms*, 4(2), may 2008.
[48] Toshimasa Watanabe and Akira Nakamura. Edge-connectivity augmentation problems. *Journal of Computer and System Sciences*, 35(1):96–144, 1987.

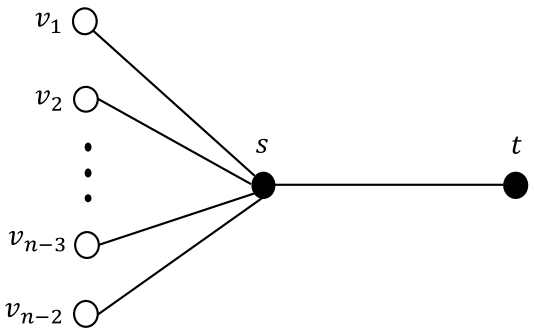


Figure 3: A graph with exponentially many extreme sets. The solid vertices s, t are terminals and the empty vertices $v_1, v_2, \ldots, v_{n-2}$ are nonterminals.

A Example of Graph with Exponentially Many Extreme Sets

The example is given in Figure 3. All vertex sets of the form $\{s\} \cup \{v_i : i \in S\}$ for any $S \subseteq [n-2]$ are extreme sets. This is because their cut size is $n - |S| + 1$ whereas any strict subset containing terminals must be of the form $\{s\} \cup \{v_i : i \in S'\}$ for some $S' \subset S$. The latter vertex set has cut value $n - |S'| + 1$, which is strictly larger than $n - |S| + 1$.

B Augmenting Connectivity by Matching: Illustrative Example of a Cycle

We are interested in how to augment the connectivity of a cycle from 2 to 3. Our running example will be the cycle graph on 8 vertices, which we call C_1. Figure 4a shows a case in which we add one edge to a cycle graph as a partial solution. Then we claim that after adding the edge (v_1, v_3), in order to augment the connectivity of C_1 to 3, it suffices to augment the connectivity of C_2 to 3. This is because the presence of the (v_1, v_3) edge allows us to shortcut the edges (v_2, v_8) and (v_2, v_4), since all cuts of value 2 in $C_1 \cup \{(v_1, v_3)\}$ are accounted for as cuts of value 2 in C_2.

The same reasoning applies to any edge (v_i, v_j) between non-adjacent vertices in the cycle. However, notice that since v_i and v_{i+1} are adjacent in C_1, adding the edge (v_i, v_{i+1}) does not allow us to shortcut both vertices by deleting v_i and v_{i+1} and adding the edge (v_{i-1}, v_{i+2}), since $\{v_i, v_{i+1}\}$ would be a cut of value 2 not accounted for in the shortcut version.

In Algorithm 3 we actually add edges in batches of $\lceil \frac{n}{4} \rceil$ edges at a time, where n is the number of vertices in the cycle, so it is important to understand which edges are compatible with each other. Consider the example in Figure 4b. After adding the edge (v_1, v_3) we can delete and shortcut those two vertices as discussed previously, and effectively work with C_2 instead. Now, the second edge given in the example is (v_2, v_5), which in C_2 connects two non-adjacent vertices. Then, applying the same reasoning, we can delete and shortcut v_2 and v_5, and work with C_3 from Figure 4b instead. This is because the two edges to be added were compatible with each other in the sense that there is a sequence of them such that in the sequence of deletions and shortcuts, each edge connects non-adjacent vertices in the sequence of simplified cycles.

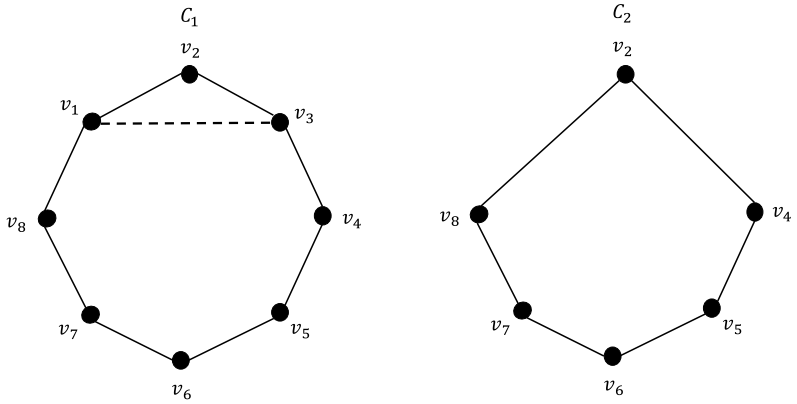
On the other hand, Figure 4c displays an example in which two edges are not compatible. That is, once we add one edge and apply deletion and the shortcut, the other edge connects adjacent vertices in the resulting graph.

In Section 4.4 we show that any compatible sequence of additional edges can be completed to an optimal connectivity augmentation of any graph by 1. Then, we analyze the probability that a randomly selected partial matching is compatible in the sense that we have just described.

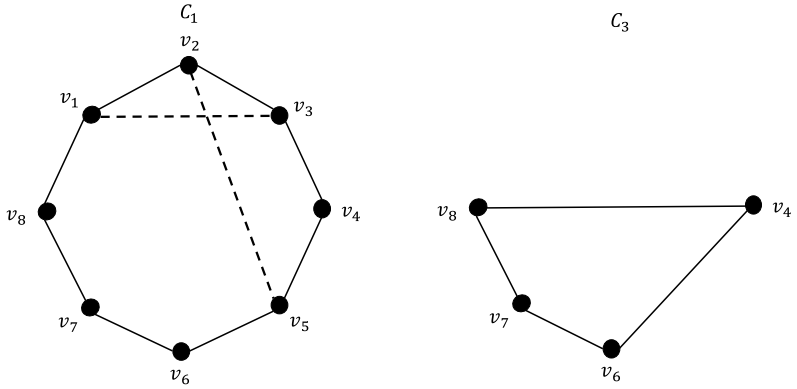
C Data Structures for Lazy Updates

In this section, we sketch the running time analysis using lazy data structures in Cen et al. [11] for maintaining supreme sets tree while the algorithm adds augmentation chains. That gives proofs for Theorem 4.2 and Theorem 5.1.

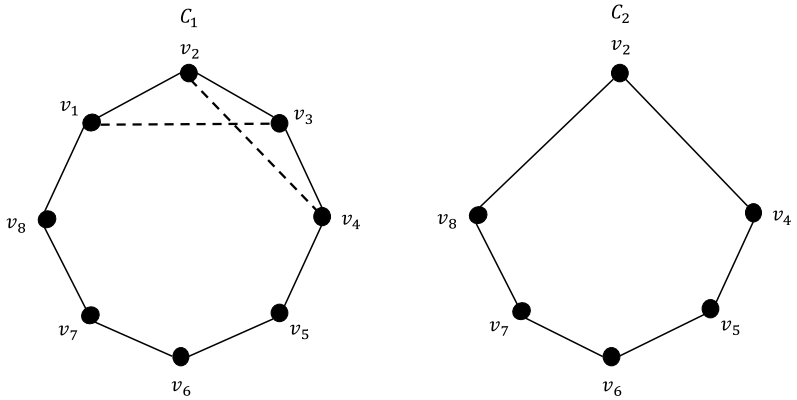
Proof. [Proof of Theorem 4.2] We begin with an initial augmentation chain $\{(a_i, b_{i+1})\}_{1 \le i \le r-1}$ with $a_i, b_i \in R_i$,



(a) Here C_1 is augmented by one edge $v_1 - v_3$. After this edge is added, augmenting C_2 to connectivity 3 gives an augmentation of C_1 to connectivity 3, so we have reduced our problem to an augmentation on a smaller graph.



(b) First add $v_1 - v_3$ to C_1 to result in a reduction of augmenting C_1 to connectivity 3 to augmenting C_2 as in Figure 4a. Then adding the edge $v_2 - v_5$ to C_2 reduces the augmentation of C_1 to connectivity 3 further to augmenting C_3 to connectivity 3.



(c) After adding the edge $v_1 - v_3$ we reduce augmenting C_1 to augmenting C_2. But now, the edge $v_2 - v_4$ does not result in a good reduction of augmentation of C_2 to a smaller problem because v_2 and v_4 are adjacent.

Figure 4: Augmenting the connectivity of a cycle by 1

where each R_i is a root in the supreme sets forest with demand at least 2. This chain can be calculated in $O(n)$ time. We add it until it is no longer an augmentation chain, in which case we say it expires. This happens when one of the following occurs:

1. Some vertex a_i uses up all of its vacant degree.
2. For some $i \le r$, $\mathrm{dem}(R_i) \le 1$.
3. For some i and $W \in \mathrm{ch}(R_i)$, $d(W) \ge d(R_i)$.

At this point, we replace $\{(a_i, b_{i+1})\}_{1 \le i \le r-1}$ with a new augmentation chain $\{(a'_i, b'_{i+1})\}_{1 \le i \le r-1}$ and keep using the new chain until it expires again. Each of the three cases can occur at most n times because each vertex can use up degree once, and each node in the supreme sets forest can be deleted once.

Given that an expiration occurred, we first show how to select a new augmentation chain. If expiration occurred because of Case 1, we replace the terminal a_i with no more vacant degree by another vacant vertex in the same supreme set. This takes constant time. If Case 2 occurs so that some X_i has demand at most 1, then we remove the edges (a_i, b_{i+1}) and (a_{i-1}, b_i) from the augmentation chain and replace them by an edge (a_{i-1}, b_{i+1}). This also takes constant time. If Case 3 occurs for some R_i, then we replace (a_i, b_{i+1}) and (a_{i-1}, b_i) by a partial augmentation chain connecting all children of R_i. The running time is proportional to the number of children. Since each node is added at most once, the total cost of Case 3 is $O(n)$.

When some edge expires and gets updated, other chain edges remain and contribute to the cut values. So instead of maintaining the exact cut values of the supreme sets, we maintain them lazily by recording the linear growth (in the number of times an augmentation chain is added) of cut values. The expiration time does not change as long as the incident chain edges remain the same. When an edge of the augmentation chain is modified, we calculate the new expiration time of the chain as follows in time $O(\log(n))$. Each update of chain edges costs constant data structure operations, and there are $O(n)$ chain updates in total, so the total running time is $O(n \log n)$.

Now we show how to determine the expiration time of a new augmentation chain given by modifying one edge of an old augmentation chain. This is easy given the correct cut values for all supreme terminal sets, because we have that the expiration time for the chain F is $t_F = \min(t_1, t_2, t_3)$, where

$$t_1 = \min_{v \in V} \lfloor \frac{\beta(v)}{d_F(v)} \rfloor,$$
$$t_2 = \min_{i \in [r]} \lfloor \frac{\tau - d(X_i)}{d_F(X_i)} \rfloor,$$
$$t_3 = \min_{i \in [r]} \min_{W \in \mathrm{ch}(X_i)} \lceil \frac{d(W) - d(X_i)}{d_F(X_i) - d_F(W)} \rceil.$$

The values t_1, t_2, t_3 can be maintained by priority queues, so finding these minimums can be done in $O(\log(n))$ time.

Finally, we need to maintain the expiration time under an update of augmentation chain. When adding a new chain edge, we first carry out previous lazy updates of relevant nodes to the current time, and then apply new lazy updates. The updates are uniformly adding a constant to the vacant degree and cut values of nodes on a tree path, so they can be handled in constant dynamic tree operations. (For each edge (a_i, b_{i+1}) in the chain F, the supreme sets that have their corresponding cut values increased are those on the forest path in L from a_i to b_{i+1} excluding the least common ancestor of a_i and b_{i+1}.) □

Proof. [Proof of Theorem 5.1] We show that the degree-constrained augmentation algorithm behaves in the same way as the unconstrained augmentation algorithm in terms of maintaining the supreme sets tree while greedily adding augmentation chains, despite several differences in algorithm details. It follows that we can use the same lazy data structures as in the unconstrained setting to get $\tilde{O}(n)$ running time.

We summarize the differences and remedies below.

1. The algorithm works on the forest L_{high} instead of L. So the data structure maintains the part of supreme sets forest corresponding to L_{high}.

2. Because the chain edges incident on nonterminals and the supreme sets will change during the algorithm, we do not know whether an edge contributes to a supreme sets in degree-constrained setting. However, Lemma 5.15 shows that we can locate each vacant degree in a current supreme set projected to a node in L_2, so by laminarity of current supreme sets, we know the contribution of an edge to all supreme sets of projection in L_{high}.

3. The algorithm needs to remove a forest node when it is no longer the terminal projection of a supreme set. By Lemma 5.17 and Lemma 5.18, this is detected by the algorithm. Moreover, the operation is the same as in unconstrained setting: by comparing whether $c(R) \geq c(W)$ for any node R and its child W. So the same data structures can be applied to maintain the forest structure in degree-constrained setting.

□