Connectivity Labeling Schemes for Edge and Vertex Faults via Expander Hierarchies

Yaowei Long
University of Michigan

Seth Pettie*
University of Michigan

Thatchaphol Saranurak†
University of Michigan

Abstract

We consider the problem of assigning short *labels* to the vertices and edges of a graph G so that given any query $\langle s, t, F\rangle$ with $|F| \leq f$, we can determine whether s and t are still connected in $G - F$, given only the labels of $F \cup \{s, t\}$.

This problem has been considered when $F \subset E$ (edge faults), where correctness is guaranteed with high probability (w.h.p.) [DP21] or deterministically [IEWM23], and when $F \subset V$ (vertex faults), both w.h.p. and deterministically [PP22, PPP24]. Our main results are as follows.

Deterministic Edge Faults. We give a new deterministic labeling scheme for edge faults that uses $\tilde{O}(\sqrt{f})$-bit labels, which can be constructed in polynomial time. This improves on Dory and Parter's [DP21] *existential* bound of $O(f \log n)$ (requiring exponential time to compute) and the efficient $\tilde{O}(f^2)$-bit scheme of Izumi, Emek, Wadayama, and Masuzawa [IEWM23]. Our construction uses an improved edge-expander hierarchy and a distributed coding technique based on Reed-Solomon codes.

Deterministic Vertex Faults. We improve Parter, Petruschka, and Pettie's [PPP24] deterministic $O(f^7 \log^{13} n)$-bit labeling scheme for vertex faults to $O(f^4 \log^{7.5} n)$ bits, using an improved vertex-expander hierarchy and better sparsification of shortcut graphs. We completely bypass deterministic graph sketching [IEWM23] and hit-and-miss families [KP21].

Randomized Edge/Verex Faults. We improve the size of Dory and Parter's [DP21] randomized edge fault labeling scheme from $O(\min\{f + \log n, \log^3 n\})$ bits to $O(\min\{f + \log n, \log^2 n \log f\})$ bits, shaving a $\log n/\log f$ factor. We also improve the size of Parter, Petruschka, and Pettie's [PPP24] randomized vertex fault labeling scheme from $O(f^3 \log^5 n)$ bits to $O(f^2 \log^6 n)$ bits, which comes closer to their $\Omega(f)$-bit lower bound [PPP24].

*Supported by NSF Grant CCF-2221980.
†Supported by NSF Grant CCF-2238138.

Contents

1 Introduction

A *labeling scheme* for a graph problem can be viewed as a distributed data structure in which all queries must be answered without inspecting the underlying graph, but only the *labels* of the query arguments. Early work focused on labeling schemes for *adjacency* [Bre66, BF67, KNR92], which is connected to finding small *induced universal graphs* [ADK17, AKTZ19].

There are now many labeling schemes for basic navigation queries in rooted trees [ADK17, AAK+06, AHL14], such as adjacency, ancestry, and least common ancestors. There are labeling schemes for computing *distances* in general graphs [AGHP16a], planar graphs [GPPR04, BGP22, GU23], and trees [GPPR04, ABR05, AGHP16b], as well as labelings for approximate distances [TZ05, AG11]. There are several labeling schemes for answering queries about the pairwise edge- and vertex-connectivity in undirected graphs [KKKP04, HL09, IN12, PSY22].

Label Schemes under Faults. Courcelle and Twigg [CT07, CGKT08] initiated the study of *forbidden set* or *fault tolerant* labeling schemes. The idea is to support a standard connectivity/distance query, subject to *faults* (deletions) of some subset F of vertices or edges. Several fault tolerant labeling schemes focus on special graph classes such as bounded treewidth graphs [CT07] and planar graphs [CGKT08, ACGP16, BCG+22, CMW23]. On general graphs, labeling schemes that handle at most one or two faults were shown for single-source reachability [Cho16] and single-source approximate distance [BCHR20].

The first labeling scheme on general graphs under multiple faults was given by Dory and Parter [DP21], for connectivity under edge faults. More precisely, they assigned labels to edges and vertices of an undirected n-vertex graph G so that, given the labels of $\langle s, t, F\rangle$ where $F \subset E(G)$, one can determine if s and t are still connected in $G - F$. When $|F| \le f$, they gave a Monte Carlo randomized construction of labels of size $O(\min\{f + \log n, \log^3 n\})$ bits that answer *each* query correctly with high probability. By increasing the size to $O(f \log n)$ bits, their scheme answers *all* queries correctly, with high probability, though confirming this property seems to require an exponential time brute force search. Recently, Izumi, Emek, Wadayama, and Masuzawa [IEWM23] gave a deterministic polynomial-time construction of labels of size $\tilde{O}(f^2)$.

Parter and Petruschka [PP22] considered the same problem, but with *vertex* faults rather than edge faults, i.e., $F \subset V(G)$. They gave deterministic labeling schemes with length $O(\log n)$ for $f = 1$, $O(\log^3 n)$ for $f = 2$, and a randomized scheme of length $\tilde{O}(n^{1-1/2^{f-2}})$ for general f. This year, Parter, Petruschka, and Pettie [PPP24] developed randomized and deterministic labeling schemes with label length $\tilde{O}(f^3)$ and $\tilde{O}(f^7)$, respectively. They also observed an $\Omega(f + \log n)$-bit lower bound for vertex faults (randomized or deterministic), which established a complexity separation between edge and vertex faults. See Table 1.

Our Results. We show new connectivity labels under both edge and vertex faults that improve the state-of-the-art as follows:

1. Deterministic labels under edge faults of size $\tilde{O}(\sqrt{f})$ bits (Theorem 2.1). This simultaneously improves Dory and Parter's [DP21] labels of size $O(f \log n)$, which require exponential time to construct, and Izumi et al.'s [IEWM23] efficiently constructed labels of size $\tilde{O}(f^2)$. In fact, Izumi et al. [IEWM23] stated that "it seems plausible that the $\Omega(f)$-bit lower bound holds." We refute this possibility.

Edge Fault Tolerant Connectivity Labels

Reference	Label Size (Bits)	Guarantee	Notes
Dory & Parter [DP21]	$O(\min\{f+\log n, \log^3 n\})$	Monte Carlo	Query correct w.h.p.
	$O(f \log n)$	Deterministic	*Existential bound*
Izumi, Emek, Wadayama & Masuzawa [IEWM23]	$O(f^2 \log^2 n \log\log n)$	Deterministic	Polynomial construction
	$O(f^2 \log^3 n)$		$\tilde{O}(mf^2)$ construction
Trivial	$\Omega(\log n)$	any	trivial lower bound
new	$O(\min\{f+\log n, \log^2 n \log(f/\log^2 n)\})$	Monte Carlo	Query correct w.h.p.
	$O(\sqrt{f} \log^2 n \log f)$	Deterministic	*Existential bound*
	$O(\sqrt{f} \log^{2.25} n \log(f \log n))$		Polynomial construction

Vertex Fault Tolerant Connectivity Labels

Reference	Label Size (Bits)	Guarantee	Notes
Parter & Petruschka [PP22]	$O(\log^3 n)$	Deterministic	$f \leq 2$
	$\tilde{O}(n^{1-2^{-f+2}})$	Monte Carlo	$f \in [3, o(\log\log n)]$
Parter, Petruschka & Pettie [PPP24]	$O(f^3 \log^5 n)$	Monte Carlo	Query correct w.h.p.
	$O(f^7 \log^{13} n)$	Deterministic	Polynomial construction
	$\Omega(f + \log n)$	any	lower bound
new	$O(f^2 \log^6 n)$	Monte Carlo	Query correct w.h.p.
	$O(f^4 \log^7 n)$	Deterministic	*Existential bound*
	$O(f^4 \log^{7.5} n)$		Polynomial construction

Table 1: All *Monte Carlo* results have a one-sided error probability of $1/\text{poly}(n)$, i.e., they may report two vertices disconnected when they are, in fact, connected. The *existential* result of Dory and Parter [DP21] constructs labels in $O(mf \log n)$ time that, with high probability, answer all queries correctly. However, to verify this fact requires a brute force search. The new existential results require solving an NP-hard problem, namely *sparsest cut*.

2. Deterministic labels under vertex faults of size $O(f^4 \log^{7.5} n)$ bits (Theorem 3.1). This improves the $O(f^7 \log^{13} n)$-bit labels of [PPP24].

3. Randomized labels under edge faults of size $O(\min\{f + \log n, \log^2 n \log f\})$ bits (Theorems 4.1 and 4.4). This improves the $O(\min\{f + \log n, \log^3 n\})$-bit labels of Dory and Parter [DP21].

4. Randomized labels under vertex faults of size $O(f^2 \log^6 n)$ bits (Theorem 5.1). This improves the $O(f^3 \log^5 n)$-bit labels of Parter et al. [PPP24].

Related Work: Connectivity Oracles. Our connectivity labels can be viewed as a distributed version of *connectivity oracles under faults*. In this problem, we must build a centralized data structure for an input graph G so that, given a query $\langle s, t, F\rangle$, we can check if s and t are still connected in $G - F$ as fast as possible using the centralized data structure. Connectivity oracles have been well-studied under both edge faults [PT07, DP20, GKKT15] and vertex faults [DP20, vdBS19, LS22, PSS$^+$22, Kos23, LW24] and the optimal preprocessing/query bounds have been proven [PT07, LS22, DP20, KPP16, HKNS15], either unconditionally, or conditioned on standard fine-grained complexity assumptions.

Our Techniques. The basis of Results 1 and 2 is an *expander hierarchy*. This is the first application of expanders in the context of fault-tolerant labeling schemes, although they have been widely applied in the centralized dynamic setting, e.g. [NSWN17, GRST21, LS22].

We give a clean definition of expander hierarchies for both *edge expansion* (Definition 2.2) and *vertex expansion* (Definition 3.2) as well as simple algorithms for computing them. In the edge expansion version, our formulation turns out to be equivalent to Pătraşcu and Thorup's [PT07], but our algorithm improves the quality by a $\Theta(\log n)$ factor. Our vertex expander hierarchy is new. Combined with the observation that any ϕ-vertex-expanding set has an $O(1/\phi)$-degree Steiner tree (Lemma 3.6),[1] this implies a new *low-degree hierarchy* that is strictly stronger and arguably cleaner than all previous low-degree hierarchies [DP20, LS22, LW24, PPP24], which are the critical structures behind vertex fault tolerant connectivity oracles.

To obtain our deterministic labels under edge faults, we first show that the edge expander hierarchy immediately leads to a simple $\tilde{O}(f)$-bit label (Theorem 2.5), which already improves the state-of-the-art [IEWM23, DP21]. Then, we introduce a new distributed coding technique based on Reed-Solomon codes to improve the label size to $\tilde{O}(\sqrt{f})$ bits, obtaining Result 1.

Our deterministic labeling scheme under vertex faults (Result 2) employs a high-level strategy from Parter et al. [PPP24], but our label is shorter by an $\tilde{\Theta}(f^3)$ factor. Roughly, this improvement comes from two sources. First, Parter et al. [PPP24] employed the deterministic graph cut sketch of Izumi et al. [IEWM23], which contributes an $\tilde{\Theta}(f^2)$ factor to the size. We can bypass deterministic sketching and pay only an $\tilde{O}(f)$ factor because our low-degree hierarchy has an additional vertex expansion property. Second, Parter et al. [PPP24] constructed a *sparsified shortcut graph* with arboricity $\tilde{O}(f^4)$ using the hit-and-miss families by Karthik and Parter [KP21]. We are able to use the simpler Nagamochi-Ibaraki sparsification [NI92] to obtain a sparse shortcut graph with arboricity $\tilde{O}(f^2)$. These two improvements cannot be applied in a modular way, so our final scheme ends up being rather different from [PPP24].

[1] Previously, [LS22] showed an $O(\log(n)/\phi)$-degree Steiner tree spanning any ϕ-vertex-expanding set.

All previous centralized connectivity oracles under vertex faults (including the one using vertex expanders [LS22]) crucially used 2D-range counting data structures, which seem inherently incompatible with the distributed labeling setting. Thus, our scheme is inherently different than the centralized oracles [LS22, DP20].

2 Deterministic Edge Fault Connectivity Labels

The goal of this section is to prove the following theorem.

Theorem 2.1. *Fix any undirected graph $G = (V, E)$ and integer $f \geq 1$. There are deterministic labeling functions $L_V : V \to \{0,1\}^{\log n}$ and $L_E : E \to \{0,1\}^{O(\sqrt{f/\phi}\log(f/\phi)\log^2 n)}$ such that given any query $\langle s, t, F\rangle$, $F \subset E$, $|F| \leq f$, one can determine whether s and t are connected in $G - F$ by inspecting only $L_V(s), L_V(t), \{L_E(e) \mid e \in F\}$. The construction time is exponential for $\phi = 1/2$ and polynomial for $\phi = \Omega(1/\sqrt{\log n})$.*

We remark that the above labeling scheme is actually more flexible. By reading only the labels of the failed edges F, it can compute a representation of connected components of $G - F$ in $\mathrm{poly}(f \log n)$ time. From this representation, we can, for example, count the number of connected components in $G - F$. This is impossible in the vertex-failure setting for any vertex-labels of size $o(n^{1-1/f}/f)$. See Section 6. Given the additional labels of s and t, we can then check whether s and t are connected in $G - F$, in $O(1 + \min\{\frac{\log\log n}{\log\log\log n}, \frac{\log f}{\log\log n}\})$ time. We can also straightforwardly handle edge insertions.

To prove Theorem 2.1, we introduce two new tools into the context of labeling schemes. The first tool is the *edge expander hierarchy*, for which we give an improved construction in Section 2.1. This tool alone already leads to a simple and efficient deterministic labeling scheme of size $\tilde{O}(f)$ bits, improving prior work [IEWM23, DP21]. In Section 2.3 we introduce a second tool, distributed *code shares*, and in Section 2.4 we combine the two tools and prove Theorem 2.1.

2.1 First Tool: Edge Expander Hierarchies

In this section we recall Pătraşcu and Thorup's [PT07] definition of an expander hierarchy, then give a new construction that improves the quality by a factor of $\Theta(\log n)$.

Given a graph $G = (V, E)$, a set $X \subseteq E$ and a vertex u, let $\mathsf{Deg}_X(u)$ denote the number of edges from X incident to u and let $\mathsf{Deg}_X(S) = \sum_{u\in S}\mathsf{Deg}_X(u)$ denote the *volume* of S with respect to X. We say that X is *ϕ-expanding* in G if, for every cut $(S, V \setminus S)$,

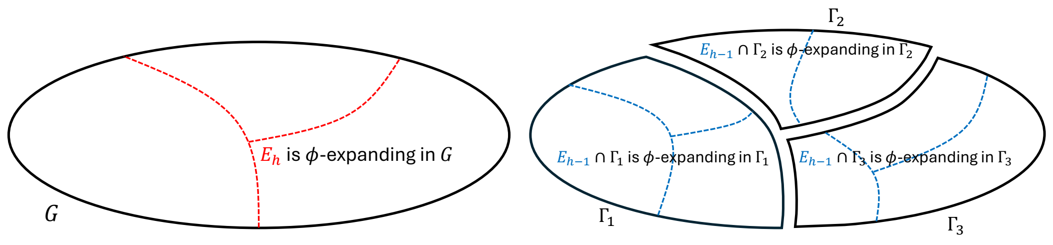
$$|E_G(S, V \setminus S)| \geq \phi \min\{\mathsf{Deg}_X(S), \mathsf{Deg}_X(V \setminus S)\}.$$

Consider a partition $\{E_1, \ldots, E_h\}$ of E. We denote $E_{\leq\ell} := \bigcup_{i\leq\ell} E_i$, $E_{>\ell} := \bigcup_{i>\ell} E_i$, and $G_{\leq\ell} := G \cap E_{\leq\ell}$. We also write $\mathsf{Deg}_\ell := \mathsf{Deg}_{E_\ell}$, $\mathsf{Deg}_{\leq\ell} := \mathsf{Deg}_{E_{\leq\ell}}$, and so on.

Definition 2.2 (Expander Hierarchy)**.** Given a graph $G = (V, E)$, an edge-partition $\mathcal{P} = \{E_1, \ldots, E_h\}$ of E *induces an (h, ϕ)-expander hierarchy* of G if, for every level $\ell \leq h$ and every connected component Γ of $G_{\leq\ell}$, $E_\ell \cap \Gamma$ is ϕ-expanding in Γ. That is, for every cut $(S, \Gamma \setminus S)$ of Γ, we have

$$|E_{\leq\ell}(S, V(\Gamma) \setminus S)| \geq \phi \min\{\mathsf{Deg}_\ell(S), \mathsf{Deg}_\ell(V(\Gamma) \setminus S)\}.$$

Over all levels ℓ, the set of all connected components Γ of $G \setminus E_{>\ell}$ form a laminar family $\mathcal{C}$. Let $\mathcal{H}$ be the tree representation of $\mathcal{C}$. We also call $(\mathcal{C}, \mathcal{H})$ an (h, ϕ)-expander hierarchy of G.



E_h is ϕ-expanding in G
G
Γ_2
$E_{h-1} \cap \Gamma_2$ is ϕ-expanding in Γ_2
$E_{h-1} \cap \Gamma_1$ is ϕ-expanding in Γ_1
$E_{h-1} \cap \Gamma_3$ is ϕ-expanding in Γ_3
Γ_1
Γ_3

Figure 1: Illustration of the top two levels of an (h, ϕ)-expander hierarchy. E_h and E_{h-1} are drawn in red and blue, respectively.

See Figure 1 for an example. Below, we give an improved construction of the expander hierarchy.

Theorem 2.3. *There exists an algorithm that, given a graph G, computes an (h, ϕ)-expander hierarchy with $h \le \log n$ and $\phi = 1/2$ in exponential time, or $\phi \ge \Omega(1/\sqrt{\log n})$ in polynomial time.*

Pătraşcu and Thorup [PT07] gave an exponential-time algorithm for $h \le \log m$ and $\phi = 1/(2\log n)$ and a polynomial-time algorithm with $\phi \ge \Omega(1/\log^{1.5} n)$. Theorem 2.3 shaves one $\log n$ factor in ϕ for both settings.[2]

Theorem 2.3 is perhaps surprising. Recall a related and seemingly weaker concept of ϕ-expander decomposition. A ϕ-expander decomposition of a graph G is an edge set $X \subseteq E$ such that, for each connected component Γ of $G \setminus X$, $E \cap \Gamma$ is ϕ-expanding in $G[\Gamma]$. It is known that there is no expander decomposition with $X \le 0.99|E|$ where $\phi = \omega(1/\log n)$ [AALOG18, MS18]. Here, we give an expander *hierarchy* with $\phi = 1/2$.

Theorem 2.3 follows immediately from the lemma below, inspired by [RST14, Lemma 3.1].

Lemma 2.4. *There exists an exponential-time algorithm that, given a graph $G = (V, E)$, computes an edge set X such that every connected component of $G \setminus X$ contains at most $n/2$ vertices and X is $\frac{1}{2}$-expanding. In polynomial time, we can instead guarantee that X is $\Omega(1/\sqrt{\log n})$-expanding.*

Proof. Initialize $X \leftarrow E$. If X is $\frac{1}{2}$-expanding, we are done. Otherwise, we repeatedly update X as follows. Since X is not $\frac{1}{2}$-expanding, there exists a vertex set S where $|S| \le n/2$ such that $|E(S, V \setminus S)| < \frac{1}{2}\min\{\mathsf{Deg}_X(S), \mathsf{Deg}_X(V \setminus S)\}$. Update $X \leftarrow X \cup E(S, V \setminus S) \setminus (X \cap E(S, S))$.

Let X' denote X after the update. Observe that every connected component of $G \setminus X'$ still contains at most $n/2$ vertices because $|S| \le n/2$. Moreover, since

$$\frac{1}{2}(2|X \cap E(S, S)| + |X \cap E(S, V \setminus S)|) = \frac{1}{2}\mathsf{Deg}_X(S) > |E(S, V \setminus S)|,$$

we have

$$|X \cap E(S, S)| > |E(S, V \setminus S)| - \frac{1}{2}|X \cap E(S, V \setminus S)| \ge |E(S, V \setminus S) \setminus X|.$$

[2]The factor 1/2 in Theorem 2.3 is quite artificial. It can be improved to 1 if we slightly change the definition such that X is ϕ-expanding in G if, for every cut $(S, V \setminus S)$, $|E_G(S, V \setminus S)| \ge \phi \min\{|E_G(S, V) \cap X|, |E_G(V \setminus S, V) \cap X|\}$.

Thus, $|X'| < |X|$ and there can be at most $|E|$ iterations of the procedure.

To get a polynomial time construction, we instead apply the sparsest cut algorithm of Arora et al. [ARV09] that, given X, either guarantees that X is $\Omega(1/\sqrt{\log n})$-expanding or returns a set S where $|S| \le n/2$ and $|E(S, V \setminus S)| < \frac{1}{2}\min\{\mathsf{Deg}_X(S), \mathsf{Deg}_X(V \setminus S)\}$. $\square$

Proof of Theorem 2.3. Given G, compute the edge set X from Lemma 2.4 and set $E_h \leftarrow X$. To compute $E_{h-1}, E_{h-2}, \dots, E_1$, we recurse on each connected component C of $G \setminus X$. We have $h \le \log n$ because each component C has size $|C| \le n/2$. $\square$

2.2 A Simple $\tilde{O}(f)$-Bit Labeling Scheme

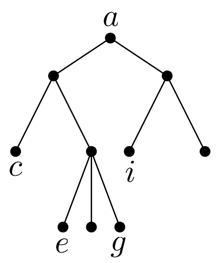
In this section, we prove Theorem 2.5, which uses $\tilde{O}(f)$-bit labels. It is the basis for our final $\tilde{O}(\sqrt{f})$-bit labeling scheme presented in Theorem 2.1.

Theorem 2.5. *Fix any undirected graph $G = (V, E)$ and integer $f \ge 1$. There are deterministic labeling functions $L_V : V \to \{0,1\}^{\log n}$ and $L_E : E \to \{0,1\}^{O(f\phi^{-1}\log^2 n)}$ such that given any query $\langle s, t, F\rangle$, $F \subset E$, $|F| \le f$, one can determine whether s and t are connected in $G - F$ by inspecting only $L_V(s), L_V(t), \{L_E(e) \mid e \in F\}$. The construction time is exponential for $\phi = 1/2$ and polynomial for $\phi = \Omega(1/\sqrt{\log n})$.*

At a very high level, Theorem 2.5 is proved by adapting Pătraşcu and Thorup's [PT07] centralized edge-failure connectivity oracle to the distributed setting. Note that Theorem 2.5 already improves the state-of-the-art polynomial-time computable $\tilde{O}(f^2)$-bit labeling of [IEWM23].

Given a function $\mathsf{level} : E \to [h]$, let $\{E_1, \dots, E_h\}$ be the corresponding edge partition, where $E_\ell = \mathsf{level}^{-1}(\ell)$. Suppose $\{E_1, \dots, E_h\}$ induces an (h, ϕ)-expander hierarchy $\mathcal{H}$. Define T^* to be a minimum spanning tree with respect to level, and let $\mathsf{Euler}(T^*)$ be its Euler tour, which is a list of length $n + 2(n-1)$ that includes each vertex once (its first appearance) and each T^*-edge $\{u, v\}$ twice, as (u, v) and (v, u), according to a DFS traversal of T^*, starting from an arbitrary root vertex. Each vertex u is identified by its position in $\mathsf{Euler}(T^*)$, denoted $\mathsf{DFS}(u)$. We call each edge in $\mathsf{Euler}(T^*)$ an *oriented* edge. See Fig. 2 for a small example.

For each $\ell \le h$, let $\mathcal{T}_{\le \ell}$ be the set of *level-ℓ trees* in the forest $T^* \cap E_{\le \ell}$. Observe that for each $T \in \mathcal{T}_{\le \ell}$, $\mathsf{Euler}(T)$ is a subsequence of $\mathsf{Euler}(T^*)$, not necessarily contiguous. Furthermore, because T^* is a minimum spanning tree with respect to level, each connected component Γ of $G_{\le \ell}$ has a unique level-ℓ tree $T \in \mathcal{T}_{\le \ell}$ such that T spans Γ.



$$\mathsf{Euler}(T^*) = \Big(a, (a,b), b, (b,c), c, (c,b), (b,d), d, (d,e), e, (e,d), (d,f), f, (f,d), (d,g), g, (g,d), (d,b), (b,a), (a,h), h, (h,i), i, (i,h), (h,j), j, (j,h), (h,a)\Big).$$

Figure 2: Left: T^* on vertex set $\{a, b, c, \dots, j\}$, rooted at a. Right: $\mathsf{Euler}(T^*)$.

Definition 2.6 (Simple Deterministic Edge Labels)**.** For $v \in V$, the vertex label $L_V(v)$ is just $\mathsf{DFS}(v)$. Each *non-tree* edge $e = \{u, v\} \notin E(T^*)$ has a $2\log n$-bit label $L_E(e) = (\mathsf{DFS}(u), \mathsf{DFS}(v))$. Each *tree edge* $e = \{u, v\} \in E(T^*)$ is assigned an $O((f/\phi)\log^2 n)$-bit label, generated as follows.

1. Store $(L_V(u), L_V(v), \mathsf{level}(e))$.

2. For each $\ell \in [\mathsf{level}(e), h]$, let $T_\ell \in \mathcal{T}_{\leq \ell}$ be the tree containing e. Write $\mathsf{Euler}(T_\ell)$ as

$$\mathsf{Euler}(T_\ell) = X \cdot (u,v) \cdot Y \cdot (v,u) \cdot Z.$$

   For each $W \in \{X, Y, Z\}$,

   (a) Store the labels of the first and last elements of W, and store the labels of the first and last vertices in W (i.e. $\min_{u \in W} \mathsf{DFS}(u)$ and $\max_{v \in W} \mathsf{DFS}(v)$).

   (b) Store the first $f/\phi + 1$ level-ℓ non-tree edges incident to vertices in W. (Each such edge $\{u,v\}$ is encoded by $L_E(\{u,v\}) = (\mathsf{DFS}(u), \mathsf{DFS}(v))$.)

The Query Algorithm. For each level $1 \leq \ell \leq h$, let $\mathcal{T}_{\leq \ell, F} \subseteq \mathcal{T}_{\leq \ell}$ collect all level-ℓ trees T such that T intersects F. Recall that each connected component Γ of G_ℓ has a unique $T \in \mathcal{T}_{\leq \ell}$ as its spanning tree. We define $G_{\leq \ell, F}$ to be a subgraph of G_ℓ that only collects the connected components of $G_{\leq \ell}$ whose spanning tree is in $\mathcal{T}_{\leq \ell, F}$.

Our goal is to sequentially build vertex partitions $\mathcal{P}_1, \ldots, \mathcal{P}_h$, where $\mathcal{P}_\ell$ is a partition of $V(G_{\leq \ell, F})$ that reflects the connected components of $G_{\leq \ell, F} - F$. In fact, we will compute, for each $T \in \mathcal{T}_{\leq \ell, F}$, a partition $\mathcal{P}_\ell[T]$ of $V(\Gamma)$ that reflects the connected component of $\Gamma - F$ (where Γ is the connected component of $G_{\leq \ell}$ that has T as its spanning tree), and then $\mathcal{P}_\ell$ is simply the union of $\mathcal{P}_\ell[T]$ over all $T \in \mathcal{T}_{\leq \ell, F}$. After $\mathcal{P}_h$ is computed, we can count the number of connected components in $G - F$, or answer s-t connectivity queries, given $L_V(s), L_V(t)$.

Each $\mathcal{P}_\ell[T]$ has a compact representation as follows. We start with defining *intervals*. Consider a $T \in \mathcal{T}_{\leq \ell}$. We can detect whether T intersects F and if so, enumerate $T \cap F = \{e_1, \ldots, e_{f_0}\}$ using Item 2a of the F labels. If we remove the oriented copies of $T \cap F$, $\mathsf{Euler}(T)$ breaks into a set of $2f_0 + 1$ intervals $\mathcal{J}(T, F)$. Note that from Item 2a, for each interval in $\mathcal{J}(T, F)$, we can obtain the labels of its first and last elements and its first and last vertices. Towards the compact representation of $\mathcal{P}_\ell[T]$, we can think of each $J \in \mathcal{J}(T, F)$ as the vertex set $J \cap V(T)$. Each part $P \in \mathcal{P}_\ell[T]$ is represented by a set of intervals $\mathcal{J}_P \subseteq \mathcal{J}(T, F)$ such that $P = \bigcup_{J \in \mathcal{J}_P} J \cap V(T)$.

For the purposes of *point location*, we will write $J \in \mathcal{J}(T, F)$ as $[\min_{u \in J \cap T} \mathsf{DFS}(u), \max_{v \in J \cap T} \mathsf{DFS}(v)]$. Note that in general, an interval $[\mathsf{DFS}(u), \mathsf{DFS}(v)]$ in $\mathcal{J}(T, F)$ contains (the DFS numbers of) vertices outside of T. Nonetheless, for vertices in $V(T)$, these intervals allow us to do correct point location.

Observation 2.7 (Point Location). *Suppose $x \in V(T)$, $T \in \mathcal{T}_{\leq \ell, F}$. If $[\mathsf{DFS}(u), \mathsf{DFS}(v)]$ is the (unique) interval in $\mathcal{J}(T, F)$ containing $\mathsf{DFS}(x)$ then there is a component in $T - F$ containing $u, x,$ and v.*

The query algorithm *only* does point location on vertices x known to be in $V(T)$, so Observation 2.7 suffices for correctness.

Suppose $\mathcal{P}_{\ell-1}$ has been computed. For each $T \in \mathcal{T}_{\leq \ell}$ intersecting F, we enumerate $T \cap F = \{e_1, \ldots, e_{f_0}\}$ and initialize $\mathcal{P}_\ell[T] \leftarrow \mathcal{J}(T, F)$ (i.e. each part $P \in \mathcal{P}_\ell[T]$ is only one interval), then proceed to unify parts of $\mathcal{P}_\ell[T]$ by applying rules **R1**–**R4** and the operation $\mathsf{Unite}_T(x, y)$. The x, y in $\mathsf{Unite}_T(x, y)$ can be vertices in $V(T)$, intervals in $\mathcal{J}(T, F)$ or even parts in $\mathcal{P}_\ell[T]$, and $\mathsf{Unite}_T(x, y)$ will unite the parts containing x and y in $\mathcal{P}_\ell[T]$.

R1. If J, J' are two intervals of $\mathsf{Euler}(T)$ that share a common endpoint, say J ends with 'u' and J' begins with '(u,v)', call $\mathsf{Unite}_T(J, J')$.

R2. For each call to $\mathsf{Unite}_{T'}(x, y)$ made in the construction of $\mathcal{P}_{\ell-1}[T']$, $T' \subset T$, call $\mathsf{Unite}_T(x, y)$.

R3. For each non-tree edge $\{u, v\} \in E_\ell$ encoded in the labels $\{L_E(e_i) \mid i \in [f_0]\}$, if $\{u, v\} \notin F$, call $\mathsf{Unite}_T(u, v)$.

Rule **R1** is implemented with Item 2a of the F-labels. The enumeration of edges in **R3** uses Item 2b of the F-labels, but to implement $\mathsf{Unite}_T(u, v)$ we need to locate the intervals in $\mathcal{J}(T, F)$ containing u, v. Since $\mathsf{level}(\{u, v\}) = \ell$, both u, v are in $V(T)$, and by Observation 2.7 we can locate the intervals containing u, v, given $\mathsf{DFS}(u), \mathsf{DFS}(v)$.

According to **R2**, every $\mathsf{Unite}_{T'}(x, y)$ performed at level $\ell - 1$ on some $T' \subset T$ is re-executed verbatim as $\mathsf{Unite}_T(x, y)$ if x, y are vertices. Since $x, y \in V(T') \subseteq V(T)$, Observation 2.7 lets us identify the intervals in $\mathcal{J}(T, F)$ containing x, y. If x, y were intervals from $\mathcal{J}(T', F)$ we can pick the first vertices from x and y, say x', y', and call $\mathsf{Unite}_T(x', y')$. Once again, $x', y' \in V(T') \subseteq V(T)$, so Observation 2.7 applies.

After executing **R1**, $\mathcal{P}_\ell[T]$ reflects the connected components of $T - F$. After executing **R2**, $\mathcal{P}_\ell[T]$ reflects the connected components of $(G_{\leq \ell-1}[V(T)] \cup T) - F$. If it were the case that **R3** had access to *all* level-ℓ non-tree edges, then it would be sufficient to find all connected components of $G_{\leq \ell}[V(T)] - F$. However, Item 2b of the F-labels are only guaranteed to reveal up to $f/\phi + 1$ level-ℓ non-tree edges per interval in $\mathcal{J}(T, F)$.

Lemma 2.8. *If $P_1, \ldots, P_k \in \mathcal{P}_\ell[T]$ are those parts with $\mathsf{Deg}_\ell(P_i) > f/\phi$, then $\bigcup_{i=1}^k P_i$ are contained in a single connected component of $G_{\leq \ell} - F$.*

Proof. Suppose the claim is false, that there is some partition of $\{P_1, \ldots, P_k\}$ into A and B which are disconnected. Then
$$\frac{|E_{\leq \ell}(A, B)|}{\min\{\mathsf{Deg}_\ell(A), \mathsf{Deg}_\ell(B)\}} < \frac{f}{f/\phi} = \phi,$$
contradicting the fact that $\mathcal{H}$ is an (h, ϕ)-expander hierarchy. □

In light of Lemma 2.8, we continue to unify parts according to a fourth rule.

R4. If $P, P' \in \mathcal{P}_\ell[T]$ have $\mathsf{Deg}_\ell(P), \mathsf{Deg}_\ell(P') > f/\phi$, call $\mathsf{Unite}_T(P, P')$.

The *full* partition $\mathcal{P}_\ell$ is obtained by taking the union of all $\mathcal{P}_\ell[T]$, for $T \in \mathcal{T}_{\leq \ell}$ intersecting F, plus the trivial partitions $\mathcal{P}_\ell[T] = \{V(T)\}$ for every $T \in \mathcal{T}_{\leq \ell}$ disjoint from F.

Lemma 2.9 (Correctness). *If $P \in \mathcal{P}_\ell$, then P is a connected component in $G_{\leq \ell} - F$.*

Proof. Rules **R1**–**R3** are clearly sound and Lemma 2.8 implies **R4** is sound. We consider completeness. If there is a path between $u, v \in V(T)$ in $G_{\leq \ell-1} - F$ then by induction on ℓ, u and v will be in the same part of $\mathcal{P}_\ell$ after executing **R1** and **R2**. Suppose u, v are joined by a path in $G_{\leq \ell} - F$, but u, v are in different parts P_u, P_v connected by a level-ℓ edge $e' = \{u', v'\}$. Because **R3** could not be applied, e' is not contained in Item 2b of the F-edges bounding the intervals in $\mathcal{J}(T, F)$ containing u', v', implying $\mathsf{Deg}_\ell(P_u), \mathsf{Deg}_\ell(P_v) > f/\phi$, but then by **R4**, P_u, P_v would have been united. □

Once we have constructed $\mathcal{P}_h$, a connectivity query $\langle s,t,F\rangle$ works as follows. First, identify the two intervals in $\mathsf{Euler}(T^*)$ of the forest T^*-F that contains s and t. This can be done using the predecessor search over the endpoints of the intervals in $O(\min\{\frac{\log\log n}{\log\log\log n}, \frac{\log f}{\log\log n}\})$ time [PT06, PT14], since there are $O(f)$ intervals, each represented by $O(\log n)$-bit numbers. Then we check whether the two intervals are in the same part of $\mathcal{P}_h$, corresponding to the same connected component of $G-F$.

2.3 Second Tool: Code Shares

Theorem 2.10 gives the distributed coding scheme. We will only invoke it with $d=2$.

Theorem 2.10 (Reed-Solomon Code Shares)**.** *Let* $\mathbf{m}\in\mathbb{F}_q^k$ *be a message, with* $q>k$*. For any integer parameter* $d\geq 2$*, there are* $O(d\log q)$*-bit* code shares $C_1,\ldots,C_k$ *so that for any index set* $J\subset[k]$ *with* $|J|\geq k/d$*, we can reconstruct* $\mathbf{m}$ *from the code shares* $\{C_j \mid j\in J\}$ *in polynomial time.*

Proof. One can regard $\mathbf{m}$ as the coefficients of a degree-$(k-1)$ polynomial g_1 over $\mathbb{F}_q$, or even as a degree-$(\lceil k/d\rceil-1)$ polynomial g_d over $\mathbb{F}_{q^d}$. The code shares $(C_i)_{1\leq i\leq k}$ are defined to be distinct evaluations of g_d.

$$C_i=(i,g_d(i)).$$

Given the code shares $\{C_i \mid i\in J\}$ for $|J|\geq k/d$, we can reconstruct g_d and hence $\mathbf{m}$ in polynomial time via polynomial interpolation. □

2.4 An $\tilde{O}(\sqrt{f})$-Bit Labeling Scheme

High-level Idea. The labeling scheme of Section 2.2 is *non-constructive* inasmuch as rule **R4** infers that two P,P' are connected, not by finding a path between them, but by checking if their volumes $\mathsf{Deg}_\ell(P),\mathsf{Deg}_\ell(P')>f/\phi$. In this section, we give a labeling scheme that is *even more* non-constructive. We can sometimes infer that a part P with $\mathsf{Deg}_\ell(P)<f/\phi$ is nonetheless in a connected component C of $G_{\leq\ell}-F$ with $\mathsf{Deg}_\ell(C)>f/\phi$ without explicitly knowing an edge incident to P.

A key idea in the construction is to store a large volume of information about an interval of an Euler tour as *code shares* distributed across labels of "nearby" edges. Given a sufficient number of code shares, we will be able to reconstruct the information about the interval.

Notations. We shall assume without loss of generality that the graph has degree 3. Given any $G'=(V',E')$ with irregular degrees, we form $G=(V,E)$ by substituting for each $v'\in V$ a $\mathsf{Deg}(v')$-cycle, then attach each edge incident to v' to a distinct vertex in the cycle, hence $|V|=|E'|/2$. Given labelings $L_V:V\to\{0,1\}^*, L_E:E\to\{0,1\}^*$, we let $L_{E'}(e')=L_E(\phi(e')), L_{V'}(v')=L_V(\phi(v'))$, where $\phi(e')\in E$ is the edge corresponding to e' and $\phi(v')\in V$ is any vertex in the $\mathsf{Deg}(v')$-cycle of v'. Correctness is immediate, since $F'\subset E'$ disconnects $s,t\in V'$ iff $\phi(F')\subset E$ disconnects $\phi(s),\phi(t)\in V$.

Recall T^* is the minimum spanning tree of G with respect to the level function. Fix a level ℓ. Let $T\in\mathcal{T}_{\leq\ell}$ be a tree spanning a component of $G_{\leq\ell}$. For each $v\in V(T)$, let

$$\mathsf{wt}_\ell(v)=\begin{cases}1 & \text{if } v \text{ is incident to a level-}\ell\text{ non-tree edge,}\\ 0 & \text{otherwise,}\end{cases}$$

and for each oriented tree edge (u,v), let $\mathsf{wt}_\ell(u,v)=0$. If S is an interval of vertices and oriented edges in $\mathsf{Euler}(T)$, $\mathsf{wt}_\ell(S)=\sum_{x\in S}\mathsf{wt}_\ell(x)$.

Let $[\alpha,\beta]$ denote the interval of $\mathsf{Euler}(T)$ starting at α and ending at β. Then $\mathsf{dist}_\ell(\alpha,\beta)=\mathsf{dist}_\ell(\beta,\alpha)\overset{\text{def}}{=}\sum_{\gamma\in[\alpha,\beta]\setminus\{\alpha,\beta\}}\mathsf{wt}_\ell(\gamma)$.[3] For any vertex/edge element α in $\mathsf{Euler}(T)$, the set of all vertices within distance r is:

$$\mathsf{Ball}_\ell(\alpha,r)=\{v\in V(T)\mid \mathsf{dist}_\ell(\alpha,v)\le r\}.$$

We overload the Ball-notation for edges $e=\{u,v\}$. Here $(u,v),(v,u)$ refer to the oriented occurrences of e in an Euler tour, if e is a tree edge.

$$\mathsf{Ball}_\ell(e,r)=\begin{cases}\mathsf{Ball}_\ell((u,v),r)\cup\mathsf{Ball}_\ell((v,u),r) & \text{when } e\in E(T) \text{ is a tree edge,}\\ \mathsf{Ball}_\ell(u,r)\cup\mathsf{Ball}_\ell(v,r) & \text{when } e\notin E(T) \text{ is a non-tree edge.}\end{cases}$$

Henceforth, the only balls we consider have radius r,

$$r\overset{\text{def}}{=}\lceil\sqrt{f/\phi}\rceil.$$

Assume, without loss of generality, that $\mathsf{wt}(\mathsf{Euler}(T))$ is a power of 2, by padding the end with dummy weight-1 elements if necessary. For every $j\le j_{\max}\overset{\text{def}}{=}\lceil\log(f/\phi)\rceil$, define $\mathcal{I}_j$ to be a partition of $\mathsf{Euler}(T)$ into consecutive intervals with weight 2^j, each the union of two intervals from $\mathcal{I}_{j-1}$.

The Scheme. The key idea of this labeling scheme is to focus on *large gap edges*. See Fig. 3.

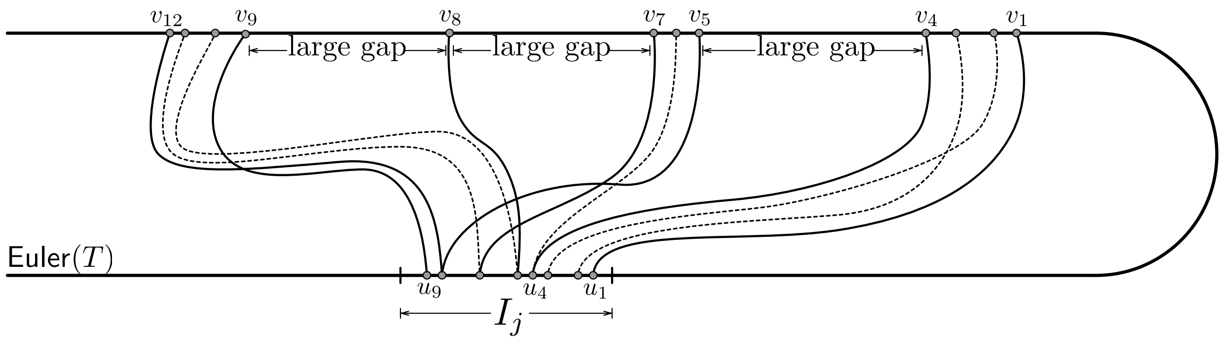


Figure 3: An interval $I_j\in\mathcal{I}_j$ with $\mathsf{wt}(I_j)=2^j$ $(j=3)$. There are 12 level-ℓ edges in $E(I_j,\overline{I_j})$, $\{u_1,v_1\},\ldots,\{u_{12},v_{12}\}$, ordered by their non-$I_j$ endpoint. The large gap edges of I_j are $\{u_1,v_1\},\{u_4,v_4\},\{u_5,v_5\},\{u_7,v_7\},\{u_8,v_8\},\{u_9,v_9\},\{u_{12},v_{12}\}$.

Definition 2.11 (Large Gap Edges)**.** Fix an interval $I_j\in\mathcal{I}_j$ and let $E_\ell(I_j):=E_\ell(I_j,\overline{I_j})$ be all level-ℓ edges with exactly one endpoint in I_j. We write $E_\ell(I_j)=\{\{u_1,v_1\},\{u_2,v_2\},\ldots\}$ such that, for all i, $u_i\in I_j$ and $v_i\in\overline{I_j}$. Order the edges according to v_i, so

$$\mathsf{DFS}(v_1)\le\mathsf{DFS}(v_2)\le\cdots\le\mathsf{DFS}(v_{|E_\ell(I_j)|}).\qquad(1)$$

[3]We exclude the endpoints of the intervals from the sum just to avoid double counting at the endpoints when we sum distances of two adjacent intervals.

If, for $q \in [1, |E_\ell(I_j)|)$, $v_{q+1} \notin \mathsf{Ball}_\ell(v_q, r)$, then $\{u_q, v_q\}, \{u_{q+1}, v_{q+1}\}$ are called ***large gap edges*** w.r.t. ℓ and I_j. The first and last edges $\{u_1, v_1\}, \{u_{|E_\ell(I_j)|}, v_{|E_\ell(I_j)|}\}$ are always large gap edges. Define $\mathsf{LGE}_\ell(I_j) \subseteq E_\ell(I_j)$ to be the set of large gap edges and $\mathsf{lge}_\ell(I_j) = |\mathsf{LGE}_\ell(I_j)|$ to be their number.

Regard $\mathsf{LGE}_\ell(I_j)$ as a *message* $\mathbf{m}_\ell(I_j) \in \mathbb{F}_q^{\mathsf{lge}_\ell(I_j)}$, where $q = \mathrm{poly}(n)$ is large enough to encode a single edge. We apply Theorem 2.10 with $d = 2$ to break $\mathbf{m}_\ell(I_j)$ into code shares so that given any set of $\mathsf{lge}_\ell(I_j)/2$ shares we can reconstruct $\mathbf{m}_\ell(I_j)$. If $\{u, v\} \in \mathsf{LGE}_\ell(I_j)$ is a large gap edge with $u \in I_j$ and $v \in \overline{I_j}$, let $C_{\ell,j}(v, u)$ be the code share of $\{u, v\}$ w.r.t. ℓ and I_j. Set $C_{\ell,j}(v, u) \overset{\text{def}}{=} \perp$ if $\{u, v\} \notin \mathsf{LGE}_\ell(I_j)$ is not a large gap edge. (Note that $C_{\ell,j}(u, v)$ would be a different code share w.r.t. some interval $I'_j \ni v$.) To simplify notation, for each edge $e = \{u, v\}$, we define the *level-ℓ code share of* e as a bundle $C_\ell(e) = \{C_{\ell,j}(u, v), C_{\ell,j}(v, u) \mid j \le j_{\max}\}$. Note that $C_\ell(e)$ also indicates whether e is a large gap edge w.r.t. all intervals $I_j \ni u$ and $I'_j \ni v$, for all $j \le j_{\max}$.

Definition 2.12 (Shorter Deterministic Edge Labels)**.** An $O\left(\sqrt{f/\phi}\log(f/\phi)\log^2 n\right)$-bit label $L_E(e)$ for each edge $e = \{u, v\}$ is constructed as follows.

1. Store $(L_V(u), L_V(v), \mathsf{level}(e))$.
2. For each $\ell \in [\mathsf{level}(e), h]$ and all level-ℓ edges $e' = \{u', v'\}$ incident to $\mathsf{Ball}_\ell(e, r)$ vertices,
   (a) store $(\mathsf{DFS}(u'), \mathsf{DFS}(v'))$,
   (b) store the level-ℓ code share bundle $C_\ell(e')$.
3. If $e \in T^*$, then for each $\ell \in [\mathsf{level}(e), h]$, let $T_\ell \in \mathcal{T}_{\le \ell}$ be the tree containing e. Write $\mathsf{Euler}(T_\ell)$ as
$$\mathsf{Euler}(T_\ell) = X \cdot (u, v) \cdot Y \cdot (v, u) \cdot Z.$$
   (a) For each $W \in \{X, Y, Z\}$, store the labels of the first and last elements of W, and store the labels of the first and last vertices in W (i.e. $\min_{u \in W} \mathsf{DFS}(u)$ and $\max_{v \in W} \mathsf{DFS}(v)$).
   (b) For each $j \le j_{\max}$, let $I_j^{(1)}, I_j^{(2)}, I_j^{(3)}, I_j^{(4)} \in \mathcal{I}_j$ be the closest intervals on either side of $(u, v), (v, u)$ that do not contain (u, v) or (v, u). For each $k \in \{1, \ldots, 4\}$, store $\mathsf{lge}_\ell(I_j^{(k)})$, and if $\mathsf{lge}_\ell(I_j^{(k)}) \le 4r$, store $\mathsf{LGE}_\ell(I_j^{(k)})$.

The bit-length of the edge labeling is justified as follows. Item 1 has length $O(\log n)$. Since $C_\ell(e)$ uses $O(j_{\max}\log n)$ bits by Theorem 2.10, we have that Item 2 has length $O(hrj_{\max}\log n) = O(\sqrt{f/\phi}\log(f/\phi)\log^2 n)$. Item 3 has length $h \cdot (O(\log n) + rj_{\max}O(\log n)) = O(\sqrt{f/\phi}\log(f/\phi)\log^2 n)$ bits.

The Query Algorithm. We initialize $\mathcal{P}_\ell[T]$, $T \in \mathcal{T}_{\le \ell, F}$, exactly as in the proof of Theorem 2.5, and proceed to apply rules **R1**–**R3** as-is. Note that we can implement **R1** using Item 3a and **R3** using Item 2a. **R2** is simply re-executing calls to $\mathsf{Unite}_{T'}$ from those $T' \in \mathcal{T}_{\le \ell-1, F}$ such that $T' \subset T$. We replace rule **R4** with the similar rule **R4'**.

R4'. Suppose that $\mathcal{Q} \subseteq \mathcal{P}_\ell[T]$ is such that for each $P \in \mathcal{Q}$, P must be in a connected component C of $G_\ell - F$ with $\mathsf{Deg}_\ell(C) > f/\phi$. Then, unite all parts of $\mathcal{Q}$ with calls to Unite_T.

To prove correctness, we recall the definition of $\mathcal{J} = \mathcal{J}(T, F)$. Let $F \cap E(T) = \{e_1, \ldots, e_{f_0}\}$ be the set of deleted tree edges of T, which we can enumerate using Item 3a of the edge labels. These edges break $\mathsf{Euler}(T)$ into a set of $2f_0 + 1$ intervals denoted by $\mathcal{J}(T, F)$. Each part $P \in \mathcal{P}_\ell[T]$ consists of a collection of intervals from $\mathcal{J}(T, F)$. Our goal is to prove the following.

Lemma 2.13. *For each interval $J \in \mathcal{J}$, we can either (i) list all other intervals $J' \in \mathcal{J}$ adjacent to J, or (ii) infer that J is in a connected component C of $G_{\leq \ell} - F$ with $\mathsf{Deg}_\ell(C) > f/\phi$.*

The above lemma implies correctness of the algorithm as we can keep applying **R3** and **R4'** to obtain the correct $\mathcal{P}_\ell[T]$ at the end.

Consider an interval $J \in \mathcal{J}$. Let the F-edges bounding J be α_0, α_1. Observe that we can partition J into less than $2(j_{\max} + 1)$ intervals from $\mathcal{I}_0 \cup \cdots \cup \mathcal{I}_{j_{\max}}$. We consider each of these intervals $I_j \in \mathcal{I}_j$ individually. Below, we say that F *reveals* e' if e' is incident to a vertex in $\mathsf{Ball}_\ell(e, r)$, for some failed edge $e \in F$. Whenever F reveals e', Item 2 of the edge labels gives us the position of its endpoints and the code share bundle for e'. The following lemma is crucial.

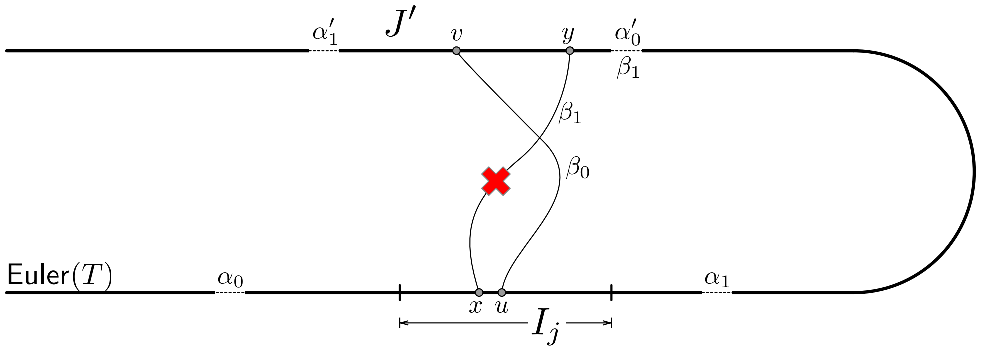


Figure 4: Illustration of Lemma 2.14. An interval $I_j \subseteq J$ is incident to J'. $J, J' \in \mathcal{J}$ are bounded by $\alpha_0, \alpha_1 \in F$ and $\alpha_0', \alpha_1' \in F$, respectively. Either β_0 is a large gap edge, and stored in either $L_E(\alpha_0)$ or $L_E(\alpha_1)$, or it is stored in $L_E(\beta_1)$, where $\beta_1 = \{x, y\} \in F$ (if it exists), or $\beta_1 = \alpha_0'$.

Lemma 2.14. *Consider an interval $I_j \subseteq J \in \mathcal{J}$ where $I_j \in \mathcal{I}_j$. If we have access to the set $\mathsf{LGE}_\ell(I_j)$ of large gap edges, then we can check if another interval $J' \in \mathcal{J}$ is adjacent to I_j.*

Proof. Suppose that $J' \in \mathcal{J}$ is adjacent to I_j. Let $\beta_0 = \{u, v\} \in E - F$ be the *first* level-ℓ non-deleted edge with $u \in I_j, v \in J'$, when ordered by DFS number. We claim that either $\beta_0 \in \mathsf{LGE}_\ell(I_j)$ is a large gap edge or β_0 is revealed by F. In either case, we learn the endpoints of β_0, which certifies that J' is adjacent to I_j.

Let J' be bounded by F-edges α_0', α_1'. Consider a level-ℓ edge $\{x, y\}$ that is the predecessor of β_0 according to Equation (1). In particular, $x \in I_j$ and $y \notin I_j$ has the largest $\mathsf{DFS}(y)$ such that $\mathsf{DFS}(y) \leq \mathsf{DFS}(v)$. Suppose $\beta_0 \notin \mathsf{LGE}_\ell(I_j)$. Then, $v \in \mathsf{Ball}_\ell(y, r)$ by definition. Now, we claim there is a $\beta_1 \in F$ where $v \in \mathsf{Ball}_\ell(\beta_1, r)$. Hence, β_0 is revealed by F which would complete the proof. There are just two cases.

- If $y \in J'$, then $\{x, y\} \in F$ since β_0 is the first non-deleted edge. We choose $\beta_1 = \{x, y\} \in F$ and therefore $v \in \mathsf{Ball}_\ell(\beta_1, r)$.
- If $y \notin J'$, then $\mathsf{dist}_\ell(v, \alpha_0') \leq \mathsf{dist}_\ell(v, y) \leq r$ and therefore $v \in \mathsf{Ball}_\ell(\alpha_0', r)$. We set $\beta_1 = \alpha_0' \in F$ so $v \in \mathsf{Ball}_\ell(\beta_1, r)$. □

We are now ready to prove Lemma 2.13.

Proof of Lemma 2.13. There are three cases.

Case 1: An $I_j \subseteq J$ has $\mathsf{lge}_\ell(I_j) \leq 4r$. By Item 3b of the label $L_E(\alpha_0)$ or $L_E(\alpha_1)$, we can access the whole set $\mathsf{LGE}_\ell(I_j)$. So, by Lemma 2.14, we can list all intervals $J' \in \mathcal{J}$ adjacent to I_j.

Case 2: An $I_j \subseteq J$ has $\mathsf{lge}_\ell(I_j) > 4r$, and F reveals at least $(\mathsf{lge}_\ell(I_j))/2$ large gap edges in $\mathsf{LGE}_\ell(I_j)$. Given $(\mathsf{lge}_\ell(I_j))/2$ code shares, by Theorem 2.10, we can also reconstruct $\mathsf{LGE}_\ell(I_j)$. So, we can again list all intervals $J' \in \mathcal{J}$ adjacent to I_j, by Lemma 2.14.

Case 3: An $I_j \subseteq J$ has $\mathsf{lge}_\ell(I_j) > 4r$, and F reveals at most $(\mathsf{lge}_\ell(I_j))/2$ large gap edges in $\mathsf{LGE}_\ell(I_j)$. In this case, we claim that the connected component C in $G_{\leq \ell} - F$ containing I_j has $\mathsf{Deg}_\ell(C) > f/\phi$. Observe that for each *unrevealed* large gap edge $\{u, v\} \in \mathsf{LGE}_\ell(I_j)$ with $u \in I_j$, it must be that $\mathsf{Ball}_\ell(v, r) \subseteq C$ because there is no failed edge $e \in F$ within distance r from v. Each $\mathsf{Ball}_\ell(v, r)$ has weight at least $2r$, and the sum of their weights can be at most four times the weight of their union. So $\mathsf{Deg}_\ell(C) > (2r/4) \cdot (\mathsf{lge}_\ell(I_j)/2) > (2r/4) \cdot (4r/2) = r^2 \geq f/\phi$ as desired. □

3 Deterministic Vertex Fault Connectivity Labels

This section is dedicated to proving Theorem 3.1 concerning labels for *vertex* faults.

Theorem 3.1 (Improved Deterministic Vertex Labels)**.** *Fix any undirected graph $G = (V, E)$ with n vertices and integer $f \geq 1$. There are deterministic labeling functions $L_V : V \to \{0,1\}^{O(f^4\phi^{-1}\log^7 n)}$ such that given any query $\langle s, t, F\rangle$, $F \subset V$, $|F| \leq f$, one can determine whether s and t are connected in $G - F$ by inspecting only $L_V(s), L_V(t), \{L_V(v) \mid v \in F\}$. The construction time is exponential when $\phi = 1$ and polynomial when $\phi = \Omega(1/\sqrt{\log n})$. The query time is* $\mathrm{poly}(f, \log n)$*.*

To prove Theorem 3.1, we first describe the underlying hierarchical structure of the algorithm in Section 3.2. This structure allows us to prove a divide-and-conquer lemma (Lemma 3.14) that is crucial for answering connectivity queries under vertex failures in a bottom-up manner on the hierarchy. Based on the divide-and-conquer lemma, in Section 3.3 we then describe how to answer connectivity queries assuming access to primitives that return information about the hierarchy. Finally, we describe how to implement these primitives in a distributed manner by providing the labeling scheme in Section 3.4.

3.1 Overview and Challenges

We briefly discuss our approach at a very high level. We would like to highlight the specific challenges that arise when tolerating vertex faults relative to edge faults.

The First Challenge. We start with the ideal scenario that the input graph is already a vertex expander. The first challenge is how to obtain a *stable connectivity certificate*. Recall that in the edge fault scenario (also assuming the graph is an edge expander), we can simply take a spanning tree as a stable connectivity certificate. The removal of any f edges will only break the spanning tree into $f+1$ subtrees. However, in general there is no upper bound on the number of subtrees when removing f vertices.

This is a natural barrier to handling vertex faults, and several previous works, e.g. [DP20, LS22, PPP24], follow the same idea to overcome it. Take a *low-degree* spanning tree as a stable connectivity certificate. A vertex expander indeed admits a low-degree spanning tree; this is formally stated in Lemma 3.6.

Using this idea, one can easily generalize our edge fault connectivity labeling scheme to obtain an $\widetilde{O}(f)$-size vertex fault connectivity labeling scheme for vertex expander input graphs. Roughly speaking, let F be the vertex faults. From the low-degree spanning tree T, we first obtain an initial partition $\mathcal{P}$ of $G \setminus F$ consisting of the connected components of $T \setminus F$. Then we exploit the nature of a ϕ-vertex expander G: all sets $A \in \mathcal{P}$ s.t. $|A \cup N_G(A)| > f/\phi$—call them *giant sets*—must be inside the same connected component of $G \setminus F$. On the other hand, for each non-giant set $A \in \mathcal{P}$, its neighbor set $N_G(A)$ is of size at most f/ϕ and we can obtain $N_G(A)$ explicitly by designing labels on the Euler tour of the low-degree spanning tree. Therefore, we first merge all the giant sets of $\mathcal{P}$ together, and then merge further using $N_G(A)$ for each non-giant $A \in \mathcal{P}$.

The Second Challenge. The second challenge arises when the input graph is *not* a vertex expander. In fact, an input graph G admitting a two-level vertex expander hierarchy already captures this challenge. A graph G admits a two-level ϕ-vertex expander hierarchy if there is a separator $X \subseteq V(G)$ s.t. X is ϕ-expanding in G, and each connected component C of $G \setminus X$ is a ϕ-vertex expander.

Let F be the vertex faults. We assume that we are given an initial partition $\mathcal{P}$ of $V(G) \setminus F$ that captures (1) the connectivity of $C \setminus F$ for each connected component C of $G \setminus X$, and (2) the connectivity of X in $T \setminus F$ certified by the stable connectivity certificate T. For example, we can think of

$$\mathcal{P} = \mathcal{P}_X \cup \bigcup_{\text{components } C \text{ of } G \setminus X} \{\text{components of } C \setminus F\},$$

where $\mathcal{P}_X$ is some partition of $X \setminus F$ which is with respect to the connectivity of X in $G \setminus F$, but may not fully capture the connectivity of X in $G \setminus F$.

Clearly, we are done if we can further merge sets in $\mathcal{P}$ using edges incident to X, call them *X-edges*. Let us try the same algorithm and see how it breaks. The fact that X is ϕ-expanding in G tells us that all sets $A \in \mathcal{P}$ s.t. $|(A \cup N_G(A)) \cap X| > f/\phi$ (giant sets) must belong to the same connected component of $G \setminus F$. Also, for each non-giant set $A \in \mathcal{P}$, $N_G(A) \cap X$ is of size at most f/ϕ and we can obtain it explicitly with $\widetilde{O}(f)$-bit labels. Again, we first merge giant sets into one *giant group*, and then for each $A \in \mathcal{P}$, merge it with each group intersecting $N_G(A) \cap X$. Let us try to confirm the correctness of this merging procedure. We use $\{x, v\}$ to denote an X-edge with $x \in X$.

1. Any two giant sets $A_1, A_2 \in \mathcal{P}$ are indeed merged.
2. Suppose an X-edge $\{x, v\}$ joins $A_1, A_2 \in \mathcal{P}$ with $x \in A_1, v \in A_2$, where A_2 is non-giant. Then we will merge A_1 and A_2 because A_1 intersects $N_G(A_2) \cap X$.
3. Suppose an X-edge $\{x, v\}$ joins $A_1, A_2 \in \mathcal{P}$ with $x \in A_1, v \in A_2$, but now A_1 is non-giant and A_2 is giant. Although A_1 intersects $N_G(A_2) \cap X$, we are $\underline{\textit{not}}$ guaranteed that A_1 and A_2 are merged because A_2 is $\underline{\textit{giant}}$. In other words, X-edges in case 3 will not be detected by this method.

This *asymmetry* is the major difference between the vertex-fault case and the edge-fault case. To overcome it, the key observation is that if there exists a case-3 X-edge for a non-giant set A_1, i.e. there exists an X-edge $\{x, v\}$ such that $x \in A_1$ and v is in some giant set, then A_1 must be merged with the giant group. In other words, for a non-giant A_1, instead of knowing all case-3 X-edges incident to A_1, it suffices to check if any such case-3 X-edges *exist*.

Therefore, we will *count* the number of case-3 X-edges for each non-giant set A_1. Roughly speaking, this is possible because this number is exactly $\delta_{\text{all}} - \delta_{\text{non-giant}} - \delta_F$, where

$$\begin{aligned}
\delta_{\text{all}} &= \text{number of } X\text{-edges } \{x, v\} \text{ s.t. } x \in A_1, \\
\delta_{\text{non-giant}} &= \text{number of } X\text{-edges } \{x, v\} \text{ s.t. } x \in A_1 \text{ and } v \text{ is in some non-giant } A_2, \\
\delta_F &= \text{number of } X\text{-edges } \{x, v\} \text{ s.t. } x \in A_1 \text{ and } v \in F.
\end{aligned}$$

We will not elaborate now on how to count these numbers.

3.2 The Structure

3.2.1 The Basis: A Vertex Expander Hierarchy

In this section, we construct an expander hierarchy for vertex expansion similar to the edge expansion version from Section 2.1. This will be the basis of our structure.

For any graph $G = (V, E)$, a vertex cut (L, S, R) is a partition of V such that $L, R \neq \emptyset$ and there is no edge between L and R. For any vertex set $X \subseteq V$, we say that X is *ϕ-vertex-expanding* in G if for every vertex cut (L, S, R) in G,

$$|S| \geq \phi \min\{|X \cap (L \cup S)|,\ |X \cap (R \cup S)|\}.$$

Consider a partition $\{V_1, \ldots, V_h\}$ of V. We denote $V_{\leq \ell} := \bigcup_{i \leq \ell} V_i$ and $V_{>\ell} := \bigcup_{i > \ell} V_i$. Let $G_{\leq \ell}$ be the graph induced by $V_{\leq \ell}$.

Definition 3.2 (Vertex Expander Hierarchy)**.** Given a graph $G = (V, E)$, a vertex-partition $\mathcal{P} = \{V_1, \ldots, V_h\}$ of V induces an (h, ϕ)*-vertex-expander hierarchy* if, for every level $\ell \leq h$ and every connected component Γ in $G_{\leq \ell}$, $V_\ell \cap \Gamma$ is ϕ-vertex-expanding in Γ. That is, for every vertex cut (L, S, R) of Γ,

$$|S| \geq \phi \min\{|V_\ell \cap (L \cup S)|,\ |V_\ell \cap (R \cup S)|\}.$$

From $\mathcal{P}$, the connected components Γ in $G_{\leq \ell}$ for all levels ℓ form a laminar family $\mathcal{C}$. Let $\mathcal{H}$ be the tree representation of $\mathcal{C}$. We also call $(\mathcal{C}, \mathcal{H})$ an (h, ϕ)-vertex-expander hierarchy of G.

The following theorem is analogous to Theorem 2.3.

Theorem 3.3. *There exists an algorithm that, given a graph G, computes an (h, ϕ)-vertex-expander hierarchy with $h \leq \log n$ and $\phi = 1$ in exponential time, or $h \leq \log n$ and $\phi \geq \Omega(1/\sqrt{\log n})$ in polynomial time.*

Long and Saranurak's [LS22] vertex expander hierarchy is weaker, both qualitatively and structually. To be precise, the Long-Saranurak hierarchy only guarantees $\phi \geq 1/n^{o(1)}$, but it admits almost-linear construction time. Furthermore, the expander components in the Long-Saranurak hierarchy may not form a laminar family. The proof of Theorem 3.3 follows immediately from the lemma below.

Lemma 3.4. *There exists an exponential-time algorithm that, given a graph $G = (V, E)$, computes a vertex set X such that every connected component of $G \setminus X$ contains at most $n/2$ vertices and X is 1-vertex-expanding. In polynomial time, we instead guarantee that X is $\Omega(1/\sqrt{\log n})$-vertex-expanding.*

The proofs of Theorem 3.3 and Lemma 3.4 follow in exactly the same way as how we proved the analogous results in the edge version. We include them for completeness.

Proof. Initialize $X \leftarrow V$. If X is 1-expanding, we are done. Otherwise, we repeatedly update X as follows. Since X is not 1-expanding, there exists a vertex cut (L, S, R) where $|L| \leq n/2$ such that $|S| < \min\{|X \cap (L \cup S)|, |X \cap (R \cup S)|\}$. Update $X \leftarrow X \setminus (X \cap L) \cup S$.

Let X' denote X after the update. Observe that every connected component of $G \setminus X'$ still contains at most $n/2$ vertices because $|L| \leq n/2$. Moreover, $|X'| < |X|$ because, while we added at most $|S \setminus X|$ new vertices to X, we removed $|X \cap L| > |S \setminus X|$ vertices from X where the inequality holds because $|S \cap X| + |S \setminus X| = |S| < |X \cap (L \cup S)| = |X \cap L| + |S \cap X|$. Therefore, there are at most $|V|$ iterations before X is 1-vertex-expanding. This concludes the proof of the exponential-time algorithm.

To get polynomial time, we instead apply the sparsest cut algorithm by [FHL05] that, given X, either guarantees that X is $\Omega(1/\sqrt{\log n})$-vertex-expanding or returns a vertex cut (L, S, R) where $|L| \leq n/2$ such that $|S| < \min\{|X \cap (L \cup S)|, |X \cap (R \cup S)|\}$. $\square$

Proof of Theorem 3.3. Given G, compute the vertex set X from Lemma 3.4 and set $V_h \leftarrow X$. To compute $V_{h-1}, V_{h-2}, \ldots, V_1$, we recurse on each connected component C of $G \setminus X$. We have $h \leq \log n$ because each component C has size $|C| \leq n/2$. $\square$

Notation in subsequent sections. Let $(\mathcal{C}, \mathcal{H})$ be an (h, ϕ)-vertex-expander hierarchy of G. For each level-ℓ component $\Gamma \in \mathcal{C}$, we define $\gamma := V_\ell \cap \Gamma$ to be the *core* of Γ. The following Observation 3.5 is straightforward from the definition.

Observation 3.5. *We have the following.*

1. *There is no edge connecting two disjoint components in $\mathcal{C}$.*

2. *For each component $\Gamma \in \mathcal{C}$, its core γ is ϕ-vertex-expanding in Γ.*

3. *The cores $\{\gamma \mid \Gamma \in \mathcal{C}\}$ partition $V(G)$.*

By convention, the core of a Γ decorated with subscripts, superscripts, and diacritic marks inherits those decorations, e.g., $\hat{\gamma}_i^j$ is the core of $\hat{\Gamma}_i^j$. For two components Γ, Γ' s.t. $\Gamma' \preceq \Gamma$ (resp. $\Gamma' \prec \Gamma$), we also write $\gamma' \preceq \gamma$ (resp. $\gamma' \prec \gamma$). For each component Γ, $N(\Gamma)$ denotes the set of neighbors of Γ in $V - \Gamma$. Define $N_{\hat{\gamma}}(\Gamma) = N(\Gamma) \cap \hat{\gamma}$, which is only non-empty when $\gamma \prec \hat{\gamma}$. We call such $N_{\hat{\gamma}}(\Gamma)$ *neighbor sets*. For each vertex $v \in V(G)$, we use γ_v to denote the unique core containing v, so Γ_v denotes the corresponding component of γ_v.

3.2.2 The Initial Structure: Low-Degree Steiner Trees and Shortcut Graphs

Based on the vertex expander hierarchy, we construct low-degree Steiner trees and shortcut graphs, both of which will help answer connectivity queries.

Low-Degree Steiner Trees. For each component $\Gamma \in \mathcal{C}$, using Lemma 3.6, we will compute a Steiner tree T_γ with maximum degree $\Delta = O(1/\phi)$ that spans the core γ in $G[\Gamma]$. Sometimes, we will call the vertices in Steiner tree T_γ *nodes*, just to be consistent with the terminology of *extended Steiner trees* introduced later. In particular, each node $u \in \gamma \subseteq V(T_\gamma)$ is a *terminal node*, and the other nodes $V(T_\gamma) \setminus \gamma$ are *Steiner nodes*. Observe that each vertex in Γ will correspond to at most one node in T_γ, and vertices in γ are in one-to-one correspondence with terminal nodes in T_γ.

A hierarchy with such low-degree Steiner trees but without the vertex-expanding property was first introduced by Duan and Pettie [DP20] as the *low-degree hierarchy*, which has been shown to be useful for the vertex-failure connectivity problem in both the centralized [DP20, LS22] and labeling scheme [PPP24] settings. Roughly speaking, these Steiner trees are useful because they serve as connectivity certificates. By the low-degree property, when f vertices fail, each Steiner tree will be broken into at most $O(f/\phi)$ subtrees, each of them still being connected in the new graph. The query algorithm need only look for edges that reconnect the subtrees rather than determine connectivity from scratch.

Long and Saranurak [LS22] gave an almost linear time algorithm to compute an $O(\log n/\phi)$-degree Steiner tree spanning a ϕ-vertex expanding set A in G. We give an improved algorithm that computes an $O(1/\phi)$-degree Steiner trees based on Fürer and Raghavachari [FR94], albeit with a slower running time. This improvement to the degree will shave logarithmic factors off our final label size. The algorithm below can be of independent interest. Its proof is deferred to Appendix A.

Lemma 3.6 (Low-degree Steiner Trees)**.** *Given a graph G such that a set $A \subseteq V(G)$ is ϕ-vertex-expanding in G, there is an algorithm that computes an $O(1/\phi)$-degree Steiner tree that spans A in G. The running time is $O(mn \log n)$.*

The Neighborhood Hitter $\mathcal{S}$. We want the Steiner trees to have low degree Δ so that f vertex failures generate at most $f\Delta$ subtrees. This argument only requires that *failed* vertices have low degree. Following [PPP24], we generate a partition $\mathcal{S} = (S_1, \ldots, S_{f+1})$ of the vertex set, and build a version of the data structure for each $S_i \in \mathcal{S}$, which one can think of as vertices that are *not allowed* to fail. By the pigeonhole principle, for any failure set F there exists an $S = S_i$ such that $S \cap F = \emptyset$. Thus, it is fine if, in the data structure with failure-free set S, all S-vertices have *unbounded* degrees.

The main benefit of having a failure-free S is to effectively reduce neighborhood sizes, as follows. If we were to generate the partition $\mathcal{S}$ randomly, then with high probability either (i) $N_{\hat{\gamma}}(\Gamma) \cap S \neq \emptyset$ or (ii) $|N_{\hat{\gamma}}(\Gamma)| \leq \lambda_{\text{nb}} \overset{\text{def}}{=} O(f \log n)$. In case (i) we can link $T_{\hat{\gamma}}$ and T_γ without increasing the degrees of non-S vertices by much (see *extended cores* below), and in case (ii) we have a good upper bound on $|N_{\hat{\gamma}}(\Gamma)|$. In fact, it is possible to achieve this guarantee deterministically using the method of conditional expectations [PPP24]. Concretely, we just compute $\mathcal{S}$ by invoking Lemma 3.7 with all such neighbor sets $N_{\hat{\gamma}}(\Gamma)$ as the inputs.

Lemma 3.7 ([PPP24, Lemma 8.1])**.** *Given a graph G with a polynomial number of vertex sets $\{B_k \subseteq V \mid 1 \leq k \leq \text{poly}(n)\}$, there is a deterministic algorithm that computes a partition $\mathcal{S} = \{S_1, ..., S_{f+1}\}$ of $V(G)$ s.t. for each S_i and B_k, either $S_i \cap B_k \neq \emptyset$ or $|B_k| \leq O(f \log n)$. The running time is polynomial.*

Henceforth we use S to refer to an arbitrary part of the partition $\mathcal{S}$. In the preprocessing phase we generate a data structure for each $S \in \mathcal{S}$, but in the context of a query $\langle s, t, F \rangle$, S refers to any part for which $S \cap F = \emptyset$.

Extended Cores and Extended Steiner Trees. Each component $\Gamma \in \mathcal{C}$ has an *extended core* γ^{ext} defined as follows:

$$\gamma^{\text{ext}} \overset{\text{def}}{=} \gamma \cup \bigcup_{\substack{\gamma' \prec \gamma \\ \text{s.t. } N_\gamma(\Gamma') \cap S \neq \emptyset}} \gamma'.$$

Observe that if $\gamma' \nsubseteq \gamma^{\text{ext}}$, then $|N_\gamma(\Gamma')| \leq \lambda_{\text{nb}} = O(f \log n)$. Whenever $N_{\hat{\gamma}}(\Gamma) \cap S \neq \emptyset$ is non-empty, let $s_{\hat{\gamma}}(\Gamma) \in N_{\hat{\gamma}}(\Gamma) \cap S$ be an arbitrary representative in its neighborhood set.

Just as each core γ has a Steiner tree T_γ, the extended core γ^{ext} has an *extended Steiner tree* $T_{\gamma^{\text{ext}}}$. Each tree node in $V(T_{\gamma^{\text{ext}}})$ is either a *terminal node* or *Steiner node.* As we will see in the construction, the terminal nodes are one-one corresponding to vertices in γ^{ext}. Each Steiner node corresponds to exactly one vertex in Γ, while each vertex in Γ can correspond to arbitrary numbers of Steiner node in $V(T_{\gamma^{\text{ext}}})$.

Construction of $T_{\gamma^{\text{ext}}}$. The construction of $T_{\gamma^{\text{ext}}}$ is as follows.

1. First, we make a copy of T_γ, and for each strict descendant $\gamma' \subseteq \gamma^{\text{ext}}$, make a copy of $T_{\gamma'}$.
2. Let $P_{\gamma' \to \gamma}$ be a copy of an arbitrary simple path in the graph $G[\Gamma' \cup \{s_\gamma(\Gamma')\}]$ connecting the vertex $s_\gamma(\Gamma') \in \gamma$ (which corresponds to the terminal node $s_\gamma(\Gamma') \in V(T_\gamma)$) to some vertex $v' \in \Gamma'$ such that v' corresponds to some tree node in $V(T_{\gamma'})$.
3. Finally, we obtain $T_{\gamma^{\text{ext}}}$ by attaching the copy of $T_{\gamma'}$ (for all strict descendants $\gamma' \subseteq \gamma^{\text{ext}}$) to the copy of T_γ using the path $P_{\gamma' \to \gamma}$. That is, we glue the endpoint $s_\gamma(\Gamma')$ of $P_{\gamma' \to \gamma}$ to the terminal node $s_\gamma(\Gamma') \in V(T_\gamma)$, and glue the other endpoint v' of $P_{\gamma' \to \gamma}$ to the tree node $v' \in V(T_{\gamma'})$.

By the construction, $V(T_{\gamma^{\text{ext}}})$ is made up of $V(T_\gamma)$, $V(T_{\gamma'})$ of each strict descendant $\gamma' \subseteq \gamma^{\text{ext}}$, and the internal nodes of each path $P_{\gamma' \to \gamma}$. We define the terminal nodes in $V(T_{\gamma^{\text{ext}}})$ to be

- the terminal nodes in $V(T_\gamma)$ (they one-one correspond to vertices in γ), and
- the terminal nodes in $V(T_{\gamma'})$ for each strict descendant $\gamma' \subseteq \gamma^{\text{ext}}$ (they one-one correspond to vertices in γ').

Other tree nodes in $V(T_{\gamma^{\text{ext}}})$ are Steiner nodes. By definition, the terminal nodes in $V(T_{\gamma^{\text{ext}}})$ one-one correspond to vertices in γ^{ext}.

Properties of $T_{\gamma^{\text{ext}}}$. First, $T_{\gamma^{\text{ext}}}$ has some kind of low degree guarantee of non-S vertices, as shown in Lemma 3.8.

Lemma 3.8. *For each vertex $v \in \Gamma \setminus S$, the tree nodes corresponding to v have total $T_{\gamma^{\text{ext}}}$-degree at most $O(h\Delta)$.*

Proof. Recall that $V(T_{\gamma^{\text{ext}}})$ is made up of $V(T_\gamma)$, $V(T_{\gamma'})$ of each strict descendant $\gamma' \subseteq \gamma^{\text{ext}}$, and the internal nodes of each path $P_{\gamma' \to \gamma}$.

- v can correspond to at most one tree node in $V(T_\gamma)$. This tree node have degree at most Δ in $T_{\gamma^{\text{ext}}}$ because it is not an endpoint of any path $P_{\gamma' \to \gamma}$ (since $v \notin S$).

- For each strict descendant $\gamma' \subseteq \gamma^{\text{ext}}$, if $v \in \Gamma'$, v can correspond to at most one tree node in $V(T_{\gamma'})$ (note that if $v \notin \Gamma'$, v must correspond to no tree node in $V(T_{\gamma'})$ since $V(T_{\gamma'}) \subseteq \Gamma'$). This tree node have $T_{\gamma^{\text{ext}}}$-degree at most $\Delta + 1$ because it has degree at most Δ in $T_{\gamma'}$ and it has degree 1 in $P_{\gamma' \to \gamma}$ if it is an endpoint of $P_{\gamma' \to \gamma}$.

  The number of such γ' is at most h since the number of components containing v is at most h. Therefore, this part contributes at most $h(\Delta + 1)$.

- For each path $P_{\gamma' \to \gamma}$, if $v \in \Gamma'$, v can correspond to at most one internal node of $P_{\gamma' \to \gamma}$ (note that if $v \notin \Gamma'$, v will correspond to no internal node of $P_{\gamma' \to \gamma}$ since all internal nodes are in Γ'). This node has $T_{\gamma^{\text{ext}}}$-degree 2.

  Again the number of such path $P_{\gamma' \to \gamma}$ is at most h, and this part contributes at most $2h$.

□

The second property is the simple Observation 3.9. We will exploit it in the proof of Lemma 3.13 and Section 3.3.

Observation 3.9. *For each $\gamma' \prec \gamma$ such that $\gamma' \subseteq \gamma^{\text{ext}}$, its Steiner tree $T_{\gamma'}$, or even $T_{\gamma'} \cup P_{\gamma' \to \gamma}$, is a subtree of $T_{\gamma^{\text{ext}}}$, where $T_{\gamma'} \cup P_{\gamma' \to \gamma}$ denote the tree obtained by gluing the endpoint v' of $P_{\gamma' \to \gamma}$ to the tree node $v' \in V(T_{\gamma'})$.*

Lemma 3.8 shows that when f vertices fail, they break $T_{\gamma^{\text{ext}}}$ into at most $O(fh\Delta) = O(f \log n/\phi)$ subtrees, since S-vertices are not allowed to fail. The analogue of Lemma 3.8 in [PPP24] creates extended Steiner trees with degree $\Delta + h$ rather than $O(h\Delta)$, but might not satisfy the property that $T_{\gamma'}$ is a subtree of $T_{\gamma^{\text{ext}}}$, which is used in our labeling scheme.

The Shortcut Graphs. The *global shortcut graph* $\hat{G}$ is also defined w.r.t. an arbitrary $S \in \mathcal{S}$. It is formed by adding *shortcut edges* to G, each with an assigned *type*.[4] For each component $\Gamma \in \mathcal{C}$ and each strict ancestor $\hat{\Gamma}$ of Γ, we define

$$\hat{N}_{\hat{\gamma}}(\Gamma) = \begin{cases} N_{\hat{\gamma}}(\Gamma) & \text{if } N_{\hat{\gamma}}(\Gamma) \text{ is disjoint from } S, \\ \{s_{\hat{\gamma}}(\Gamma)\} & \text{if } N_{\hat{\gamma}}(\Gamma) \text{ intersects } S. \end{cases}$$
$$\hat{N}(\Gamma) = \bigcup_{\hat{\gamma}} \hat{N}_{\hat{\gamma}}(\Gamma).$$

$\hat{G}$ is a *typed* multigraph with the same vertex set as G and

$$E(\hat{G}) = E(G) \cup \bigcup_{\Gamma} \mathsf{Clique}(\hat{N}(\Gamma)),$$

where all edges in $E(G)$ have type *original* and all edges in the clique $\mathsf{Clique}(\hat{N}(\Gamma))$ have type γ.

The *shortcut graph w.r.t.* $\Gamma \in \mathcal{C}$, denoted by $\hat{G}_\Gamma$ is the subgraph of $\hat{G}$ induced by edges with both endpoints in Γ and at least one in the core γ.

[4]Previous papers call a shortcut edge an artificial edge and call a shortcut graph an auxiliary graph. We change the names to make them more descriptive.

The idea of adding shortcut edges has appeared in prior works [DP20, LS22, PPP24, CPR11] on the vertex-failure connectivity problem. Intuitively, the simplest way to add shortcut edges is to add a clique on $N(\Gamma)$ for each component Γ. With the shortcut edges, when failed vertices come, if some component Γ is *unaffected* (it has no failed vertices) the query algorithm can ignore vertices in Γ, and use the shortcut edges to capture the connectivity provided by Γ. However, generally the performance of the algorithm depends on the sparsity of the shortcut edges, so this naive construction will not give good algorithms. Indeed, most prior work [DP20, LS22, PPP24] introduced different sparsification techniques on shortcut edges. In our work, we sparsify the shortcut edges by adding a clique on the sparsified neighbor set $\hat{N}(\Gamma)$ instead of the original one $N(\Gamma)$.

3.2.3 Structures Affected by Queries

In this subsection, we will define notations and terms related to a particular query $\langle s, t, F\rangle$, and then introduce the query strategy from a high level. Recall that $S \in \mathcal{S}$ represents any part of the partition disjoint from F.

Affected Components/Cores, Affected Edges, and Query Graphs. We introduce the following notations and terms.

- For each component $\Gamma \in \mathcal{C}$, if Γ intersects $F \cup \{s, t\}$, we say Γ is an *affected component* and γ is an *affected core*, otherwise they are *unaffected*.
- For each edge $e = \{u, v\} \in E(\hat{G}_\Gamma)$ in the shortcut graph w.r.t. Γ, it is an *affected edge* if the type of e is γ' for some affected γ'. Let $\hat{E}_{\Gamma,\text{aff}} \subseteq E(\hat{G}_\Gamma)$ collect all affected edges in $\hat{G}_\Gamma$, and let $\hat{E}_{\Gamma,\text{unaff}} = E(\hat{G}_\Gamma) \setminus \hat{E}_{\Gamma,\text{aff}}$ be the set of *unaffected* edges.
- For each affected component Γ, its *query set* is $Q_\Gamma = \gamma \cup \bigcup_{\text{affected } \gamma' \prec \gamma} \gamma'$ and its *extended query set* is $Q^{\text{ext}}_\Gamma = \gamma^{\text{ext}} \cup \bigcup_{\text{affected } \gamma' \prec \gamma} \gamma'$.
- We define the *query graph* $\hat{G}^{\text{qry}}_\Gamma$ of Γ to be $\hat{G}^{\text{qry}}_\Gamma = \hat{G}_\Gamma[Q^{\text{ext}}_\Gamma] \setminus \hat{E}_{\Gamma,\text{aff}}$. Namely, the query graph $\hat{G}^{\text{qry}}_\Gamma$ is the subgraph of $\hat{G}_\Gamma$ induced by the extended query set Q^{ext}_Γ, excluding all affected edges.

Observation 3.10. *The number of affected components is at most $h(f + 2)$.*

3.2.4 A Divide-and-Conquer Lemma

Next, we state the key lemma, Lemma 3.11, saying the connectivity after failures can be captured by the structure we defined in previous sections. Roughly, for any affected component Γ, the connectivity between vertices can be captured by either (1) shortcut edges in Γ, (2) extended Steiner trees $T_{\gamma^{\text{ext}}}$, or (3) the recursive structure on affected children of Γ. Naturally, this equivalence hints at a divide-and-conquer strategy by querying bottom-up from the hierarchy. We will formally describe this strategy in Section 3.3.

Before stating Lemma 3.11, we introduce some notations. For an undirected graph H and a subset of vertices $A \subseteq V(H)$, we define $\mathsf{Conn}(A, H)$ to be an undirected graph on vertices A s.t. an edge (u, v) exists in $\mathsf{Conn}(A, H)$ if and only if u and v are connected in H. We note that when H

refers to an extended Steiner tree and $A \subseteq V(G)$ refers to a set of original vertices, this notation $\mathsf{Conn}(A, H)$ is still well-defined as long as each vertex in A corresponds to exactly one terminal node in H. For an extended Steiner tree $T_{\gamma^{\text{ext}}}$, we use $T_{\gamma^{\text{ext}}} \setminus F$ to denote the forest by removing all nodes corresponding to vertices in F.

Lemma 3.11. *Let $\Gamma \in \mathcal{C}$ be an affected component. Each pair of vertices $x, y \in Q_\Gamma \setminus F$ are connected in $G[\Gamma] \setminus F$ if and only if they are connected in the union of*

1. $\mathsf{Conn}(Q^{\text{ext}}_\Gamma \setminus F, \hat{G}^{\text{qry}}_\Gamma \setminus F)$,
2. $\mathsf{Conn}(\gamma^{\text{ext}} \setminus F, T_{\gamma^{\text{ext}}} \setminus F)$, *and*
3. $\bigcup_{\Gamma_{\text{child}}} \mathsf{Conn}(Q_{\Gamma_{\text{child}}} \setminus F, G[\Gamma_{\text{child}}] \setminus F)$, *where the union is over all* affected *children Γ_{child} of Γ.*

Proof. It is relatively simple to see that x, y are connected in $G_\Gamma \setminus F$ if they are connected in the union, because each of $\hat{G}^{\text{qry}}_\Gamma \setminus F$, $T_{\gamma^{\text{ext}}} \setminus F$ and $G[\Gamma_{\text{child}}] \setminus F$ only use valid connectivity in $G[\Gamma] \setminus F$. To be precise, for the graph $\hat{G}^{\text{qry}}_\Gamma \setminus F$, consider an edge $e = \{u, v\}$ in it.

- If e has type *original* then e also exists in $G[\Gamma] \setminus F$.
- Otherwise, e is a shortcut edge with some type γ' such that γ' is unaffected. By definition, $u, v \in N(\Gamma')$ and Γ' is disjoint from F, hence u, v are connected in $G[\Gamma] \setminus F$.

Similarly, $T_{\gamma^{\text{ext}}} \setminus F$ is a subgraph of $G[\Gamma] \setminus F$ and $G[\Gamma_{\text{child}}] \setminus F$ is obviously a subgraph of $G[\Gamma] \setminus F$.

From now we focus on proving the other direction. Let P_{xy} be a simple path connecting x and y in $G[\Gamma] \setminus F$. We can write P_{xy} as $P_{xy} = P_1 \circ P_2 \circ \cdots \circ P_\ell$ where the endpoints u_i, v_i of P_i are the Q_Γ-vertices and each subpath P_i are internally disjoint with Q_Γ. It suffices to show that for each subpath P_i, u_i and v_i are connected in the union.

Case (a). Suppose $\gamma_{u_i}, \gamma_{v_i} \prec \gamma$ (recall that $\gamma_{u_i}, \gamma_{v_i}$ are the cores containing u_i, v_i respectively). Then there is a child Γ_{child} of Γ s.t. $\gamma_{u_i}, \gamma_{v_i} \preceq \gamma_{\text{child}}$ and all vertices in P_i are inside $\Gamma_{\text{child}} \setminus F$. To see this, assume for contradiction that P_i contains two vertices from two different children $\Gamma_{\text{child}}, \Gamma'_{\text{child}}$ of Γ. However, by property 1 of the hierarchy, P_i must go through some vertex in γ under this assumption, contradicting that P_i is internally disjoint from Q_Γ. Furthermore, we know γ_{child} is affected because γ_{u_i} and γ_{v_i} are affected (since $\gamma_{u_i}, \gamma_{v_i} \subseteq Q_\Gamma$ and Q_Γ only collects affected cores). Also, note that $u_i, v_i \in Q_{\Gamma_{\text{child}}} \setminus F$ because $Q_\Gamma \cap \Gamma_{\text{child}} = Q_{\Gamma_{\text{child}}}$. Putting it all together, we know u_i and v_i are connected in $\mathsf{Conn}(Q_{\Gamma_{\text{child}}} \setminus F, G[\Gamma_{\text{child}}] \setminus F)$ (i.e. Part 3) by the existence of P_i.

In what follows, we will argue that arbitrary two vertices $u, v \in Q_\Gamma \setminus F$ are connected in the union, if they satisfy

(i) $u \in \gamma$ or $v \in \gamma$, and

(ii) there is an original edge $e = \{u, v\} \in E(G[\Gamma])$ or there is an *unaffected* component $\Gamma' \prec \Gamma$ s.t. $u, v \in N(\Gamma')$.

Indeed, for each above subpath P_i not in Case (a), its endpoints u_i and v_i must satisfy the conditions (i) and (ii). Condition (i) is easy to see. For condition (ii), if P_i has only one edge, that this edge $e = \{u_i, v_i\}$ is an original edge in $G[\Gamma]$. If P_i has more than one edge, all internal vertices of P_i fall in unaffected cores, because P_i is internally disjoint from Q_Γ. Again by property 1 of the hierarchy, there exists an unaffected component Γ' containing all internal vertices of P_i, so $u_i, v_i \in N(\Gamma')$.

Case (b). Suppose condition (ii) tells there is an original edge $e = \{u, v\} \in E(G[\Gamma])$. Then this edge is added to $\hat{G}_\Gamma$ because $u \in \gamma$ or $v \in \gamma$, and it is inside $\hat{G}_\Gamma^{\text{qry}}$ because it has type original and $u, v \in Q_\Gamma \setminus F \subseteq Q_\Gamma^{\text{ext}} \setminus F$. Therefore, u and v are connected in $\mathsf{Conn}(Q_\Gamma^{\text{ext}} \setminus F, \hat{G}_\Gamma^{\text{qry}} \setminus F)$ (i.e. Part 1).

From now on, we suppose condition (ii) tells there is an unaffected component $\Gamma' \prec \Gamma$ s.t. $u, v \in N(\Gamma')$. We can further assume u, v, Γ' satisfy condition

(iii) there is a path P connecting u and v with all internal vertices inside Γ', and P intersects the core γ'.

This is without loss of generality by letting Γ' be the *minimal* component s.t. $u, v \in N(\Gamma')$.

Case (c). Suppose $u, v \in \gamma$.

Subcase (c1). Suppose $N_\gamma(\Gamma')$ is disjoint from S. Recall the way we add shortcut edges, we add a clique on $\hat{N}(\Gamma') \supseteq N_\gamma(\Gamma')$ with type γ'. Because $u, v \in N(\Gamma') \cap \gamma = N_\gamma(\Gamma')$, there is a shortcut edge $e = \{u, v\}$ in this clique. Note that e will be added into $\hat{G}_\Gamma$ because $u \in \gamma$ or $v \in \gamma$, and e will survive in $\hat{G}_\Gamma^{\text{qry}} \setminus F$ because $u, v \in Q_\Gamma^{\text{ext}} \setminus F$ and the type γ' is unaffected. Therefore, u and v are connected in $\mathsf{Conn}(Q_\Gamma^{\text{ext}} \setminus F, \hat{G}_\Gamma^{\text{qry}} \setminus F)$ (i.e. Part 1).

Subcase (c2). Otherwise $N_\gamma(\Gamma')$ intersects S. By definition, γ' is in the extended core γ^{ext} of component Γ. Condition (iii) tells that P intersect γ'. Let $w_u, w_v \in P \cap \gamma'$ be the P-vertices closest to u and v respectively. Claim 3.12 tells that w_u and w_v are connected in $\mathsf{Conn}(\gamma^{\text{ext}} \setminus F, T_{\gamma^{\text{ext}}} \setminus F)$ (i.e. Part 2).

Claim 3.12. *We have $w_u, w_v \in \gamma^{\text{ext}} \setminus F$, and the terminal nodes w_u and w_v are connected in $T_{\gamma^{\text{ext}}} \setminus F$.*

Proof. First, we have $w_u, w_v \in \gamma' \setminus F \subseteq \gamma^{\text{ext}} \setminus F$. Next, the terminal nodes w_u and w_v are connected in $T_{\gamma^{\text{ext}}} \setminus F$ is because (1) the terminal nodes w_u and w_v fall in $T_{\gamma'}$ (since the vertices w_u, w_v are inside γ'), (2) the Steiner tree $T_{\gamma'}$ connects w_u and w_v, (3) $T_{\gamma'}$ is a subtree of $T_{\gamma^{\text{ext}}}$ (since $\gamma' \subseteq \gamma^{\text{ext}}$) and (4) $T_{\gamma'}$ is disjoint from F (since Γ' is unaffected). □

It remains to show that $u \in \gamma$ and $w_u \in \gamma'$ are connected in the union, and that $w_v \in \gamma'$ and $v \in \gamma$ are connected in the union. Actually, these two claims can be derived from Lemma 3.13. Here we show that u and w_u satisfy the requirements of Lemma 3.13 (so do v and w_v by a similar argument). For u, trivially $u \notin F$ and $u \in \gamma$. For w_u, recall that γ_{w_u} denotes the core containing w_u, and we know $\gamma_{w_u} = \gamma'$. Hence, as we mentioned above, $\gamma_{w_u} \prec \gamma$, γ_{w_u} is unaffected and $N_\gamma(\Gamma_{w_u})$ intersects S. Because we take the $w_u \in P \cap \gamma'$ closest to u, the subpath of P from u to w_u either has one edge or has all interval vertices inside some $\gamma'' \prec \gamma_{w_u}$, which means $e = \{u, w_u\} \in E(G[\Gamma])$ or $u, w_u \in N(\Gamma'')$.

Case (d). Suppose $u \in \gamma$ and $v \in \gamma_v \prec \gamma$. Note that u and v satisfy the requirements of Lemma 3.13, so they are connected in the union.

Lemma 3.13. *Let u, v be two vertices satisfying that*

- *$u \notin F$ and $u \in \gamma$;*

- *$v \notin F$, $\gamma_v \prec \gamma$ and either*
  - *γ_v is affected or*
  - *γ_v is unaffected and $N_\gamma(\Gamma_v)$ intersects S;*
- *The edge $e = \{u, v\} \in G[\Gamma]$, or there is an unaffected component $\gamma' \prec \gamma_v$ s.t. $u, v \in N(\Gamma')$.*

We have u and v are connected in the union.

Proof. First observe that γ_v belongs to the extended core γ^{ext} of γ because $N_\gamma(\Gamma_v)$ intersects S.

Similar to the Case (b) above, if there is an original edge $e = \{u, v\} \in E(G[\Gamma])$, then this edge is added to $\hat{G}_\Gamma$ because $u \in \gamma$, and it survives in $\hat{G}^{\text{qry}}_\Gamma \setminus F$ because it has type original, $u \in \gamma \in Q^{\text{ext}}_\Gamma$ and $v \in \gamma_v \subseteq \gamma^{\text{ext}} \subseteq Q^{\text{ext}}_\Gamma$. Therefore, u and v are connected in $\mathsf{Conn}(Q^{\text{ext}}_\Gamma \setminus F, \hat{G}^{\text{qry}}_\Gamma \setminus F)$ (i.e. Part 1).

From now we assume there is an unaffected component $\gamma' \prec \gamma_v$ s.t. $u, v \in N(\Gamma')$. The following argument is similar to the Case (d) above. Again, by choosing γ' whose component Γ' is the minimal one that contains all internal vertices of P, we can assume there is a path P connecting u and v with all internal vertices falling in Γ', and P intersects γ'.

Case (1). Suppose $N_{\gamma_v}(\Gamma')$ intersects S, and $v \neq s_{\gamma_v}(\Gamma')$. We show that v and $v' \overset{\text{def}}{=} s_{\gamma_v}(\Gamma')$ are connected in the union.

- If γ_v is affected, the vertices v and v' are connected by Part 3 by the following reasons. Let Γ_{child} be the child of Γ such that $\Gamma_v \preceq \Gamma_{\text{child}}$. First, γ_{child} is affected because $\gamma_v \subseteq Q_\Gamma$ (it means γ_v is affected) and $\gamma_v \preceq \gamma_{\text{child}}$. Second, $v, v' \in Q_{\Gamma_{\text{child}}} \setminus F$ because $v, v' \in \gamma_v \setminus F$ ($v' \notin F$ because $v' \in S$ and S is disjoint from F), and $\gamma_v \subseteq Q_\Gamma \cap \Gamma_{\text{child}} = Q_{\Gamma_{\text{child}}}$. Third, v and v' are connected in $G[\Gamma_{\text{child}}] \setminus F$ because $v, v' \in N(\Gamma')$, Γ' is an unaffected component, and $\Gamma' \prec \Gamma_v \preceq \Gamma_j$.
- If γ_v is unaffected and $N_\gamma(\Gamma_v)$ intersects S, the terminal nodes v and $v' = s_{\gamma_v}(\Gamma')$ are connected in $\mathsf{Conn}(\gamma^{\text{ext}} \setminus F, T_{\gamma^{\text{ext}}} \setminus F)$ (i.e. Part 2) by the following reasons. First, T_{γ_v} is a subtree of $T_{\gamma^{\text{ext}}}$ (since $\gamma_v \subseteq \gamma^{\text{ext}}$ from $N_\gamma(\Gamma_v)$ intersects S). Furthermore, T_{γ_v} has no vertex in F (since γ_v is unaffected), and the terminal nodes in $T_{\gamma^{\text{ext}}}$ corresponding to vertices v, v' are on the subtree T_{γ_v} (since $v, v' \in \gamma_v$).

Therefore, it suffices to show that u is connected to v' in the union, which can be reduced to the following cases.

Case (2). Suppose $N_{\gamma_v}(\Gamma')$ is disjoint from S, or $v = s_{\gamma_v}(\Gamma')$. Further assume that $N_\gamma(\Gamma')$ is disjoint from S. Recall the construction of shortcut edges, we have $u \in N_\gamma(\Gamma') \subseteq \hat{N}(\Gamma')$ and $v \in \hat{N}(\Gamma')$ (if $N_{\gamma_v}(\Gamma')$ is disjoint from S, then $v \in N_{\gamma_v}(\Gamma') \subseteq \hat{N}(\Gamma')$; if $v = s_{\gamma_v}(\Gamma')$, then $v = s_{\gamma_v}(\Gamma') \subseteq \hat{N}(\Gamma')$), so there is a shortcut edge connected u' and v with type γ'. This edge is in $\hat{G}_\Gamma$ because $u \in \gamma$, and it survives in $\hat{G}^{\text{qry}}_\Gamma \setminus F$ because γ' is unaffected, $u \in \gamma \subseteq Q^{\text{ext}}_\Gamma$ and $v \in \gamma_v \subseteq \gamma^{\text{ext}} \subseteq Q^{\text{ext}}_\Gamma$. Therefore, u and v are connected in $\mathsf{Conn}(Q^{\text{ext}}_\Gamma \setminus F, \hat{G}^{\text{qry}}_\Gamma \setminus F)$ (i.e. Part 1).

Case (3). Suppose $N_{\gamma_v}(\Gamma')$ is disjoint from S, or $v = s_{\gamma_v}(\Gamma')$. Further assume that $N_\gamma(\Gamma')$ intersects S. Let $u' = s_\gamma(\Gamma')$. Let w be the P-vertex in γ' closest to u.

First, v and u' are connected in $\mathsf{Conn}(Q^{\text{ext}}_\Gamma \setminus F, \hat{G}^{\text{qry}}_\Gamma \setminus F)$ (i.e. Part 1) by the following reasons. When adding shortcut edges, we have $v, u' \in \hat{N}(\Gamma')$ (by the same reason as above), so there is a shortcut edge connecting u' and v with type γ'. This edge is in $\hat{G}_\Gamma$ because $u \in \gamma$, and it survives in $\hat{G}^{\text{qry}}_\Gamma \setminus F$ because γ' is unaffected, $u' \in \gamma \subseteq Q^{\text{ext}}_\Gamma$, $u' \notin F$ (since $u' \in S$) and $v \in \gamma_v \subseteq \gamma^{\text{ext}} \subseteq Q^{\text{ext}}_\Gamma$.

Next, u' and w are connected in $\mathsf{Conn}(\gamma^{\text{ext}} \setminus F, T_{\gamma^{\text{ext}}} \setminus F)$ (i.e. Part 2). This is because in the extended Steiner tree $T_{\gamma^{\text{ext}}}$, the terminal nodes u' and w fall in subtree $T_{\gamma'} \cup P_{\gamma' \to \gamma}$ (from Observation 3.9), and all vertices in $T_{\gamma'} \cup P_{\gamma' \to \gamma}$ are not in F (since all vertices in $T_{\gamma'}$ fall in the unaffected component Γ', and all vertices in $P_{\gamma' \to \gamma}$ also fall in Γ' except one endpoint $s_\gamma(\Gamma')$, which is in S).

It remains to show that w and u are connected in the union. We can reuse the argument in the whole proof of Lemma 3.13. We now verify that u and w satisfy the conditions of Lemma 3.13. Because u is unchanged, $u \notin F$ and $u \in \gamma$. For w (note that $\gamma_w = \gamma'$), because $w \in \gamma_w \prec \gamma_v \prec \gamma$ and γ_v is unaffected, we know $w \notin F$, $\gamma_w \prec \gamma$ and γ_w is unaffected. Moreover, $N_\gamma(\gamma_w)$ intersects S is from the assumption of case (3). Next, consider the subpath of P from u to w, denoted by P'. If P' has only one edge $e = \{u, w\}$, then we are good. Otherwise, P' has some internal vertices. Note that by our choice of w, the internal vertices of P' are all in $\Gamma' \setminus \gamma'$. By Property Item 1 of the hierarchy, they must belong to Γ'_{child} for some child γ'_{child} of γ'. In other words, the unaffected core $\gamma'_{\text{child}} \prec \gamma_w$ has $u, w \in N(\Gamma'_{\text{child}})$. In conclusion u, w indeed satisfy the conditions of Lemma 3.13. Finally, this recursive argument will stop because when we reach Case (3) again, the γ' this time will have depth larger than the one of the last time, and the hierarchy has finite depth. □

□

3.2.5 An Improved Divide-and-Conquer Lemma via Sparsified Shortcut Graphs

The goal of this section is to show an improved version Lemma 3.11 which proves precisely the same statement except that the query graph $\hat{G}^{\text{qry}}_\Gamma$ is replaced by its sparsified version $\widetilde{G}^{\text{qry}}_\Gamma$.

Lemma 3.14. *Let $\Gamma \in \mathcal{C}$ be an affected component. For each pair of vertices $x, y \in Q_\Gamma \setminus F$, they are connected in $G[\Gamma] \setminus F$ if and only if they are connected in the union of*

1. $\mathsf{Conn}(Q^{\text{ext}}_\Gamma \setminus F, \widetilde{G}^{\text{qry}}_\Gamma \setminus F)$,
2. $\mathsf{Conn}(\gamma^{\text{ext}} \setminus F, T_{\gamma^{\text{ext}}} \setminus F)$, *and*
3. $\mathsf{Conn}(Q_{\Gamma_{\text{child}}} \setminus F, G[\Gamma_{\text{child}}] \setminus F)$ *for all affected children Γ_{child} of Γ.*

Working the sparsified query graph $\widetilde{G}^{\text{qry}}_\Gamma$ is crucial for bounding the size of the vertex label to be poly$(f \log n)$ in Section 3.4. This technique was also used in [PPP24]. However, sparsified query graphs are not crucial for understanding the overall strategy of the algorithm in Section 3.3. Hence, during the first read, we suggest readers assume $\widetilde{G}^{\text{qry}}_\Gamma = \hat{G}^{\text{qry}}_\Gamma$, skip this section, and continue until when sparsified query graphs are needed in Section 3.4.

Below, we define sparsified shortcut graphs and sparsified query graphs and then prove Lemma 3.14.

Sparsified Shortcut Graphs. Here, we define the sparsified shortcut graph $\widetilde{G}_\Gamma$ of the shortcut graph $\hat{G}_\Gamma$ with respect to each component $\Gamma \in \mathcal{C}$. Roughly speaking but not precisely, we will sparsify the subgraph $\hat{G}_\Gamma[\gamma^{\text{ext}}]$, abbreviated as $\hat{G}_{\gamma^{\text{ext}}}$, into a graph $\widetilde{G}_{\gamma^{\text{ext}}}$ with arboricity $\widetilde{O}(f^2)$ while keeping the (pairwise) connectivity unchanged under $\widetilde{O}(f^2)$ vertex failures. We will see later why it should tolerate $\widetilde{O}(f^2)$ failures rather than f. Let us recall the guarantees of the Nagamochi-Ibaraki [NI92] sparsifiers.

Lemma 3.15 (Nagamochi and Ibaraki [NI92])**.** *Given a* simple *undirected graph R and a parameter d, there is a deterministic algorithm that computes a subgraph $\widetilde{R}$ of R with $V(\widetilde{R}) = V(R)$ satisfying the following.*

1. *$\widetilde{R}$ has arboricity d.*
2. *Given arbitrary vertex failures $F \subseteq V(R)$ s.t. $|F| < d$, each pair of vertices $u, v \in V(R) \setminus F$ are connected in $R \setminus F$ if and only if they are connected in $\widetilde{R} \setminus F$.*

A graph with arboriticy d is one whose edge-set can be partitioned into d forests and as a consequence, the edge-set can be *oriented* so that the maximum out-degree is d. Parter et al. [PPP24] also computed low-arboricity sparsifiers deterministically, but used hit-miss hash families [KP21] rather than Nagamochi-Ibaraki sparsification.

Finally, we are ready to formally describe how to sparsify $\hat{G}_\Gamma$ into $\widetilde{G}_\Gamma$.

1. Let $\hat{G}_{\gamma^{\text{ext}}} = \hat{G}_\Gamma[\gamma^{\text{ext}}]$ be the subgraph of $\hat{G}_\Gamma$ induced by the extended core γ^{ext}, and let $\hat{G}^{\text{sp}}_{\gamma^{\text{ext}}}$ be the *simple* graph corresponding to $\hat{G}_{\gamma^{\text{ext}}}$, i.e., a bundle of edges with the same endpoints but different types collapse to one edge.
2. We obtain the sparsified graph $\widetilde{G}^{\text{sp}}_{\gamma^{\text{ext}}}$ of $\hat{G}^{\text{sp}}_{\gamma^{\text{ext}}}$ by applying Lemma 3.15 with parameter $d = \lambda_{\text{arbo}} \overset{\text{def}}{=} f + h^2 f \lambda_{\text{nb}} + 1 = O(f^2 \log^3 n)$.
3. Obtain $\widetilde{G}_{\gamma^{\text{ext}}}$ by including, for each $\{u, v\} \in E(\widetilde{G}^{\text{sp}}_{\gamma^{\text{ext}}})$, *all* edges with endpoints $\{u, v\}$ in $\hat{G}_{\gamma^{\text{ext}}}$ (such edges may have various types).
4. Lastly, obtain $\widetilde{G}_\Gamma$ by substituting $\widetilde{G}_{\gamma^{\text{ext}}}$ for $\hat{G}_{\gamma^{\text{ext}}}$ in $\hat{G}_\Gamma$. Namely, $E(\widetilde{G}_\Gamma) = (E(\hat{G}_\Gamma) \setminus E(\hat{G}_{\gamma^{\text{ext}}})) \cup E(\widetilde{G}_{\gamma^{\text{ext}}})$. By this definition, $\widetilde{G}_\Gamma[\gamma^{\text{ext}}] = \widetilde{G}_{\gamma^{\text{ext}}}$.

Sparsified Query Graphs. Recall that we define the *query graph* $\hat{G}^{\text{qry}}_\Gamma$ of Γ to be $\hat{G}^{\text{qry}}_\Gamma = \hat{G}_\Gamma[Q^{\text{ext}}_\Gamma] \setminus \hat{E}_{\Gamma,\text{aff}}$. Naturally, we define a *sparsified query graph* $\widetilde{G}^{\text{qry}}_\Gamma = \widetilde{G}_\Gamma[Q^{\text{ext}}_\Gamma] \setminus \hat{E}_{\Gamma,\text{aff}}$. That is, the sparsified query graph $\hat{G}^{\text{qry}}_\Gamma$ is the subgraph of $\widetilde{G}_\Gamma$ induced by the extended query set Q^{ext}_Γ, excluding all affected edges.

Now, we proceed to prove Lemma 3.14.

Lemma 3.16. *Let $\Gamma \in \mathcal{C}$ be an affected component. Two vertices $x, y \in \gamma^{\text{ext}} \setminus F$ are connected in $\hat{G}^{\text{qry}}_\Gamma[\gamma^{\text{ext}}] \setminus F$ if and only if they are connected in $\widetilde{G}^{\text{qry}}_\Gamma[\gamma^{\text{ext}}] \setminus F$.*

Proof. By definition, $\hat{G}^{\text{qry}}_\Gamma[\gamma^{\text{ext}}] = \hat{G}_\Gamma[\gamma^{\text{ext}}] \setminus \hat{E}_{\Gamma,\text{aff}}$ and $\widetilde{G}^{\text{qry}}_\Gamma[\gamma^{\text{ext}}] = \widetilde{G}_\Gamma[\gamma^{\text{ext}}] \setminus \hat{E}_{\Gamma,\text{aff}}$, because $\gamma^{\text{ext}} \subseteq Q^{\text{ext}}_\Gamma$. Recall that when sparsifying $\hat{G}_\Gamma$ into $\widetilde{G}_\Gamma$, we construct $\widetilde{G}_{\gamma^{\text{ext}}} = \widetilde{G}_\Gamma[\gamma^{\text{ext}}]$ from $\hat{G}_{\gamma^{\text{ext}}} = \hat{G}_\Gamma[\gamma^{\text{ext}}]$ by the following way.

- First let $\hat{G}^{\text{sp}}_{\gamma^{\text{ext}}}$ be the corresponding simple graph of $\hat{G}_{\gamma^{\text{ext}}}$.
- $\widetilde{G}^{\text{sp}}_{\gamma^{\text{ext}}}$ is the sparsified simple graph from applying Lemma 3.15 on $\hat{G}^{\text{sp}}_{\gamma^{\text{ext}}}$ with parameter $d = f + h^2 f \lambda_{\text{nb}} + 1$.
- Lastly let $\widetilde{G}_{\gamma^{\text{ext}}} = \hat{G}_{\gamma^{\text{ext}}} \cap E(\widetilde{G}^{\text{sp}}_{\gamma^{\text{ext}}})$.

Trivially, x and y are connected in $\hat{G}^{\mathrm{qry}}_{\Gamma}[\gamma^{\mathrm{ext}}]$ if they are connected in $\widetilde{G}^{\mathrm{qry}}_{\Gamma}[\gamma^{\mathrm{ext}}]$, because $\widetilde{G}^{\mathrm{qry}}_{\Gamma}[\gamma^{\mathrm{ext}}]$ is a subgraph of $\hat{G}^{\mathrm{qry}}_{\Gamma}[\gamma^{\mathrm{ext}}]$ by definition.

From now we focus on the other direction. It suffices to show that, for each edge $\hat{e} \in E(\hat{G}^{\mathrm{qry}}_{\Gamma}[\gamma^{\mathrm{ext}}]) = E(\hat{G}_{\gamma^{\mathrm{ext}}}) \setminus \hat{E}_{\Gamma,\mathrm{aff}}$ with endpoints $u, v \notin F$, we have u and v are connected in $\widetilde{G}^{\mathrm{qry}}_{\Gamma}[\gamma^{\mathrm{ext}}] \setminus F = (\widetilde{G}_{\gamma^{\mathrm{ext}}} \setminus \hat{E}_{\Gamma,\mathrm{aff}}) \setminus F$. First, if $\{u,v\} \in E(\widetilde{G}^{\mathrm{sp}}_{\gamma^{\mathrm{ext}}})$, then trivially $e \in E(\widetilde{G}_{\gamma^{\mathrm{ext}}})$ by the construction, which immediately gives u and v are connected in $(\widetilde{G}_{\gamma^{\mathrm{ext}}} \setminus \hat{E}_{\Gamma,\mathrm{aff}}) \setminus F$.

Hence, we assume u and v are not adjacent in $\widetilde{G}^{\mathrm{sp}}_{\gamma^{\mathrm{ext}}}$ from now. Recall the construction of shortcut graphs, for each $\gamma' \prec \gamma$, the shortcut edges with type γ' must have their endpoints in $\hat{N}(\Gamma')$, and we know that $|\hat{N}(\Gamma')| \leq h \cdot \lambda_{\mathrm{nb}}$. Let

$$V_F = ((F \cup \bigcup_{\text{affected } \gamma'} \hat{N}(\Gamma')) \cap \gamma^{\mathrm{ext}}) \setminus \{u,v\}.$$

That is, V_F collects all γ^{ext}-vertices which are failed or incident to some affected edges, excluding u and v. We have that $|V_F| \leq f + h^2(f+2)\lambda_{\mathrm{nb}}$, because the number of affected γ' is at most $h(f+2)$ by Observation 3.10.

Let $\widetilde{G}^{\mathrm{sp}}_{\gamma^{\mathrm{ext}},\mathrm{valid}}$ be the corresponding simple graph of $(\widetilde{G}_{\gamma^{\mathrm{ext}}} \setminus \hat{E}_{\Gamma,\mathrm{aff}}) \setminus F$. Because $\hat{e} \in E(\hat{G}_{\gamma^{\mathrm{ext}}})$, we have $\{u,v\} \in E(\hat{G}^{\mathrm{sp}}_{\gamma^{\mathrm{ext}}})$, and furthermore $\{u,v\} \in E(\hat{G}^{\mathrm{sp}}_{\gamma^{\mathrm{ext}}} \setminus V_F)$ because $u, v \notin V_F$. This means u and v are connected in $\widetilde{G}^{\mathrm{sp}}_{\gamma^{\mathrm{ext}}} \setminus V_F$ by Lemma 3.15 and the fact that $|V_F| \leq d-1$. Finally, by Claim 3.17, u and v are connected in $\widetilde{G}^{\mathrm{sp}}_{\gamma^{\mathrm{ext}},\mathrm{valid}}$, so they are also connected in $(\widetilde{G}_{\gamma^{\mathrm{ext}}} \setminus \hat{E}_{\Gamma,\mathrm{aff}}) \setminus F$ as desired.

Claim 3.17. *$\widetilde{G}^{\mathrm{sp}}_{\gamma^{\mathrm{ext}}} \setminus V_F$ is a subgraph of $\widetilde{G}^{\mathrm{sp}}_{\gamma^{\mathrm{ext}},\mathrm{valid}}$.*

Proof. Consider an edge $\{x,y\} \in E(\widetilde{G}^{\mathrm{sp}}_{\gamma^{\mathrm{ext}}} \setminus V_F)$, and we will show that $\{x,y\} \in E(\widetilde{G}^{\mathrm{sp}}_{\gamma^{\mathrm{ext}},\mathrm{valid}})$. Because we have assumed that u and v are not adjacent in $\widetilde{G}^{\mathrm{sp}}_{\gamma^{\mathrm{ext}}}$, we have either x or y is not in $\{u,v\}$, say $x \notin \{u,v\}$. From the construction of $\widetilde{G}_{\gamma^{\mathrm{ext}}}$, there must be an edge $\widetilde{e} \in E(\widetilde{G}_{\gamma^{\mathrm{ext}}})$ connecting x, y.

We now show that $\widetilde{e}$ is an unaffected edge. Assume for contradiction that $\widetilde{e}$ is affected. Let γ' be the type of $\widetilde{e}$. Then γ' is affected, and $x \in \hat{N}_{\Gamma'}$. Because we also have $x \notin \{u,v\}$ and $V(\widetilde{G}^{\mathrm{sp}}_{\gamma^{\mathrm{ext}}}) = \gamma^{\mathrm{ext}}$, we know $x \in (\hat{N}_{\Gamma'} \cap \gamma^{\mathrm{ext}}) \setminus \{u,v\} \subseteq V_F$, a contradiction.

Now because $\widetilde{e} \in E(\widetilde{G}_{\gamma^{\mathrm{ext}}})$ is an unaffected edge connecting $x, y \notin F$ (because $F \subseteq V_F$), $\widetilde{e}$ shows up in $(\widetilde{G}_{\gamma^{\mathrm{ext}}} \setminus \hat{E}_{\Gamma,\mathrm{aff}}) \setminus F$, and $\{x,y\} \in E(\widetilde{G}^{\mathrm{sp}}_{\gamma^{\mathrm{ext}},\mathrm{valid}})$. □

□

Corollary 3.18. *Let $\Gamma \in \mathcal{C}$ be an affected component. Two vertices $x, y \in Q^{\mathrm{ext}}_{\Gamma} \setminus F$ are connected in $\hat{G}^{\mathrm{qry}}_{\Gamma} \setminus F$ if and only if they are connected in $\widetilde{G}^{\mathrm{qry}}_{\Gamma} \setminus F$.*

Proof. Note that the edge set $E(\hat{G}_{\Gamma}) \setminus E(\hat{G}_{\Gamma}[\gamma^{\mathrm{ext}}])$ is exactly the same as $E(\widetilde{G}_{\Gamma}) \setminus E(\widetilde{G}_{\Gamma}[\gamma^{\mathrm{ext}}])$. This means $E(\hat{G}^{\mathrm{qry}}_{\Gamma}) \setminus E(\hat{G}^{\mathrm{qry}}_{\Gamma}[\gamma^{\mathrm{ext}}])$ is exactly the same as $E(\widetilde{G}^{\mathrm{qry}}_{\Gamma}) \setminus E(\widetilde{G}^{\mathrm{qry}}_{\Gamma}[\gamma^{\mathrm{ext}}])$, because $\hat{G}^{\mathrm{qry}}_{\Gamma}, \widetilde{G}^{\mathrm{qry}}_{\Gamma}$ are just constructed from $\hat{G}_{\Gamma}, \widetilde{G}_{\Gamma}$ by removing affected edges and taking the restriction on vertices $Q^{\mathrm{ext}}_{\Gamma}$. Combining Lemma 3.16, we get this corollary immediately. □

Combining Lemma 3.11 and Corollary 3.18, we get Lemma 3.14 immediately.

3.3 The Strategy for Handling Queries

In this section, we will describe the query algorithm, but we will assume some interfaces along the way. The labeling scheme for these interfaces will be deferred to the next subsection.

As we discussed above, the query algorithm will solve the connectivity of $Q_\Gamma \setminus F$ on $G[\Gamma] \setminus F$ for all affected components Γ in a bottom-up order. In fact, here we already need an interface $\mathsf{ListAffectedComps}()$ which lists all the affected components in a bottom-up order.

At a particular step Γ, naturally the output is a partition $\mathcal{P}_\Gamma$ of $Q_\Gamma \setminus F$ which capture the connectivity (i.e. two vertices $x, y \in Q_\Gamma \setminus F$ are in the same group of $\mathcal{P}_\Gamma$ if and only if they are connected in $G[\Gamma] \setminus F$). However, in our implementation, we will represent $\mathcal{P}_\Gamma$ implicitly by a partition $\mathcal{K}_\Gamma$ of *subtrees* on the Steiner trees.

Definition 3.19 (Subtrees w.r.t. F)**.** For each affected core γ, we break the Steiner tree T_γ into *subtrees* by removing failed vertices F, and let $\mathcal{T}_\gamma$ be the set of these subtrees. Similarly, for the extended Steiner tree $T_{\gamma^{\text{ext}}}$, we define the subtrees $\mathcal{T}_{\gamma^{\text{ext}}}$ by removing *all nodes* corresponding to failed vertices. For each subtree τ, we let $V^{\text{tmn}}(\tau)$ denote the terminal nodes in τ, and without ambiguity, $V^{\text{tmn}}(\tau)$ also refers to the vertices in G corresponding to the terminal nodes in $V^{\text{tmn}}(\tau)$.

For each affected component Γ, we define

$$\mathcal{T}_\Gamma = \bigcup_{\text{affected } \gamma' \preceq \gamma} \mathcal{T}_{\gamma'}$$

Note that $\{V^{\text{tmn}}(\tau) \mid \tau \in \mathcal{T}_\Gamma\}$ partitions $Q_\Gamma \setminus F$, because $\{\gamma' \setminus F \mid \text{affected } \gamma' \preceq \gamma\}$ partitions $Q_\Gamma \setminus F$ (by Observation 3.5 and the definition of Q_Γ), and for each affected $\gamma' \preceq \gamma$, $\{V^{\text{tmn}}(\tau) \mid \tau \in \mathcal{T}_{\gamma'}\}$ partitions $\gamma' \setminus F$. Moreover, each subtree $\tau \in \mathcal{T}_\Gamma$ certifies that vertices in $V^{\text{tmn}}(\tau)$ are connected in $G[\Gamma] \setminus F$. Therefore, we can indeed represent the partition $\mathcal{P}_\Gamma$ of vertices $Q_\Gamma \setminus F$ using a partition $\mathcal{K}_\Gamma$ of subtrees $\mathcal{T}_\Gamma$. More precisely, the output of the step at Γ is a partition $\mathcal{K}_\Gamma$ of $\mathcal{T}_\Gamma$ s.t. its mapping $\mathcal{P}_\Gamma = \{\bigcup_{\tau \in K} V^{\text{tmn}}(\tau) \mid K \in \mathcal{K}_\Gamma\}$ on $Q_\Gamma \setminus F$ capture the connectivity of $Q_\Gamma \setminus F$ in $G[\Gamma] \setminus F$.

Solving New Connectivity at Γ. As shown in Lemma 3.14, the new connectivity at step Γ is the connectivity of $Q^{\text{ext}}_\Gamma \setminus F$ in $\widetilde{G}^{\text{qry}}_\Gamma \setminus F$ and the connectivity of (terminal nodes) $\gamma^{\text{ext}} \setminus F$ in $T_{\gamma^{\text{ext}}} \setminus F$. Again, we want a partition $\mathcal{P}^{\text{ext}}_\Gamma$ of $Q^{\text{ext}}_\Gamma \setminus F$ that captures the new connectivity, and in the implementation we will represent $\mathcal{P}^{\text{ext}}_\Gamma$ implicitly by a partition $\mathcal{K}^{\text{ext}}_\Gamma$ of subtrees of extended Steiner subtrees.

Formally, we define

$$\mathcal{T}^{\text{ext}}_\Gamma = \mathcal{T}_{\gamma^{\text{ext}}} \cup \bigcup_{\substack{\text{affected } \gamma' \prec \gamma \\ \text{s.t. } \gamma' \text{ is not in } \gamma^{\text{ext}}}} \mathcal{T}_{\gamma'}.$$

We also have $\{V^{\text{tmn}}(\tau) \mid \tau \in \mathcal{T}^{\text{ext}}_\Gamma\}$ partitions $Q^{\text{ext}}_\Gamma \setminus F$. Our goal is to compute a partition $\mathcal{K}^{\text{ext}}_\Gamma$ of $\mathcal{T}^{\text{ext}}_\Gamma$ with its mapping $\mathcal{P}^{\text{ext}}_\Gamma = \{\bigcup_{\tau \in K} V^{\text{tmn}}(\tau) \mid K \in \mathcal{K}^{\text{ext}}_\Gamma\}$ satisfying the following.

1. Each $u, v \in Q^{\text{ext}}_\Gamma \setminus F$ in the same part of $\mathcal{P}^{\text{ext}}_\Gamma$ are connected in $G[\Gamma] \setminus F$.
2. Each $u, v \in Q^{\text{ext}}_\Gamma \setminus F$ connected in $\widetilde{G}^{\text{qry}}_\Gamma \setminus F$ are in the same part of $\mathcal{P}^{\text{ext}}_\Gamma$.
3. Each $u, v \in \gamma^{\text{ext}} \setminus F \subseteq Q^{\text{ext}}_\Gamma \setminus F$ whose terminal nodes are connected in $T_{\gamma^{\text{ext}}} \setminus F$ are in the same part of $\mathcal{P}^{\text{ext}}_\Gamma$.

Note that requirement 3 is automatically satisfied because each pair of terminal nodes $u, v \in \gamma^{\text{ext}} \setminus F$ connected in $T_{\gamma^{\text{ext}}} \setminus F$ must belong to $V^{\text{tmn}}(\tau)$ for the same subtree $\tau \in \mathcal{T}_{\gamma^{\text{ext}}}$.

To compute a partition $\mathcal{K}_\Gamma^{\text{ext}}$ satisfying requirements 1 and 2, we will exploit the following key observation, Lemma 3.20, from the vertex expander hierarchy. Roughly speaking, those subtrees with too many γ-vertices inside and surrounding must belong to the same connected component of $G_\Gamma \setminus F$.

Before stating Lemma 3.20, we introduce some notations. For a graph G' and a vertex set $A \subseteq V(G')$, we let $N^{G'}(A)$ denote the neighbors of A in G'. For each subtree $\tau \in \mathcal{T}_\Gamma^{\text{ext}}$, let $V_\gamma^{\text{tmn}}(\tau) = V^{\text{tmn}}(\tau) \cap \gamma$ be the γ-vertices inside $V^{\text{tmn}}(\tau)$,[5] and let $N_\gamma(\tau) = N^{\widetilde{G}_\Gamma^{\text{qry}}}(V^{\text{tmn}}(\tau)) \cap \gamma$ be the intersection of γ and the neighbors of $V^{\text{tmn}}(\tau)$ in $\widetilde{G}_\Gamma^{\text{qry}}$.

Lemma 3.20. *Let $\tau_1, \tau_2 \in \mathcal{T}_\Gamma^{\text{ext}}$ be two subtrees such that $|N_\gamma(\tau_1)| + |V_\gamma^{\text{tmn}}(\tau_1)| > f/\phi$ and $|N_\gamma(\tau_2)| + |V_\gamma^{\text{tmn}}(\tau_2)| > f/\phi$. Then $V^{\text{tmn}}(\tau_1)$ and $V^{\text{tmn}}(\tau_2)$ are contained in the same connected component of $G[\Gamma] \setminus F$.*

Proof. Assume for contradiction that τ_1 and τ_2 belong to two different connected components C_1 and C_2 of $G[\Gamma] \setminus F$. It must be that $N^{G[\Gamma]}(C_i) \subseteq F$ for $i \in \{1, 2\}$.

We clearly have $V_\gamma^{\text{tmn}}(\tau_1) \cup N_\gamma(\tau_1) \subseteq \left(C_1 \cup N^{G[\Gamma]}(C_1)\right) \cap \gamma$. $V_\gamma^{\text{tmn}}(\tau_1)$ is contained in $C_1 \cap \gamma$ by assumption. A $y \in N_\gamma(\tau_1)$ is joined by an edge $\{x, y\} \in E(\widetilde{G}_\Gamma^{\text{qry}})$ s.t. $x \in \tau_1$. This edge can have type *original*, or type γ' for some unaffected $\gamma' \preceq \gamma$. Either way, there is a path from x to y whose internal vertices are in Γ'. Note that Γ' is disjoint from F, so $y \in C_1 \cup N^{G[\Gamma]}(C_1)$. We have shown that for $i \in \{1, 2\}$,

$$\left|(C_i \cup N^{G[\Gamma]}(C_i)) \cap \gamma\right| \geq \left|V^{\text{tmn}}(\tau_i)\right| + |N_\gamma(\tau_i)| > f/\phi.$$

Because γ is ϕ-vertex-expanding in $G[\Gamma]$, we have

$$\left|N^{G[\Gamma]}(C_1)\right| \geq \phi \cdot \min\left\{\left|(C_1 \cup N^{G[\Gamma]}(C_1)) \cap \gamma\right|, |(\Gamma \setminus C_1) \cap \gamma|\right\} > \phi \cdot (f/\phi) = f.$$

However, $|N^{G[\Gamma]}(C_1)| > f$ contradicts the fact that $N^{G[\Gamma]}(C_1) \subseteq F$. □

We say a subtree $\tau \in \mathcal{T}_\Gamma^{\text{ext}}$ is *giant* if $|V_\gamma^{\text{tmn}}(\tau)| + |N_\gamma(\tau)| > f/\phi$, otherwise it is *non-giant*. We start from a trivial partition of $\mathcal{T}_\Gamma^{\text{ext}}$ in which each subtree forms a singleton group, and then merge the groups according to the following rules.

R1. Put all giant subtrees into the same group, called the *giant group*. Whenever a connected group $\{\tau_i\}$ of subtrees collectively has $|V_\gamma^{\text{tmn}}(\{\tau_i\})| + |N_\gamma(\{\tau_i\})| > f/\phi$, merge it with the giant group.

R2. Let $\tau_x \in \mathcal{T}_\Gamma^{\text{ext}}$ be non-giant and $\tau_y \in \mathcal{T}_\Gamma^{\text{ext}}$ be such that $N_\gamma(\tau_x)$ intersects $V^{\text{tmn}}(\tau_y)$. Then merge the groups containing τ_x and τ_y.

R3. Let $\tau_y \in \mathcal{T}_\Gamma^{\text{ext}}$ be non-giant and $\tau_x \in \mathcal{T}_\Gamma^{\text{ext}}$ be giant such that $N_\gamma(\tau_x)$ intersects $V^{\text{tmn}}(\tau_y)$. Then merge τ_y with the giant group.

[5] In fact, $V_\gamma^{\text{tmn}}(\tau)$ is not empty only when $\tau \in \mathcal{T}_{\gamma^{\text{ext}}}$.

Before we move on to discuss the implementation of these rules, we first show that the partition $\mathcal{K}_\Gamma^{\text{ext}}$ generated as above will satisfy requirements 1 and 2. The requirement 1 is satisfied because the rules are using valid connectivity in $G[\Gamma] \setminus F$. In particular, **R1** is safe according to Lemma 3.20. For each of **R2** and **R3**, $N_\gamma(\tau_x)$ intersects $V^{\text{tmn}}(\tau_y)$ means there is an edge in $E(\widetilde{G}_\Gamma^{\text{qry}})$ connecting $x \in V^{\text{tmn}}(\tau_x)$ and $y \in V^{\text{tmn}}(\tau_y)$, which means x and y are connected in $G[\Gamma] \setminus F$ because this edge is unaffected and $x, y \notin F$ (note that $V^{\text{tmn}}(\tau_x)$ and $V^{\text{tmn}}(\tau_y)$ are disjoint from F). To show that requirement 2 is satisfied, we will show that for each edge $\{x, y\} \in E(\widetilde{G}_\Gamma^{\text{qry}} \setminus F)$ connecting $V^{\text{tmn}}(\tau_1)$ and $V^{\text{tmn}}(\tau_2)$, the subtrees τ_1 and τ_2 will be merged by one of the rules. Recall that the edges in graph $\hat{G}_\Gamma^{\text{qry}}$ must be incident to γ, and $\widetilde{G}_\Gamma^{\text{qry}}$ is a subgraph of $\hat{G}_\Gamma^{\text{qry}}$. Hence we have either $x \in \gamma$ or $y \in \gamma$, and without loss of generality, we assume $y \in \gamma$.

- When both τ_1 and τ_2 are giant they will be merged by **R1**.
- When both τ_1 and τ_2 are non-giant they will be merged by **R2**.
- When τ_1 is non-giant and τ_2 is giant, they will be merged by either **R2** or **R3** (depending on whether $y \in V^{\text{tmn}}(\tau_2)$ or $y \in V^{\text{tmn}}(\tau_1)$).

The following primitives will allow us to implement rules **R1**, **R2**, and **R3**. Here Γ is an affected component and $\tau \in \mathcal{T}_\Gamma^{\text{ext}}$.

$\mathsf{ListSubtrees}^{\text{ext}}(\Gamma)$**:** List all subtrees of $\mathcal{T}_\Gamma^{\text{ext}}$.

$\mathsf{ListSubtrees}(\Gamma)$**:** List all subtrees of $\mathcal{T}_\Gamma$.

$\mathsf{ListTerminals}(\tau, \Gamma)$**:** Return up to $f/\phi + 1$ elements of $V_\gamma^{\text{tmn}}(\tau)$.

$\mathsf{ListNeighbors}(\tau, \Gamma)$**:** Return up to $f/\phi + 1$ elements of $N_\gamma(\tau)$.

$\mathsf{IsTerminal}(v, \tau)$**:** Return true iff v is a terminal in τ.

$\mathsf{PickTerminal}(\tau)$**:** Return any element of $V^{\text{tmn}}(\tau)$.

$\mathsf{EnumFromGiant}(\tau, \Gamma)$**:** Requirement: $\tau \in \mathcal{T}_{\gamma^{\text{ext}}}$, and τ is non-giant. Return the *number* of edges in $E(\widetilde{G}_\Gamma^{\text{qry}} \setminus F)$ joining $V_\gamma^{\text{tmn}}(\tau)$ and $\bigcup_{\text{giant } \tau_x \in \mathcal{T}_\Gamma^{\text{ext}}} V^{\text{tmn}}(\tau_x)$.

Clearly $\mathsf{ListSubtrees}^{\text{ext}}(\Gamma)$, $\mathsf{ListTerminals}(\tau, \Gamma)$, and $\mathsf{ListNeighbors}(\tau, \Gamma)$ can be used to list all subtrees and determine which are giant. This suffices to implement rule **R1**. $\mathsf{ListNeighbors}$ and $\mathsf{IsTerminal}$ suffice to implement rule **R2**. We implement rule **R3** *non-constructively.* In other words, we do not find a *specific* giant τ_x such that $N_\gamma(\tau_x) \cap V^{\text{tmn}}(\tau_y) \neq \emptyset$, but merely infer that there exists such a τ_x, if $\mathsf{EnumFromGiant}(\tau_y, \Gamma) > 0$. Observe that the condition of rule **R3** is $\tau_y \in \mathcal{T}_\Gamma^{\text{ext}}$. If $\tau_y \notin \mathcal{T}_{\gamma^{\text{ext}}}$ then we cannot call $\mathsf{EnumFromGiant}$, but in this case $\tau_y \in \mathcal{T}_\Gamma^{\text{ext}} \setminus \mathcal{T}_{\gamma^{\text{ext}}}$ implies $V_\gamma^{\text{tmn}}(\tau_y) = \emptyset$, so **R3** cannot be applied anyway.

Obtain $\mathcal{K}_\Gamma$ from $\mathcal{K}_\Gamma^{\text{ext}}$ and all $\mathcal{K}_{\Gamma_{\text{child}}}$. Next, we discuss how to compute $\mathcal{K}_\Gamma$ for a particular affected component Γ, assuming we have already got the $\mathcal{K}_\Gamma^{\text{ext}}$ above (capturing the new connectivity at Γ) and the $\mathcal{K}_{\Gamma_{\text{child}}}$ of all affected children Γ_{child} of Γ.

We call $\mathsf{ListSubtrees}(\Gamma)$ and form their initialize partition $\mathcal{K}_\Gamma$ as:

$$\mathcal{K}_\Gamma^0 = \{\{\tau\} \mid \tau \in \mathcal{T}_\gamma\} \cup \bigcup_{\Gamma\text{'s affected children } \Gamma_{\text{child}}} \mathcal{K}_{\Gamma_{\text{child}}}.$$

That is, any subtrees grouped in $\mathcal{K}_{\Gamma_{\text{child}}}$ are initially grouped in $\mathcal{K}_\Gamma^0$. It remains to group subtrees according to the connectivity learned from $\mathcal{K}_\Gamma^{\text{ext}}$.

For each group $K^{\text{ext}} \in \mathcal{K}_\Gamma^{\text{ext}}$, we merge all the groups $K^0 \in \mathcal{K}_\Gamma^0$ such that $V^{\text{tmn}}(K^0)$ intersects $V^{\text{tmn}}(K^{\text{ext}})$. By Observation 3.9, $V^{\text{tmn}}(\tau)$ is either contained in or disjoint from $V^{\text{tmn}}(\tau^{\text{ext}})$, for $\tau \in K^0 \in \mathcal{K}_\Gamma^0$ and $\tau^{\text{ext}} \in K^{\text{ext}} \in \mathcal{K}_\Gamma^{\text{ext}}$. Thus we can detect whether they intersect by calling $v \leftarrow \mathsf{PickTerminal}(\tau)$ and $\mathsf{IsTerminal}(v, \tau^{\text{ext}})$. After all grouping, the final partition is $\mathcal{K}_\Gamma$.

Answering Query $\langle s, t, F\rangle$. We compute the partitions $\{\mathcal{K}_\Gamma\}$ for all affected Γ in postorder, culminating in the root partition $\mathcal{K}_{\Gamma_{\text{root}}}$. Then s and t are connected iff there exists a $K \in \mathcal{K}_{\Gamma_{\text{root}}}$ and $\tau_s, \tau_t \in K$ such that $\mathsf{IsTerminal}(s, \tau_s) = \mathsf{IsTerminal}(t, \tau_t) = \mathsf{true}$.

3.4 The Labeling Scheme: Implementing the Strategy

Finally, we describe a labeling scheme that supports the interfaces required by the query algorithm. Note that we still fix an $S_i \in \mathcal{S}$ and assume that vertices in S_i will never fail, and the labeling scheme we describe is only for this S_i. At the very end, we will discuss our final labeling scheme and how to find a non-failed S_i for a query.

We restate the interfaces in a formal way as follows.

- $\mathsf{ListAffectedComps}()$ outputs the identifiers of affected components in a bottom-up order.
- $\mathsf{ListSubtrees}(\Gamma), \mathsf{ListSubtrees}^{\text{ext}}(\Gamma)$ receive the identifier of an affected component Γ, and list the profiles of subtrees in $\mathcal{T}_\Gamma$ and $\mathcal{T}_\Gamma^{\text{ext}}$ respectively.
- $\mathsf{ListTerminals}(\tau, \Gamma)$ receives the identifier of an affected component Γ and the profile of a subtree $\tau \in \mathcal{T}_\Gamma^{\text{ext}}$, and either detects $|V_\gamma^{\text{tmn}}(\tau)| > f/\phi$ or outputs the profiles of all vertices in $V_\gamma^{\text{tmn}}(\tau)$.
- $\mathsf{ListNeighbors}(\tau, \Gamma)$ receives the identifier of an affected component Γ and the profile of a subtree $\tau \in \mathcal{T}_\Gamma^{\text{ext}}$, and either detects $|V_\gamma^{\text{tmn}}(\tau)| + |N_\gamma(\tau)| > f/\phi$ or outputs the profiles of all vertices in $N_\gamma(\tau)$.
- $\mathsf{IsTerminal}(v, \tau)$ receives the profile of a vertex v and the profile of a subtree τ (an arbitrary one from Definition 3.19), and outputs whether v is in $V^{\text{tmn}}(\tau)$ or not.
- $\mathsf{PickTerminal}(\tau)$ receives the profile of a subtree $\tau \in \mathcal{T}_\gamma$ of some affected γ, and outputs the profile of an arbitrary vertex in $V^{\text{tmn}}(\tau)$.
- $\mathsf{EnumFromGiant}(\tau_y, \Gamma)$ receives the profile of a non-giant subtree $\tau_y \in \mathcal{T}_{\gamma^{\text{ext}}}$, and outputs the number of edges in $E(\widetilde{G}_\Gamma^{\text{qry}} \setminus F)$ connecting $\bigcup_{\text{giant } \tau_x \in \mathcal{T}_\Gamma^{\text{ext}}} V^{\text{tmn}}(\tau_x)$ and $V_\gamma^{\text{tmn}}(\tau_y)$.

We make some remarks on the terms *identifiers* and *profiles*. For those objects defined in the preprocessing phase (i.e. vertices in G, components/cores[6], Steiner trees and extended Steiner

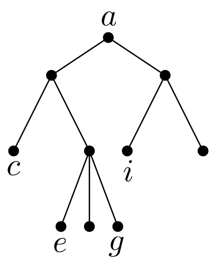
[6]A component Γ and its core γ has the same identifier.

trees), we assign each of them a distinct $O(\log n)$-bit integer as its identifier, denoted by $\mathsf{id}(\cdot)$. Note that $O(\log n)$ bits suffice because the number of such objects is polynomial. During the query phase, we may further generate *profiles* for the objects (including those defined in the query phase, e.g. subtrees from Definition 3.19). A profile is something that representing this object and generally it is not only a single integer.

3.4.1 The Euler Tours of (Extended) Steiner Trees

Like what we did for the edge fault connectivity labels, we will exploit the Euler tour to "linearize" the (extended) Steiner trees. However, the Euler tours here are slightly different from the definition in Section 2.

Definition 3.21 (Euler Tours)**.** For each (extended) Steiner tree T, we define $\mathsf{Euler}(T)$ to be its Euler tour, which is a list that includes all *occurrence* of nodes according to a DFS traversal of T, starting from an arbitrary root node. For convenience, we add two *virtual occurences* $\mathsf{start}(T)$ and $\mathsf{end}(T)$ at the front and the end of $\mathsf{Euler}(T)$ as the "guards". See Figure 5 for a small example.



$$\mathsf{Euler}(T) = \Big(\mathsf{start}(T), a, b, c, b, d, e, d, f, d, g, d, b, a, h, i, h, j, h, a, \mathsf{end}(T)\Big)$$

Figure 5: Left: an (extended) Steiner tree T, rooted at a. Right: $\mathsf{Euler}(T)$.

For each occurrence $v_{\text{oc}} \in \mathsf{Euler}(T)$, we say a node $v_{\text{nd}} \in V(T)$ *owns* v_{oc} if v_{oc} is an occurrence of v_{nd}, and naturally the vertex that *owns* v_{oc} is the the vertex in G corresponding to the node in $V(T)$ that owns v_{oc}. We let $\mathsf{pos}(v_{\text{oc}})$ denote the position of v_{oc} in $\mathsf{Euler}(T)$. In particular, $\mathsf{pos}(\mathsf{start}(T)) = 0$ and $\mathsf{pos}(\mathsf{end}(T)) = 2|V(T)|$, since the number of non-virtual occurrences in $\mathsf{Euler}(T)$ is $2|V(T)|-1$. The profile of an occurrence $v_{\text{oc}} \in \mathsf{Euler}(T)$ is defined as $\mathsf{profile}(v_{\text{oc}}) = (\mathsf{id}(v), \mathsf{id}(T), \mathsf{pos}(v_{\text{oc}}))$, where v is the vertex that owns v_{oc}. Exceptionally, $\mathsf{profile}(\mathsf{start}(T)) = (\perp, \mathsf{id}(T), \mathsf{pos}(\mathsf{start}(T)))$ and $\mathsf{profile}(\mathsf{end}(T)) = (\perp, \mathsf{id}(T), \mathsf{pos}(\mathsf{end}(T)))$.

Label $\mathsf{occurrences}(v, T)$. For each vertex $v \notin S$ and each (extended) Steiner tree $T \in \{T_\gamma \mid v \in V(T_\gamma)\} \cup \{T_{\gamma^{\text{ext}}} \mid v \in V(T_{\gamma^{\text{ext}}})\}$, the label $\mathsf{occurrences}(v, T)$ stores the profiles of occurrences in $\mathsf{Euler}(T)$ owned by v, i.e. $\mathsf{occurrences}(v, T) = (\mathsf{profile}(v_{\text{oc},1}), \mathsf{profile}(v_{\text{oc},2}), ...)$, where $v_{\text{oc},1}, v_{\text{oc},2}, \cdots \in \mathsf{Euler}(T)$ are owned by v.

3.4.2 Profiles of Vertices, Components and Subtrees

We define the profiles of vertices, components and subtrees, and use them to implement interfaces $\mathsf{ListAffectedComps}()$, $\mathsf{ListSubtrees}(\Gamma)$, $\mathsf{ListSubtrees}^{\text{ext}}(\Gamma)$ and $\mathsf{IsTerminal}(v, \tau)$.

Label $\mathsf{profile}(\Gamma)$. The profile of a component $\Gamma \in \mathcal{C}$, denoted by $\mathsf{profile}(\Gamma)$, is constructed as follows, and we will store $\mathsf{profile}(\Gamma)$ at each vertex $v \in \Gamma$.

- Store $\mathsf{id}(\Gamma)$, $\mathsf{id}(T_\gamma)$ and $\mathsf{id}(T_{\gamma^{\text{ext}}})$.

- For each ancestor $\hat{\Gamma}$ s.t. $\Gamma \prec \hat{\Gamma}$, store $\mathsf{id}(\hat{\Gamma})$ and a one-bit indicator indicating whether $N_{\hat{\gamma}}(\Gamma)$ intersects S.

Label $\mathsf{profile}(v)$. The profile of a vertex $v \in V(G)$, denoted by $\mathsf{profile}(v)$, is constructed as follows, and we will store $\mathsf{profile}(v)$ at vertex v.

- Store $\mathsf{id}(v)$.
- For each (extended) Steiner tree $T \in \{T_{\gamma_v}\} \cup \{T_{\gamma^{\text{ext}}} \mid v \in \gamma^{\text{ext}}\}$ (i.e. v has a corresponding terminal node in T), let $\pi(v, T)$ denote the occurrence $\mathsf{Euler}(T)$ owned by the terminal node (corresponding to the vertex) v with the least $\mathsf{pos}(v_{\text{oc}})$[7], called the *principal occurrence* of the vertex v on T, and we store $\mathsf{profile}(\pi(v, T))$.

Implement $\mathsf{ListAffectedComps}()$. Note that we can already implement the interface $\mathsf{ListAffectedComps}()$ by inspecting the profiles of components storing at vertices $\{s, t\} \cup F$. In fact, we can even support the basic operations in Observation 3.22.

Observation 3.22. *By inspecting the profiles of components storing at vertices $\{s, t\} \cup F$, we can suport the following.*

- *List* $\mathsf{id}(\Gamma)$ *of all affected components* Γ.
- *Given* $\mathsf{id}(\Gamma)$ *of an affected component* Γ*, get* $\mathsf{id}(T_\gamma)$ *and* $\mathsf{id}(T_{\gamma^{\text{ext}}})$.
- *Given* $\mathsf{id}(T)$ *of an (extended) Steiner tree of some (unknown) affected component* Γ*, decide whether it is a Steiner tree or an extended Steiner tree and get* $\mathsf{id}(\Gamma)$.
- *Given* $\mathsf{id}(\Gamma_1), \mathsf{id}(\Gamma_2)$ *of two affected components* Γ_1, Γ_2*, distinguish among (1)* $\Gamma_1 \preceq \Gamma_2$*, (2)* $\Gamma_2 \preceq \Gamma_1$*, and (3) no ancestral relation between* Γ_1 *and* Γ_2.
- *Given* $\mathsf{id}(\Gamma'), \mathsf{id}(\Gamma)$ *of two affected components* Γ', Γ *s.t.* $\Gamma' \prec \Gamma$*, decide if* $\gamma' \subseteq \gamma^{\text{ext}}$.

Profiles of Subtrees. For an (extended) Steiner tree $T \in \{T_\gamma, T_{\gamma^{\text{ext}}} \mid \gamma \text{ is affected}\}$, we will represent a subtree $\tau \in \mathcal{T}$ ($\mathcal{T}$ is the set of subtrees of T w.r.t. F from Definition 3.19) by a collection of *intervals* on $\mathsf{Euler}(T)$.

Definition 3.23 (Intervals)**.** For each (extended) Steiner tree $T \in \{T_\gamma, T_{\gamma^{\text{ext}}} \mid \gamma \text{ is affected}\}$, we break $\mathsf{Euler}(T)$ into *intervals* by removing $\mathsf{start}(T)$, $\mathsf{end}(T)$ and all occurrences owned by failed vertices, and let $\mathcal{I}_T$ denote these intervals. For each interval I, we use $\ell_{\text{oc},I}$ and $r_{\text{oc},I}$ to denote the left and right *outer endpoints* of I. To be precise, for an interval including occurrences from position a to b, the outer endpoints of I are the occurrences at position $a-1$ and $b+1$. The profile of an interval $I \in \mathcal{I}_T$ is $\mathsf{profile}(I) = (\mathsf{profile}(\ell_{\text{oc},I}), \mathsf{profile}(r_{\text{oc},I}))$. Furthermore, we define $V^{\text{tmn}}(I)$ to be the set of vertices with its principal occurrences $\pi(v, T)$ falling in I.

Observation 3.24. *There is an assignment that assigns a subset of interval* $\mathcal{I}_\tau \subseteq \mathcal{I}_T$ *to each subtree* $\tau \in \mathcal{T}$*, satisfying the following.*

- *For each subtree* $\tau \in \mathcal{T}$*,* $V^{\text{tmn}}(\tau) = \bigcup_{I \in \mathcal{I}_\tau} V^{\text{tmn}}(I)$.

[7]In fact, we can fix $\pi(v, T)$ to be an arbitrary occurrence owned by the terminal node v (no need to be the one with the least $\mathsf{pos}(v_{\text{oc}})$).

- $\{\mathcal{I}_\tau \mid \tau \in \mathcal{T}\}$ *forms a partition of* $\mathcal{I}_T$.

We state the relations between intervals $\mathcal{I}_T$ and subtrees $\mathcal{T}$ in Observation 3.24. These are well-known facts from the nature of DFS traversals. Furthermore, by inspecting only $\mathsf{profile}(v)$ for vertices $v \in F$, we can obtain $\mathsf{profile}(I)$ for all $I \in \mathcal{I}_T$ explicitly. Furthermore, it is an easy exercise to compute such a partition $\{\mathcal{I}_\tau \mid \tau \in \mathcal{T}\}$ of $\mathcal{I}_T$ with properties in Observation 3.24 (roughly speaking, we can just simulate the DFS traversal using a stack).

Finally, for each subtree $\tau \in \mathcal{T}$, we define its profile to be $\mathsf{profile}(\tau) = \{\mathsf{profile}(I) \mid I \in \mathcal{I}_\tau\}$, i.e. we collect $\mathsf{profile}(I)$ of all $I \in \mathcal{I}_\tau$ into $\mathsf{profile}(\tau)$.

Implement $\mathsf{ListSubtrees}(\Gamma)$ and $\mathsf{ListSubtrees}^{\text{ext}}(\Gamma)$. We take $\mathsf{ListSubtrees}(\Gamma)$ as an example, and we can implement $\mathsf{ListSubtrees}^{\text{ext}}(\Gamma)$ in a similar way. We have already computed the profiles of all subtrees in Definition 3.19. We just scan all subtrees τ and output $\mathsf{profile}(\tau)$ if $\tau \in \mathcal{T}_\Gamma$. To check whether $\tau \in \mathcal{T}_\Gamma$, recall that $\mathcal{T}_\Gamma$ collect subtrees from Steiner tree $T_{\gamma'}$ for all affected $\Gamma' \preceq \Gamma$. Note that from $\mathsf{profile}(\tau)$, we can obtain $\mathsf{id}(T)$ of the (extended) Steiner tree T s.t. $\tau \in \mathcal{T}$, so using Observation 3.22, we can get the $\mathsf{id}(\Gamma')$ of the affected Γ' s.t. $T_{\gamma'} = T$ (or know such Γ' does not exist), and check whether $\Gamma' \preceq \Gamma$.

Implement $\mathsf{IsTerminal}(v, \tau)$. Note that $V^{\text{tmn}}(\tau) = \bigcup_{I \in \mathcal{I}_\tau} V^{\text{tmn}(I)}$ by Observation 3.24, so to check whether $v \in V^{\text{tmn}}(\tau)$, it suffices to check whether $v \in V^{\text{tmn}}(I)$ for each $I \in \mathcal{I}_\tau$. From $\mathsf{profile}(I)$ (stored in $\mathsf{profile}(\tau)$), we can get $\mathsf{id}(T)$ of the (extended) Steiner tree T s.t. I is on $\mathsf{Euler}(T)$, and further get the position of the outer endpoints of I, i.e. $\mathsf{pos}(\ell_{\text{oc},I})$ and $\mathsf{pos}(r_{\text{oc},I})$. Lastly, we look at $\mathsf{profile}(\pi(v, T))$ (stored in $\mathsf{profile}(v)$) of the principal occurrence of v on T, and check if $\mathsf{pos}(\ell_{\text{oc},I}) < \mathsf{pos}(\pi(v, T)) < \mathsf{pos}(r_{\text{oc},I})$.

3.4.3 Labels on Euler Tours

We introduce some labels related to Euler tours, and then use them to implement the interfaces $\mathsf{PickTerminal}(\tau)$, $\mathsf{ListTerminals}(\tau, \Gamma)$ and $\mathsf{ListNeighbors}(\tau, \Gamma)$.

Label $\mathsf{SuccTerminal}(v_{\text{oc}}, T)$. For each component $\Gamma \in \mathcal{C}$ and each occurrence $v_{\text{oc}} \in \mathsf{Euler}(T_\gamma)$, we construct a label $\mathsf{SuccTerminal}(v_{\text{oc}}, T_\gamma)$. Let $x \in \gamma$ be the vertex whose principal occurrence $\pi(x, T_\gamma)$ is to the right of v_{oc} and has the least $\mathsf{pos}(\pi(x, T_\gamma))$. We store $\mathsf{profile}(x)$ in the label $\mathsf{SuccTerminal}(v_{\text{oc}}, T_\gamma)$, and without ambiguity, $\mathsf{SuccTerminal}(v_{\text{oc}}, T_\gamma)$ also refers to this vertex x in our analysis. We store the label $\mathsf{SuccTerminal}(v_{\text{oc}}, T_\gamma)$ at the vertex v owning v_{oc}. Exceptionally, when v_{oc} is $\mathsf{start}(T_\gamma)$ or $\mathsf{end}(T_\gamma)$, we store $\mathsf{SuccTerminal}(v_{\text{oc}}, T_\gamma)$ at all vertices inside Γ.

Implement $\mathsf{PickTerminal}(\tau)$. Note that the input guarantees that $\tau \in \mathcal{T}_\gamma$ for some affected γ. By Observation 3.24, we have $V^{\text{tmn}}(\tau) = \bigcup_{I \in \mathcal{I}_\tau} V^{\text{tmn}}(I)$, so it suffices to obtain a vertex of $V^{\text{tmn}}(I)$ of some $I \in \mathcal{I}_\tau$. For a particular $I \in \mathcal{I}_\tau$, if $V^{\text{tmn}}(I)$ is not empty, the vertex $x = \mathsf{SuccTerminal}(\ell_{\text{oc},I})$ must be inside $V^{\text{tmn}}(I)$ by definition. Hence, we just need to check whether $x \in V^{\text{tmn}}(I)$, or equivalently, whether $\mathsf{pos}(\ell_{\text{oc},I}) < \mathsf{pos}(\pi(x, T_\gamma)) < \mathsf{pos}(r_{\text{oc},I})$. Indeed, this is doable because we can obtain $\mathsf{pos}(\ell_{\text{oc},I}), \mathsf{pos}(r_{\text{oc},I}), \mathsf{id}(T_\gamma)$ from $\mathsf{profile}(\tau)$ and obtain $\mathsf{pos}(\pi(x, T_\gamma))$ from $\mathsf{profile}(x)$ ($\mathsf{profile}(x)$ is obtained from the label $\mathsf{SuccTerminal}(\ell_{\text{oc},I}, T_\gamma)$ by Observation 3.25).

Observation 3.25. *Given* $\mathsf{profile}(I)$ *of some interval* $I \in \mathcal{I}_\tau$ *of some* $\tau \in \mathcal{T}_\gamma$, *we can access* $\mathsf{SuccTerminal}(\ell_{\text{oc},I}, T_\gamma)$.

Proof. We can obtain $\mathsf{profile}(\ell_{\mathrm{oc},I})$ from $\mathsf{profile}(I)$. If $\ell_{\mathrm{oc},I}$ is owned by some vertex v (i.e. $\ell_{\mathrm{oc},I}$ is not $\mathsf{start}(T_\gamma)$ or $\mathsf{end}(T_\gamma)$), v must be inside F, so the label $\mathsf{SuccTerminal}(\ell_{\mathrm{oc},I}, T_\gamma)$ stored at v is accessible and we can locate it using $\mathsf{profile}(\ell_{\mathrm{oc},I})$. If $\ell_{\mathrm{oc},I}$ is $\mathsf{start}(T_\gamma)$ or $\mathsf{end}(T_\gamma)$, $\mathsf{SuccTerminal}(v_{\mathrm{oc}}, T_\gamma)$ is accessible from an arbitrary vertex $v \in \Gamma \cap (\{s,t\} \cup F)$. Such v exists because Γ is affected ($\mathcal{T}_\gamma$ is well-defined only when Γ is affected). $\square$

Label $\mathsf{InnerTerminals}^{\rightarrow}_{\gamma}(v_{\mathrm{oc}}, T_{\gamma^{\mathrm{ext}}})$. For each component $\Gamma \in \mathcal{C}$ and each occurrence $v_{\mathrm{oc}} \in \mathsf{Euler}(T_{\gamma^{\mathrm{ext}}})$ not owned by S-vertices, we construct the label $\mathsf{InnerTerminals}^{\rightarrow}_{\gamma}(v_{\mathrm{oc}}, T_{\gamma^{\mathrm{ext}}})$ as follows.

Let $V^{\rightarrow}_{v_{\mathrm{oc}}}$ collect all vertices $x \in \gamma$ with principal occurrence $\pi(x, T_{\gamma^{\mathrm{ext}}})$ to the right of v_{oc}. We sort vertices $x \in V^{\rightarrow}_{v_{\mathrm{oc}}}$ in order of $\pi(x, T_{\gamma^{\mathrm{ext}}})$ from leftmost to rightmost. The label $\mathsf{InnerTerminals}^{\rightarrow}_{\gamma}(v_{\mathrm{oc}}, T_{\gamma^\star})$ will take the first $f/\phi + 1$ vertices in $V^{\rightarrow}_{v_{\mathrm{oc}}}$, and for each vertex $v \in \mathsf{InnerTerminals}^{\rightarrow}_{\gamma}(v_{\mathrm{oc}}, T_{\gamma^\star})$, we store $\mathsf{profile}(v)$ in the label $\mathsf{InnerTerminals}^{\rightarrow}_{\gamma}(v_{\mathrm{oc}}, T_{\gamma^\star})$.

We will store the label $\mathsf{InnerTerminals}^{\rightarrow}_{\gamma}(v_{\mathrm{oc}}, T_{\gamma^{\mathrm{ext}}})$ at the vertex v owning v_{oc}. Exceptionally, when v_{oc} is a virtual occurrence (i.e. $\mathsf{start}(T_{\gamma^{\mathrm{ext}}})$ or $\mathsf{end}(T^{\mathrm{ext}}_{\gamma})$), we store $\mathsf{InnerTerminals}^{\rightarrow}_{\gamma}(v_{\mathrm{oc}}, T_{\gamma^{\mathrm{ext}}})$ at all vertices $v \in \Gamma$.

Implement $\mathsf{ListTerminals}(\tau, \Gamma)$. Recall that the input guarantees that $\tau \in \mathcal{T}^{\mathrm{ext}}_{\Gamma}$. By definition, $V^{\mathrm{tmn}}_{\gamma}(\tau)$ is not empty only when $\tau \in \mathcal{T}_{\gamma^{\mathrm{ext}}}$. Using $\mathsf{id}(\tau)$ and Observation 3.22, we can easily check if $\tau \in \mathcal{T}_{\gamma^{\mathrm{ext}}}$, so from now we assume $\tau \in \mathcal{T}_{\gamma^{\mathrm{ext}}}$. For each interval $I \in \mathcal{I}_\tau$, we define $V^{\mathrm{tmn}}_{\gamma}(I) = V^{\mathrm{tmn}}(I) \cap \gamma$ to be the γ-vertices in $V^{\mathrm{tmn}}(I)$. From Observation 3.24, we know $V^{\mathrm{tmn}}_{\gamma}(\tau) = \bigcup_{I \in \mathcal{I}_\tau} V^{\mathrm{tmn}}_{\gamma}(I)$. By Lemma 3.27, it suffices to inspect $\mathsf{InnerTerminals}(\ell_{\mathrm{oc},I}, T_{\gamma^{\mathrm{ext}}})$ for all $I \in \mathcal{I}_\tau$. Note that we can access $\mathsf{InnerTerminals}(\ell_{\mathrm{oc},I}, T_{\gamma^{\mathrm{ext}}})$ by Observation 3.26.

Observation 3.26. *Given* $\mathsf{profile}(\tau)$ *of some* $\tau \in \mathcal{T}_{\gamma^{\mathrm{ext}}}$*, we can access* $\mathsf{InnerTerminals}^{\rightarrow}_{\gamma}(\ell_{\mathrm{oc},I}, T_{\gamma^{\mathrm{ext}}})$ *for all* $I \in \mathcal{I}_\tau$*.*

Proof. This is similar to Observation 3.25. For each $I \in \mathcal{I}_\tau$, we can obtain $\mathsf{profile}(\ell_{\mathrm{oc},I})$ from $\mathsf{profile}(\tau)$. If $\ell_{\mathrm{oc},I}$ is owned by some vertex v, we can access $\mathsf{InnerTerminals}^{\rightarrow}_{\gamma}(\ell_{\mathrm{oc},I}, T_{\gamma^{\mathrm{ext}}})$ from the vertex v. Otherwise, $\ell_{\mathrm{oc},I}$ is $\mathsf{start}(T_\gamma)$ or $\mathsf{end}(T_\gamma)$, and we can access $\mathsf{InnerTerminals}^{\rightarrow}_{\gamma}(\ell_{\mathrm{oc},I}, T_{\gamma^{\mathrm{ext}}})$ from the vertex v from an arbitrary vertex $v \in \Gamma \cap (\{s,t\} \cup F)$ because Γ is affected ($\mathcal{T}_{\gamma^{\mathrm{ext}}}$ is well-defined only when γ is affected). $\square$

Lemma 3.27. *Given* $\mathsf{profile}(I)$ *of an interval* $I \in \mathcal{I}_\tau$ *of some* $\tau \in \mathcal{T}_{\gamma^{\mathrm{ext}}}$*, we can either detect* $|V^{\mathrm{tmn}}_{\gamma}(I)| > f/\phi$ *or output the profiles of all vertices in* $V^{\mathrm{tmn}}_{\gamma}(I)$*, if we can access* $\mathsf{InnerTerminals}^{\rightarrow}_{\gamma}(\ell_{\mathrm{oc},I}, T_{\gamma^{\mathrm{ext}}})$*.*

Proof. We consider a candidate $\widetilde{V}^{\mathrm{tmn}}_{\gamma}(I)$ of $V^{\mathrm{tmn}}_{I}(I)$ defined as

$$\widetilde{V}^{\mathrm{tmn}}_{\gamma}(I) = \{x \in \mathsf{InnerTerminals}^{\rightarrow}_{\gamma}(\ell_{\mathrm{oc},I}, T_{\gamma^{\mathrm{ext}}}) \mid \mathsf{pos}(\ell_{\mathrm{oc},I}, T_{\gamma^{\mathrm{ext}}}) < \mathsf{pos}(\pi(x, T_{\gamma^{\mathrm{ext}}})) < \mathsf{pos}(r_{\mathrm{oc},I}, T_{\gamma^{\mathrm{ext}}})\}.$$

By definition, we have $\widetilde{V}^{\mathrm{tmn}}_{\gamma}(I) \subseteq V^{\mathrm{tmn}}_{\gamma}(I)$. Thus, if $|\widetilde{V}^{\mathrm{tmn}}_{\gamma}(I)| > f/\phi$, we detect $|V^{\mathrm{tmn}}_{\gamma}(I)| > f/\phi$.

From now, assume $|\widetilde{V}^{\mathrm{tmn}}_{\gamma}(I)| \le f/\phi$, and we claim that $\widetilde{V}^{\mathrm{tmn}}_{\gamma}(I) = V^{\mathrm{tmn}}_{\gamma}(I)$. Assume for contradiction that there exists a vertex $x \in V^{\mathrm{tmn}}_{\gamma}(I) \setminus \widetilde{V}^{\mathrm{tmn}}_{\gamma}(I)$. Note that when constructing $\mathsf{InnerTerminals}^{\rightarrow}_{\gamma}(\ell_{\mathrm{oc},I}, T_{\gamma^{\mathrm{ext}}})$, the vertex x is inside $V^{\rightarrow}_{\ell_{\mathrm{oc},I}}$ because $\pi(x, T_{\gamma^{\mathrm{ext}}}) \in I$ is indeed to the right of $\ell_{\mathrm{oc},I}$. The only scenario in which x is not selected in $\widetilde{V}^{\mathrm{tmn}}_{\gamma}(I)$ is when the size of $\mathsf{InnerTerminals}^{\rightarrow}_{\gamma}(\ell, T_{\gamma^{\mathrm{ext}}})$ reaches $f/\phi + 1$ and each vertex $x' \in \mathsf{InnerTerminals}^{\rightarrow}_{\gamma}(\ell, T_{\gamma^{\mathrm{ext}}})$ has $\mathsf{pos}(\ell_{\mathrm{oc},I}) < \mathsf{pos}(\pi(x', T_{\gamma^\star})) \le \mathsf{pos}(\pi(x, T_{\gamma^\star}))$. However, this means these $f/\phi + 1$ vertices x' will be picked into $\widetilde{V}^{\mathrm{tmn}}_{\gamma}(I)$, contradicting $|\widetilde{V}^{\mathrm{tmn}}_{\gamma}(I)| \le f/\phi$. $\square$

Corollary 3.28. *Given an interval $I \in \mathcal{I}_\tau$ of some non-giant subtree $\tau \in \mathcal{T}_{\gamma^{\text{ext}}}$, we have $V^{\text{tmn}}_\gamma(I) \subseteq \mathsf{InnerTerminals}^{\rightarrow}_\gamma(\ell_{\text{oc},I}, T_{\gamma^{\text{ext}}})$.*

Proof. Recall the candidate $\widetilde{V}^{\text{tmn}}_\gamma(I)$ in the proof of Lemma 3.27. By definition, $\widetilde{V}^{\text{tmn}}_\gamma(I) \subseteq \mathsf{InnerTerminals}^{\rightarrow}_\gamma(\ell_{\text{oc},I}, T_{\gamma^{\text{ext}}})$. Furthermore, because τ is non-giant, we have $V^{\text{tmn}}_\gamma(I) = \widetilde{V}^{\text{tmn}}_\gamma(I)$. $\square$

Labels $\mathsf{NeighborEdge}^{\rightarrow}_\gamma(v_{\text{oc}}, T)$ and $\mathsf{NeighborVertex}^{\rightarrow}_\gamma(v_{\text{oc}}, T)$. For each component $\Gamma \in \mathcal{C}$, each (extended) Steiner tree $T \in \{T_{\gamma^{\text{ext}}}\} \cup \{T_{\gamma'} \mid \gamma' \prec \gamma\}$, and each occurrence $v_{\text{oc}} \in \mathsf{Euler}(T)$ not owned by S-vertices, we construct the label $\mathsf{NeighborEdge}^{\rightarrow}_\gamma(v_{\text{oc}}, T)$ and $\mathsf{NeighborVertex}^{\rightarrow}_\gamma(v_{\text{oc}}, T)$ as follows.

For each undirected edge $\{u, v\} \in E(\widetilde{G}_\Gamma)$ with type η, we treat it as two directed edges (u, v, η) and (v, u, η). Then we let $E^{\rightarrow}_{v_{\text{oc}}}$ be a list collecting all *directed* edges $e = (x, y, \eta) \in E(\widetilde{G}_\Gamma)$ s.t. $y \in \gamma$ and x has a principal occurrence $\pi(x, T)$ to the right of v_{oc}. We sort edges $e = (x, y, \eta) \in E^{\rightarrow}_{v_{\text{oc}}}$ in order of $\pi(x, T)$ from leftmost to rightmost.

In what follows, we construct a list $\mathsf{NeighborEdge}^{\rightarrow}_\gamma(v_{\text{oc}}, T)$ of edges in $E^{\rightarrow}_{v_{\text{oc}}}$ by considering edges in $E^{\rightarrow}_{v_{\text{oc}}}$ one by one according to the order. Meanwhile, we will maintain another list

$$\mathsf{NeighborVertex}^{\rightarrow}_\gamma(v_{\text{oc}}, T) = \{y \mid (x, y, \eta) \in \mathsf{NeighborEdge}^{\rightarrow}_\gamma(v_{\text{oc}}, T)\}.$$

Each edge (x, y, η) in $\mathsf{NeighborEdge}^{\rightarrow}_\gamma(v_{\text{oc}}, T)$ is stored in the form $(\mathsf{pos}(\pi(x, T)), \mathsf{id}(y), \mathsf{id}(\eta))$, and each vertex y in $\mathsf{NeighborVertex}^{\rightarrow}_\gamma(v_{\text{oc}}, T)$ is stored as $\mathsf{profile}(y)$.

Let $e = (x, y, \eta) \in E^{\rightarrow}_{v_{\text{oc}}}$ be the current edge, and we add e into $\mathsf{NeighborEdge}^{\rightarrow}_\gamma(v_{\text{oc}}, T)$ if all the following three conditions hold at this moment.

1. There is no edge $e' = (x', y', \eta') \in \mathsf{NeighborEdge}^{\rightarrow}_\gamma(v_{\text{oc}}, T)$ s.t. $y' = y$ and $\eta' = \eta$.
2. The number of edges $e' = (x', y', \eta') \in \mathsf{NeighborEdge}^{\rightarrow}_\gamma(v_{\text{oc}}, T)$ s.t. $y' = y$ is smaller than $h(f+2)+1$.
3. The size of $\mathsf{NeighborVertex}^{\rightarrow}_\gamma(v_{\text{oc}}, T)$ should be smaller than $h(f+2)\lambda_{\text{nb}} + f/\phi + 1$.

Note that conditions 2 and 3 guarantee that the number of edges in $\mathsf{NeighborEdge}^{\rightarrow}_\gamma(v_{\text{oc}}, T)$ is at most $(h(f+2)+1)(h(f+2)\lambda_{\text{nb}} + f/\phi + 1)$.

We will store the labels $\mathsf{NeighborEdge}^{\rightarrow}_\gamma(v_{\text{oc}}, T)$ and $\mathsf{NeighborVertex}^{\rightarrow}_\gamma(v_{\text{oc}}, T)$ at the vertex v owning v_{oc}. Exceptionally, when v_{oc} is $\mathsf{start}(T)$ or $\mathsf{end}(T)$, we store $\mathsf{NeighborEdge}^{\rightarrow}_\gamma(v_{\text{oc}}, T)$ and $\mathsf{NeighborVertex}^{\rightarrow}_\gamma(v_{\text{oc}}, T)$ at all vertices $v \in \bar{\Gamma}$, where $\bar{\Gamma} = \Gamma$ if $T = T_{\gamma^{\text{ext}}}$ and $\bar{\Gamma} = \Gamma'$ if $T = T_{\gamma'}$ for some $\gamma' \prec \gamma$.

Implement $\mathsf{ListNeighbors}(\tau, \Gamma)$. From $\mathsf{id}(\tau)$, we can obtain $\mathsf{id}(T)$ where T is the (extended) Steiner tree owning τ. For each interval $I \in \mathcal{I}_\tau$, we let

$$N_\gamma(I) = N^{\widetilde{G}^{\text{qry}}_\Gamma}(V^{\text{tmn}}(I)) \cap \gamma$$

be the neighbors of $V^{\text{tmn}}(I)$ in graph $\widetilde{G}^{\text{qry}}_\Gamma$ falling in γ. Furthermore, let

$$N^{\text{inc}}_\gamma(I) = \{y \mid \{x, y\} \in E(\widetilde{G}^{\text{qry}}_\Gamma), x \in V^{\text{tmn}}(I)\} \cap \gamma \tag{2}$$

denote the vertices in γ adjacent to some $x \in V^{\text{tmn}}(I)$ in the graph $\widetilde{G}^{\text{qry}}_\Gamma$. By definition, $N^{\text{inc}}_\gamma(I)$ may include vertices in $V^{\text{tmn}}(I)$, and $N_\gamma(I) = N^{\text{inc}}_\gamma(I) \setminus V^{\text{tmn}}(I)$ is exactly the set by excluding $V^{\text{tmn}}(I)$ from $N^{\text{inc}}_\gamma(I)$. Therefore,

$$N_\gamma(I) \subseteq N^{\text{inc}}_\gamma(I) \subseteq N_\gamma(I) \cup V^{\text{tmn}}_\gamma(I).$$

To answer $\mathsf{ListNeighbors}(\tau, \Gamma)$, it suffices to apply Lemma 3.29 (with access guarantee from Observation 3.30) on each $I \in \mathcal{I}_\tau$ by the following reasons. By Observation 3.24, we have $V^{\text{tmn}}(\tau) = \bigcup_{I \in \mathcal{I}_\tau} V^{\text{tmn}}(I)$. Therefore, we have

$$N_\gamma(\tau) \subseteq \bigcup_{I \in \mathcal{I}_\tau} N_\gamma^{\text{inc}}(I) \subseteq N_\gamma(\tau) \cup V_\gamma^{\text{tmn}}(\tau).$$

If Lemma 3.29 detect $|N_\gamma^{\text{inc}}(I)| > f/\phi$ for some $I \in \mathcal{I}_\tau$, it means $|V_\gamma^{\text{tmn}}(\tau)| + |N_\gamma(\tau)| > f/\phi$. Otherwise, Lemma 3.29 outputs (the identifiers of) all vertices in $N_\gamma^{\text{inc}}(I)$ for all $I \in \mathcal{I}_\tau$. To get $N_\gamma(\tau)$, we just need to remove from $\bigcup_{I \in \mathcal{I}_\tau} N_\gamma^{\text{inc}}(I)$ vertices in $V_\gamma^{\text{tmn}}(\tau)$. Note that we can check whether a vertex $v \in \bigcup_{I \in \mathcal{I}_\tau} N_\gamma(I)$ is in $V_\gamma^{\text{tmn}}(\tau)$ using $\mathsf{IsTerminal}(v, \tau)$, or even just get $V_\gamma^{\text{tmn}}(\tau)$ explicitly using $\mathsf{ListTerminals}(\tau, \Gamma)$.

Lemma 3.29. *Given $\mathsf{profile}(I)$ of an interval $I \in \mathcal{I}_\tau$ of some $\tau \in \mathcal{T}_\Gamma^{\text{ext}}$, we can either detect $|N_\gamma^{\text{inc}}(I)| > f/\phi$ or output the profiles of all vertices in $N_\gamma^{\text{inc}}(I)$, if we can access $\mathsf{NeighborEdge}_\gamma^{\rightarrow}(\ell_{\text{oc},I}, T)$ and $\mathsf{NeighborVertex}_\gamma^{\rightarrow}(\ell_{\text{oc},I}, T)$, where T is the (extended) Steiner tree owning τ.*

Proof. We consider a candidate $\widetilde{N}_\gamma^{\text{inc}}(I)$ of $N_\gamma^{\text{inc}}(I)$ defined as

$$\begin{aligned}\widetilde{N}_\gamma^{\text{inc}}(I) = \{y \mid (x, y, \eta) \in \mathsf{NeighborEdge}_\gamma^{\rightarrow}(\ell, T) \\ \text{s.t. } \mathsf{pos}(\ell_{\text{oc},I}, T) < \mathsf{pos}(\pi(x, T)) < \mathsf{pos}(r_{\text{oc},I}, T) \text{ and } \eta \text{ is not affected}\}.\end{aligned}$$

Note that $\mathsf{pos}(\ell_{\text{oc},I}, T)$ and $\mathsf{pos}(r_{\text{oc},I}, T)$ are stored in $\mathsf{profile}(I)$, $\mathsf{pos}(\pi(x, T))$ is stored in $\mathsf{NeighborEdge}_\gamma^{\rightarrow}(\ell_{\text{oc},I}, T)$, and we can obtain $\mathsf{profile}(y)$ from $\mathsf{NeighborVertex}^{\rightarrow}(\ell_{\text{oc},I}, T)$.

By the construction of $\mathsf{NeighborEdge}_\gamma^{\rightarrow}(\ell, T)$, we have $\widetilde{N}_\gamma^{\text{inc}}(I) \subseteq N_\gamma^{\text{inc}}(I)$. Thus if $|\widetilde{N}_\gamma^{\text{inc}}(I)| > f/\phi$, we detect $|N_\gamma^{\text{inc}}(I)| > f/\phi$. From now we assume $|\widetilde{N}_\gamma^{\text{inc}}(I)| \leq f/\phi$, and we will show that $N_\gamma^{\text{inc}}(I) = \widetilde{N}_\gamma^{\text{inc}}(I)$.

Assume for contradiction that there exists y s.t. $y \in N_\gamma^{\text{inc}}(I)$ and $y \notin \widetilde{N}_\gamma^{\text{inc}}(I)$. Let $e = \{x, y\} \in E(\widetilde{G}_\Gamma^{\text{qry}})$ be an edge with endpoints $x \in V^{\text{tmn}}(I)$ and y and type η (η must be unaffected since we exclude edges with affected types from $\widetilde{G}_\Gamma^{\text{qry}}$). Because $\pi(x, T)$ is to the right of $\ell_{\text{oc},I}$ and $y \in \gamma$, when constructing $\mathsf{NeighborEdge}_\gamma^{\rightarrow}(\ell_{\text{oc},I}, T)$, we will add the edge (x, y, η) into $E_{\ell_{\text{oc},I}}^{\rightarrow}$. However, we know $(x, y, \eta) \notin \mathsf{NeighborEdge}_\gamma^{\rightarrow}(\ell_{\text{oc},I}, T)$, because otherwise y will be added in $\widetilde{N}_\gamma^{\text{inc}}(I)$. Namely, when we try to add (x, y, η) into $\mathsf{NeighborEdge}_\gamma^{\rightarrow}(\ell_{\text{oc},I}, T)$, at least one of the three conditions is violated.

Suppose Condition 1 is violated, i.e. there is an edge $(x', y', \eta') \in \mathsf{NeighborEdge}_\gamma^{\rightarrow}(\ell_{\text{oc},I}, T)$ s.t. $y' = y$ and $\eta' = \eta$. Furthermore, we have $\mathsf{pos}(\ell_{\text{oc},I}) < \mathsf{pos}(\pi(x', T)) \leq \mathsf{pos}(\pi(x, T))$ because (x', y', η') is added before. This means $y' = y$ will be selected into $\widetilde{N}_\gamma^{\text{inc}}(I)$, a contradiction.

Suppose Condition 2 is violated, i.e. there are at least $h(f+2)+1$ many edges $(x', y', \eta') \in \mathsf{NeighborEdge}_\gamma^{\rightarrow}(\ell, T)$ with $y' = y$, and all of them satisfy $\mathsf{pos}(\ell_{\text{oc},I}) < \mathsf{pos}(\pi(x', T)) \leq \pi(x, T)$. Furthermore, all these edges have different types η' by Condition 1. Because the number of affected types is at most $h(f+2)$ by Observation 3.10, at least one of these edges has an unaffected type η'. Then $y' = y$ will be selected into $\widetilde{N}_\gamma^{\text{inc}}(I)$, a contradiction.

Suppose Condition 3 is violated, i.e. the number of vertices in $\mathsf{NeighborVertex}_\gamma^{\rightarrow}(\ell, T)$ reaches $hf\lambda_{\text{nb}} + f/\phi + 1$. We know that at most $hf\lambda_{\text{nb}}$ vertices in γ can be incident to affected edges, because (1) the number of affected components is at most $h(f+2)$ by Observation 3.10, and (2) for each affected component, the shortcut edges it created will be incident to at most λ_{nb} vertices in γ (recall the construction of shortcut edges). Thus, at least $f/\phi+1$ vertices in $\mathsf{NeighborVertex}^{\rightarrow}_{\gamma}(\ell, T)$ will be incident to only unaffected edges. All these vertices will be picked into $\widetilde{N}^{\mathrm{inc}}_{\gamma}(I)$, which means $|\widetilde{N}^{\mathrm{inc}}_{\gamma}(I)| > f/\phi$, a contradiction.

$\square$

Labels $\mathsf{NeighborEdge}^{\leftarrow}_{\gamma}(v_{\mathrm{oc}}, T)$ and $\mathsf{NeighborVertex}^{\leftarrow}_{\gamma}(v_{\mathrm{oc}}, T)$. Symmetrically, for each component $\Gamma \in \mathcal{C}$, each (extended) Steiner tree $T \in \{T_{\gamma^{\mathrm{ext}}}\} \cup \{T_{\gamma'} \mid \gamma' \prec \gamma\}$, and each occurrence $v_{\mathrm{oc}} \in \mathsf{Euler}(T)$ not owned by S-vertices, we work on the *reversed* $\mathsf{Euler}(T)$, and construct and store labels $\mathsf{NeighborEdge}^{\leftarrow}_{\gamma}(v_{\mathrm{oc}}, T)$ and $\mathsf{NeighborVertex}^{\leftarrow}_{\gamma}(v_{\mathrm{oc}}, T)$ in the same way as $\mathsf{NeighborEdge}^{\rightarrow}_{\gamma}(v_{\mathrm{oc}}, T)$ and $\mathsf{NeighborVertex}^{\rightarrow}_{\gamma}(v_{\mathrm{oc}}, T)$. Observation 3.30 below can be proved using an argument similar to Observation 3.26.

We will use labels $\mathsf{NeighborEdge}^{\leftarrow}_{\gamma}(v_{\mathrm{oc}}, T)$ and $\mathsf{NeighborVertex}^{\leftarrow}_{\gamma}(v_{\mathrm{oc}}, T)$ in Section 3.4.4.

Observation 3.30. *Given* $\mathsf{id}(\Gamma)$ *and* $\mathsf{profile}(\tau)$ *of some* $\tau \in \mathcal{T}^{\mathrm{ext}}_{\Gamma}$*, we can access* $\mathsf{NeighborEdge}^{\rightarrow}_{\gamma}(\ell_{\mathrm{oc},I}, T)$, $\mathsf{NeighborVertex}^{\rightarrow}_{\gamma}(\ell_{\mathrm{oc},I}, T)$, $\mathsf{NeighborEdge}^{\leftarrow}_{\gamma}(r_{\mathrm{oc},I}, T)$, *and* $\mathsf{NeighborVertex}^{\leftarrow}_{\gamma}(r_{\mathrm{oc},I}, T)$ *for all* $I \in \mathcal{I}_{\tau}$*, where* T *is the (extended) Steiner tree owning* τ*.*

Corollary 3.31. *Given an interval* $I \in \mathcal{I}_{\tau}$ *of some non-giant subtree* $\tau \in \mathcal{T}^{\mathrm{ext}}_{\Gamma}$*, we have* $N^{\mathrm{inc}}_{\gamma}(I) \subseteq \mathsf{NeighborVertex}^{\rightarrow}_{\gamma}(\ell_{\mathrm{oc},I}, T)$ *and* $N^{\mathrm{inc}}_{\gamma}(I) \subseteq \mathsf{NeighborVertex}^{\leftarrow}_{\gamma}(r_{\mathrm{oc},I}, T)$*, where* T *is the (extended) Steiner tree owning* τ*.*

Proof. This corollary follows the proof of Lemma 3.29. Recall the definition of the candidate $\widetilde{N}^{\mathrm{inc}}_{\gamma}(I)$. When τ is non-giant, we have $|\widetilde{N}^{\mathrm{inc}}_{\gamma}(I)| \le f/\phi$ and $N^{\mathrm{inc}}_{\gamma}(I) = \widetilde{N}^{\mathrm{inc}}_{\gamma}(I)$. By definition, $\widetilde{N}^{\mathrm{inc}}_{\gamma}(I) \subseteq \mathsf{NeighborVertex}^{\rightarrow}_{\gamma}(\ell_{\mathrm{oc},I}, T)$, so $N^{\mathrm{inc}}_{\gamma}(I) \subseteq \mathsf{NeighborVertex}^{\rightarrow}_{\gamma}(\ell_{\mathrm{oc},I}, T)$. $N^{\mathrm{inc}}_{\gamma}(I) \subseteq \mathsf{NeighborVertex}^{\leftarrow}_{\gamma}(r_{\mathrm{oc},I}, T)$ can be proved similarly. $\square$

3.4.4 Labels for Implementing $\mathsf{EnumFromGiant}(\tau_y, \Gamma)$

Before introducing the labels, we first discuss the high level idea. For an undirected (multi-)graph H and two vertex set $X, Y \subseteq V(H)$ (X, Y may intersect), we use $\delta_H(X, Y)$ to denote the number of edges with one endpoint in X and the other one in Y, but an edge with both endpoints in $X \cap Y$ will be counted twice. Namely, $\delta_H(X, Y) = \sum_{x \in X, y \in Y} \delta_H(x, y)$.

Our goal is to compute $\delta_{\widetilde{G}^{\mathrm{qry}}_{\Gamma} \setminus F}(\bigcup_{\text{giant } \tau_x \in \mathcal{T}^{\mathrm{ext}}_{\Gamma}} V^{\mathrm{tmn}}(\tau_x), V^{\mathrm{tmn}}_{\gamma}(\tau_y))$ for a non-giant subtree $\tau_y \in \mathcal{T}_{\gamma^{\mathrm{ext}}}$. We first rewrite it to get rid of the affected shortcut edges. Precisely, we have

$$\begin{aligned}\delta_{\widetilde{G}^{\mathrm{qry}}_{\Gamma} \setminus F}\Big(\bigcup_{\text{giant } \tau_x \in \mathcal{T}^{\mathrm{ext}}_{\Gamma}} V^{\mathrm{tmn}}(\tau_x), V^{\mathrm{tmn}}_{\gamma}(\tau_y)\Big) = &\sum_{\text{giant } \tau_x \in \mathcal{T}^{\mathrm{ext}}_{\Gamma}} \delta_{\widetilde{G}_{\Gamma}}(V^{\mathrm{tmn}}(\tau_x), V^{\mathrm{tmn}}_{\gamma}(\tau_y)) \\ &- \sum_{\text{giant } \tau_x \in \mathcal{T}^{\mathrm{ext}}_{\Gamma}} \delta^{\mathrm{aff}}_{\widetilde{G}_{\Gamma}}(V^{\mathrm{tmn}}(\tau_x), V^{\mathrm{tmn}}_{\gamma}(\tau_y)),\end{aligned}$$

where $\delta^{\mathrm{aff}}_{\widetilde{G}_{\Gamma}}(V^{\mathrm{tmn}}(\tau_x), V^{\mathrm{tmn}}_{\gamma}(\tau_y))$ is the number of affected edges in $\widetilde{\Gamma}$ connecting $V^{\mathrm{tmn}}(\tau_x)$ and $V^{\mathrm{tmn}}_{\gamma}(\tau_y)$. This equation holds because each $\tau \in \mathcal{T}^{\mathrm{ext}}_{\Gamma}$ has $V^{\mathrm{tmn}}(\tau_x) \subseteq Q^{\mathrm{ext}}_{\Gamma} \setminus F$ and by definition $\widetilde{G}^{\mathrm{qry}}_{\Gamma} = \widetilde{G}_{\Gamma}[Q^{\mathrm{ext}}_{\Gamma}] \setminus \{e \in E(\widetilde{G}_{\Gamma}) \mid e \text{ is affected}\}$. For each term $\delta^{\mathrm{aff}}_{\widetilde{G}_{\Gamma}}(V^{\mathrm{tmn}}(\tau_x), V^{\mathrm{tmn}}_{\gamma}(\tau_y))$, its value is given by Lemma 3.32.

Therefore, it remains to compute $\sum_{\text{giant } \tau_x \in \mathcal{T}_\Gamma^{\text{ext}}} \delta_{\widetilde{G}_\Gamma}(V^{\text{tmn}}(\tau_x), V_\gamma^{\text{tmn}}(\tau_y))$. We further have

$$\sum_{\text{giant } \tau_x \in \mathcal{T}_\Gamma^{\text{ext}}} \delta_{\widetilde{G}_\Gamma}(V^{\text{tmn}}(\tau_x), V_\gamma^{\text{tmn}}(\tau_y)) = \sum_{x \in Q_\Gamma^{\text{ext}}} \delta_{\widetilde{G}_\Gamma}(x, V_\gamma^{\text{tmn}}(\tau_y)) - \sum_{x \in Q_\Gamma^{\text{ext}} \cap F} \delta_{\widetilde{G}_\Gamma}(x, V_\gamma^{\text{tmn}}(\tau_y)) - \sum_{\substack{\text{non-giant} \\ \tau_x \in \mathcal{T}_\Gamma^{\text{ext}}}} \delta_{\widetilde{G}_\Gamma}(V^{\text{tmn}}(\tau_x), V_\gamma^{\text{tmn}}(\tau_y)),$$

because $\{V^{\text{tmn}}(\tau) \mid \tau \in \mathcal{T}_\Gamma^{\text{ext}}\}$ partitions $Q_\Gamma^{\text{ext}} \setminus F$. The second equation basically says that, to compute the number of edges from giant subtrees to $V_\gamma^{\text{tmn}}(\tau_y)$, we can first compute the number of edges from all vertices in Q_Γ^{ext} to $V_\gamma^{\text{tmn}}(\tau_y)$, and then subtract those edges starting from failed vertices and non-giant subtrees. We will compute the three terms on the right hand side using Lemmas 3.33, 3.35 and 3.36 respectively.

Label $\mathsf{ArtificialEdge}(\gamma', \widetilde{G}_\Gamma)$. For each component Γ and each strict descendant $\Gamma' \prec \Gamma$, we construct a label $\mathsf{ArtificialEdge}(\gamma', \widetilde{G}_\Gamma)$ which, for each artificial edges $\{x, y\}$ with type γ' in $\widetilde{G}_\Gamma$, stores a tuple $(\mathsf{profile}(x), \mathsf{profile}(y), \mathsf{id}(\Gamma'))$. We store $\mathsf{ArtificialEdge}(\gamma', \widetilde{G}_\Gamma)$ at each vertex $v \in \Gamma'$.

Lemma 3.32. *Given* $\mathsf{id}(\Gamma)$ *and* $\mathsf{profile}(\tau_x), \mathsf{profile}(\tau_y)$ *of* $\tau_x \in \mathcal{T}_\Gamma^{\text{ext}}$ *and non-giant* $\tau_y \in \mathcal{T}_{\gamma^{\text{ext}}}$*, we can access* $\mathsf{ArtificialEdge}(\gamma', \widetilde{G}_\Gamma)$ *for each affected* $\gamma' \prec \gamma$*, and then compute* $\delta^{\text{aff}}_{\widetilde{G}_\Gamma}(V^{\text{tmn}}(\tau_x), V_\gamma^{\text{tmn}}(\tau_y))$.

Proof. First, we can get $\mathsf{id}(\Gamma')$ for all affected component $\Gamma' \prec \Gamma$. For each affected $\Gamma' \prec \Gamma$, we then access $\mathsf{ArtificialEdge}_{\widetilde{G}_\Gamma}(\gamma_{\text{aff}})$ from an arbitrary vertex in $\Gamma' \cap (F \cup \{s, t\})$ (such vertex exists because Γ' is affected). In other words, we can obtain $(\mathsf{profile}(u), \mathsf{profile}(v))$ for all affected shortcut edges $\{u, v\}$ in $\widetilde{G}_\Gamma$.

Therefore, we just need to scan each affected $\widetilde{G}_\Gamma$-edge $\{u, v\}$, and decide the membership of each u, v at each $V^{\text{tmn}}(\tau_x), V_\gamma^{\text{tmn}}(\tau_y)$. The membership at $V_\gamma^{\text{tmn}}(\tau_y)$ can be easily decided because we can obtain $V_\gamma^{\text{tmn}}(\tau_y)$ explicitly by $\mathsf{ListTerminals}(\tau_y, \Gamma)$ (note that τ_y is non-giant). To decide whether $u \in V^{\text{tmn}}(\tau_x)$ (resp. $v \in V^{\text{tmn}}(\tau_x)$), we just need to invoke $\mathsf{IsTerminal}(u, \tau_x)$ (resp. $\mathsf{IsTerminal}(v, \tau_x)$). $\square$

Label $\mathsf{Degree}(x, \widetilde{G}_\Gamma[\gamma^{\text{ext}}])$. For each component $\Gamma \in \mathcal{C}$ and each occurrence $v_{\text{oc}} \in \mathsf{Euler}(T_{\gamma^{\text{ext}}})$ not owned by S-vertices, we construct the following labels.

For each vertex $x \in \mathsf{InnerTerminals}^{\rightarrow}_\gamma(v_{\text{oc}}, T_{\gamma^{\text{ext}}})$, we let $\mathsf{Degree}(x, \widetilde{G}_\Gamma[\gamma^{\text{ext}}])$ denote the degree of vertex x in graph $\widetilde{G}_\Gamma[\gamma^{\text{ext}}]$ (i.e. the subgraph of the sparsified shortcut graph $\widetilde{G}_\Gamma$ induced by the extended core γ^{ext}). We store $\mathsf{Degree}(x, \widetilde{G}_\Gamma[\gamma^{\text{ext}}])$ along with the vertex x in the label $\mathsf{InnerTerminals}^{\rightarrow}_\gamma(v_{\text{oc}}, T_{\gamma^{\text{ext}}})$.

Labels $\mathsf{Enum}(\gamma', y, \widetilde{G}_\Gamma)$ and $\mathsf{Enum}(x, y, \widetilde{G}_\Gamma)$. For each component $\Gamma \in \mathcal{C}$ and each $\Gamma' \preceq \Gamma$ s.t. γ' is not in γ^{ext}, we construct the following labels. Note that $|N_\gamma(\Gamma')| \leq \lambda_{\text{nb}}$ because γ' is not in γ^{ext}.

- For each vertex $y \in N_\gamma(\Gamma')$, we let $\mathsf{Enum}(\gamma', y, \widetilde{G}_\Gamma)$ be the total number of $\widetilde{G}_\Gamma$-edges connecting some vertex $x \in \gamma'$ and the vertex γ. We store $\mathsf{Enum}(\gamma', y, \widetilde{G}_\Gamma)$ at each vertex $v \in \Gamma'$.
- For each vertex $x \in \gamma'$ and vertex $y \in N_\gamma(\Gamma')$, let $\mathsf{Enum}(x, y, \widetilde{G}_\Gamma)$ be the number of $\widetilde{G}_\Gamma$-edges connecting x and y (recall that $\widetilde{G}_\Gamma$ is a multigraph, so $\mathsf{Enum}(x, y, \widetilde{G}_\Gamma)$ may be larger than 1), and store it at vertex x.

Lemma 3.33. *Given* $\mathsf{id}(\Gamma)$ *and* $\mathsf{profile}(\tau_y)$ *of some non-giant subtree* $\tau_y \in \mathcal{T}_{\gamma^{\text{ext}}}$*, we can compute* $\delta_{\widetilde{G}_\Gamma}(Q_\Gamma^{\text{ext}}, V_\gamma^{\text{tmn}}(\tau_y))$.

Proof. We further decompose the expression to be

$$\delta_{\widetilde{G}_\Gamma}(Q_\Gamma^{\text{ext}}, V_\gamma^{\text{tmn}}(\tau_y)) = \sum_{x \in \gamma^{\text{ext}}} \delta_{\widetilde{G}_\Gamma}(x, V_\gamma^{\text{tmn}}(\tau_y)) + \sum_{\substack{\text{affected } \gamma' \prec \gamma \\ \text{s.t. } \gamma' \text{ is not in } \gamma^{\text{ext}}}} \delta_{\widetilde{G}_\Gamma}(\gamma', V_\gamma^{\text{tmn}}(\tau_y)),$$

because $\{\gamma' \mid \gamma' \prec \gamma,\ \gamma' \text{ is affected and } \gamma' \text{ is not in } \gamma^{\text{ext}}\} \cup \{\gamma^{\text{ext}}\}$ forms a partition of Q_Γ^{ext}.

For the first term on the right hand side, we can rewrite it as

$$\sum_{x \in \gamma^{\text{ext}}} \delta_{\widetilde{G}_\Gamma}(x, V_\gamma^{\text{tmn}}(\tau_y)) = \sum_{x \in \gamma^{\text{ext}}} \delta_{\widetilde{G}_\Gamma[\gamma^{\text{ext}}]}(x, V_\gamma^{\text{tmn}}(\tau_y)) = \sum_{y \in V_\gamma^{\text{tmn}}(\tau_y)} \mathsf{Degree}(y, \widetilde{G}_\Gamma[\gamma^{\text{ext}}]).$$

To compute it, we first obtain (the profiles of) all vertices in $V_\gamma^{\text{tmn}}(\tau_y)$ using $\mathsf{ListTerminals}(\tau_y, \Gamma)$ (because τ_y is non-giant). By Corollary 3.28, we have

$$V_\gamma^{\text{tmn}}(\tau_y) = \bigcup_{I \in \mathcal{I}_{\tau_y}} V_\gamma^{\text{tmn}}(I) \subseteq \bigcup_{I \in \mathcal{I}_{\tau_y}} \mathsf{InnerTerminals}(\ell_{\text{oc},I}, T_{\gamma^{\text{ext}}}).$$

By Observation 3.26, we can access all these $\mathsf{InnerTerminals}(\ell_{\text{oc},I}, T_{\gamma^{\text{ext}}})$, so we can further access $\mathsf{Degree}(y, \widetilde{G}_\Gamma[\gamma^{\text{ext}}])$ for all $y \in V_\gamma^{\text{tmn}}(\tau_y)$.

Regarding the second term on the right hand side, note that for each vertex $y \in \gamma$ s.t. $y \notin N_\gamma(\Gamma')$, there is no $\widetilde{G}_\Gamma$-edge connecting γ' and y. Therefore, we can compute

$$\delta_{\widetilde{G}_\Gamma}(\gamma', V_\gamma^{\text{tmn}}(\tau)) = \sum_{y \in V_\gamma^{\text{tmn}}(\tau) \cap N_\gamma(\Gamma')} \mathsf{Enum}_{\widetilde{G}_\Gamma}(\gamma', y),$$

because we can access $\mathsf{Enum}_{\widetilde{G}_\Gamma}(\gamma', y)$ for each $y \in N_\gamma(\Gamma')$ at any vertex in $\Gamma' \cap (F \cup \{s,t\})$ (note that $\Gamma' \cap (F \cup \{s,t\})$ is not empty because γ' is affected). $\square$

Labels $\mathsf{IncidentEdge}(x, \widetilde{G}_\Gamma[\gamma^{\text{ext}}])$. For each component $\Gamma \in \mathcal{C}$, we construct the following labels. Recall the construction of the sparsified shortcut graph $\widetilde{G}_\Gamma$ in Section 3.2.5, where we use $\widetilde{G}^{\text{sp}}_{\gamma^{\text{ext}}}$ to denote the simple graph corresponding to $\widetilde{G}_\Gamma[\gamma^{\text{ext}}]$. Furthermore, it is guaranteed that $\widetilde{G}^{\text{sp}}_{\gamma^{\text{ext}}}$ has arboricity λ_{arbo}. Namely, we have an orientation of $\widetilde{G}^{\text{sp}}_{\gamma^{\text{ext}}}$-edges s.t. each vertex $x \in \gamma^{\text{ext}} = V(\widetilde{G}^{\text{sp}}_{\gamma^{\text{ext}}})$ has at most λ_{arbo} incident edges oriented outwards.

For each vertex $x \in \gamma^{\text{ext}}$, we define a label $\mathsf{IncidentEdge}(x, \widetilde{G}_\Gamma[\gamma^{\text{ext}}])$ which, for all its incident edges $(x,y) \in E(\widetilde{G}^{\text{sp}}_{\gamma^{\text{ext}}})$ with orientation $x \to y$, stores $\mathsf{id}(x), \mathsf{id}(y)$ along with the number of $E(\widetilde{G}_\Gamma[\gamma^{\text{ext}}])$-edges connecting x and y (i.e. $\delta_{\widetilde{G}_\Gamma[\gamma^{\text{ext}}]}(x,y)$).

For each vertex $x \in \gamma^{\text{ext}}$, we will store $\mathsf{IncidentEdge}(x, \widetilde{G}_\Gamma[\gamma^{\text{ext}}])$ at x. Besides, for each occurrence $v_{\text{oc}} \in \mathsf{Euler}(T_{\gamma^{\text{ext}}})$ not owned by S-vertices and each vertex $y \in \mathsf{InnerTerminals}^{\rightarrow}_\gamma(v_{\text{oc}}, T_{\gamma^{\text{ext}}}) \subseteq \gamma^{\text{ext}}$, we store $\mathsf{IncidentEdge}(y, \widetilde{G}_\Gamma[\gamma^{\text{ext}}])$ along with the vertex y in the label $\mathsf{InnerTerminals}^{\rightarrow}_\gamma(v_{\text{oc}}, T_{\gamma^{\text{ext}}})$.

Observation 3.34. *Given* $\mathsf{id}(x), \mathsf{id}(y)$ *of two vertices* $x, y \in \gamma^{\text{ext}}$*, if we can access* $\mathsf{IncidentEdge}(x, \widetilde{G}_\Gamma[\gamma^{\text{ext}}])$ *and* $\mathsf{IncidentEdge}(y, \widetilde{G}_\Gamma[\gamma^{\text{ext}}])$*, we can compute* $\delta_{\widetilde{G}_\Gamma[\gamma^{\text{ext}}]}(x,y)$.

Lemma 3.35. *Given* $\mathsf{id}(\Gamma), \mathsf{profile}(\tau_y), \mathsf{id}(x)$ *of some non-giant subtree* $\tau_y \in \mathcal{T}_{\gamma^{\text{ext}}}$ *and some vertex* $x \in \gamma^{\text{ext}} \cap F$*, we can compute* $\delta_{\widetilde{G}_\Gamma}(x, V_\gamma^{\text{tmn}}(\tau_y))$*.*

Proof. Because τ_y is non-giant, we can obtain the identifiers of all vertices in $V_\gamma^{\text{tmn}}(\tau_y)$ using $\mathsf{ListTerminals}(\tau_y, \gamma)$. We consider the following two cases.

Suppose $x \in \gamma^{\text{ext}} \cap F$. Then

$$\delta_{\widetilde{G}_\Gamma}(x, V_\gamma^{\text{tmn}}(\tau_y)) = \delta_{\widetilde{G}_\Gamma[\gamma^{\text{ext}}]}(x, V_\gamma^{\text{tmn}}(\tau_y)) = \sum_{y \in V_\gamma^{\text{tmn}}(\tau_y)} \delta_{\widetilde{G}_\Gamma[\gamma^{\text{ext}}]}(x, y).$$

since $x \in \gamma^{\text{ext}}$ and $V_\gamma^{\text{tmn}}(\tau_y) \subseteq \gamma^{\text{ext}}$, so it suffices to compute the latter. We enumerate (the identifiers of) vertices $y \in V_\gamma^{\text{tmn}}(\tau_y)$, and then compute $\delta_{\widetilde{G}_\Gamma[\gamma^{\text{ext}}]}(x, y)$ using Observation 3.34. Note that we can access $\mathsf{IncidentEdge}(x, \widetilde{G}_\Gamma[\gamma^{\text{ext}}])$ because $x \in F$, and access $\mathsf{IncidentEdge}(y, \widetilde{G}_\Gamma[\gamma^{\text{ext}}])$ because $y \in V_\gamma^{\text{tmn}}(\tau_y) \subseteq \bigcup_{I \in \mathcal{I}_{\tau_y}} \mathsf{InnerTerminals}_\gamma^{\rightarrow}(\ell_{\text{oc},I}, T_{\gamma^{\text{ext}}})$ and for all $I \in \mathcal{I}_{\tau_y}$, $\mathsf{InnerTerminals}_\gamma^{\rightarrow}(\ell_{\text{oc},I}, T_{\gamma^{\text{ext}}})$ are accessible by Observation 3.26.

Suppose $x \in (Q_\Gamma^{\text{ext}} \setminus \gamma^{\text{ext}}) \cap F$, which implies γ_x (the core containing x) satisfies $\gamma_x \prec \gamma$ and γ_x is not in γ^{ext}. Thus, we can access $\mathsf{Enum}(x, y', \widetilde{G}_\Gamma)$ for all $y' \in N_\gamma(\Gamma')$ at vertex x. Again, enumerate (the identifiers of) all vertices $y \in V_\gamma^{\text{tmn}}(\tau_y)$. If $y \in N_\gamma(\Gamma_x)$, then $\delta_{\widetilde{G}_\Gamma}(x, y) = \mathsf{Enum}_{\widetilde{G}_\Gamma}(x, y)$ by definition, otherwise $\delta_{\widetilde{G}_\Gamma}(x, y) = 0$ (for implementation, just compare $\mathsf{id}(y)$ with the $\mathsf{id}(y')$ in each entry $\mathsf{Enum}(x, y, \widetilde{G}_\Gamma)$). Finally, we compute $\delta_{\widetilde{G}_\Gamma}(x, V_\gamma^{\text{tmn}}(\tau)) = \sum_{y \in V_\gamma^{\text{tmn}}(\tau)} \delta_{\widetilde{G}_\Gamma}(x, y)$.

□

<ins>Labels $\mathsf{PrefixEnum}_\gamma^{\text{inc}}(v_{\text{oc}}, T, y)$ and $\mathsf{PrefixEnum}_\gamma^{\text{exc}}(v_{\text{oc}}, T, y)$.</ins> For each component $\Gamma \in \mathcal{C}$, each (extended) Steiner tree $T \in \{T_{\gamma^{\text{ext}}}\} \cup \{T_{\gamma'} \mid \gamma' \prec \gamma\}$, and each occurrence $v_{\text{oc}} \in \mathsf{Euler}(T_\gamma)$ <ins>not</ins> owned by S-vertices, we construct the following labels. Let $\bar{\gamma}$ be the set of vertices with terminal nodes in T (i.e. $\bar{\gamma} = \gamma^{\text{ext}}$ if $T = T_{\gamma^{\text{ext}}}$, and $\bar{\gamma} = \gamma'$ if $T = T_{\gamma'}$ for some $\gamma' \prec \gamma$).

For each vertex $y \in \mathsf{NeighborVertex}_\gamma^{\rightarrow}(v_{\text{oc}}, T)$, we define label $\mathsf{PrefixEnum}_\gamma^{\text{inc}}(v_{\text{oc}}, T, y)$ to be the total number of edges in $E(\widetilde{G}_\Gamma)$ that connect a vertex $x \in \bar{\gamma}$ and the vertex y, summing over all x whose principal occurrence $\pi(x, T)$ is to the left of v_{oc} or exactly v_{oc}. We store $\mathsf{PrefixEnum}_\gamma^{\text{inc}}(v_{\text{oc}}, T, y)$ along with the vertex y in the label $\mathsf{NeighborVertex}_\gamma^{\rightarrow}(v_{\text{oc}}, T)$.

Similarly, for each vertex $y \in \mathsf{NeighborVertex}_\gamma^{\leftarrow}(v_{\text{oc}}, T)$, we define $\mathsf{PrefixEnum}_\gamma^{\text{exc}}(v_{\text{oc}}, T, y)$ to be the total number of edges in $E(\widetilde{G}_\Gamma)$ that connect a vertex $x \in \bar{\gamma}$ and the vertex y, summing over all x whose $\pi(x, T)$ is to the left of v_{oc} (excluding v_{oc}). We store $\mathsf{PrefixEnum}_\gamma^{\text{exc}}(v_{\text{oc}}, T, y)$ along with the vertex y in the label $\mathsf{NeighborVertex}_\gamma^{\leftarrow}(v_{\text{oc}}, T)$.

Lemma 3.36. *Given* $\mathsf{id}(\Gamma), \mathsf{profile}(\tau_x), \mathsf{profile}(\tau_y)$ *of two non-giant subtrees* $\tau_x \in \mathcal{T}_\Gamma^{\text{ext}}$ *and* $\tau_y \in \mathcal{T}_{\gamma^{\text{ext}}}$ *(*τ_x *and* τ_y *can be the same subtree), we can compute* $\delta_{\widetilde{G}_\Gamma}(V^{\text{tmn}}(\tau_x), V_\gamma^{\text{tmn}}(\tau_y))$*.*

Proof. Because τ_x, τ_y are non-giant subtrees, we can obtain the identifiers of vertices in $V_\gamma^{\text{tmn}}(\tau_y)$ using $\mathsf{ListTerminals}(\tau_y, \Gamma)$. Because we have

$$\delta_{\widetilde{G}_\Gamma}(V^{\text{tmn}}(\tau_x), V_\gamma^{\text{tmn}}(\tau_y)) = \sum_{y \in V_\gamma^{\text{tmn}}(\tau_y)} \delta_{\widetilde{G}_\Gamma}(V^{\text{tmn}}(\tau_x), y),$$

it suffices to compute $\delta_{\widetilde{G}_\Gamma}(V^{\text{tmn}}(\tau_x), y)$ for each $y \in V_\gamma^{\text{tmn}}(\tau_x)$. We can further rewrite

$$\delta_{\widetilde{G}_\Gamma}(V^{\text{tmn}}(\tau_x), y) = \sum_{I \in \mathcal{I}_{\tau_x}} \delta_{\widetilde{G}_\Gamma}(V^{\text{tmn}}(I), y)$$

and compute $\delta_{\widetilde{G}_\Gamma}(V^{\mathrm{tmn}}(I), y)$ for each $I \in \mathcal{I}_{\tau_x}$.

In what follows, we focus on compute $\delta_{\widetilde{G}_\Gamma}(V^{\mathrm{tmn}}(\tau_x), y)$ for a fixed $I \in \mathcal{I}_{\tau_x}$ and a fixed vertex $y \in V^{\mathrm{tmn}}_\gamma(\tau_y)$. We consider two cases.

- If $y \in N^{\mathrm{inc}}_\gamma(I)$, we will compute

$$\delta_{\widetilde{G}_\Gamma}(V^{\mathrm{tmn}}(I), y) = \mathsf{PrefixEnum}^{\mathrm{exc}}_\gamma(r_{\mathrm{oc},I}, T, y) - \mathsf{PrefixEnum}^{\mathrm{inc}}_\gamma(\ell_{\mathrm{oc},I}, T, y).$$

  The reason is that, by Corollary 3.31, $N^{\mathrm{inc}}_\gamma(I) \subseteq \mathsf{NeighborVertex}^{\rightarrow}_\gamma(\ell_{\mathrm{oc},I}, T)$ and $N^{\mathrm{inc}}_\gamma(I) \subseteq \mathsf{NeighborVertex}^{\leftarrow}_\gamma(r_{\mathrm{oc},I}, T)$, we can access $\mathsf{PrefixEnum}^{\mathrm{inc}}_\gamma(\ell_{\mathrm{oc},I}, T, y)$ and $\mathsf{PrefixEnum}^{\mathrm{exc}}_\gamma(r_{\mathrm{oc},I}, T, y)$ because we have $\mathsf{id}(y)$ and we can access $\mathsf{NeighborVertex}^{\rightarrow}_\gamma(\ell_{\mathrm{oc},I}, T)$ and $\mathsf{NeighborVertex}^{\leftarrow}_\gamma(r_{\mathrm{oc},I}, T)$ by Observation 3.30.

- If $y \notin N^{\mathrm{inc}}_\gamma(I)$, we claim that all edges in $\widetilde{G}_\Gamma$ connecting y and $V^{\mathrm{tmn}}(\tau_x)$ are *affected*. To see this, assume for contradiction that there is an *unaffected* edge $e \in E(\widetilde{G}_\Gamma)$ connecting y and $V^{\mathrm{tmn}}(\tau_x)$. Recall that $\widetilde{G}^{\mathrm{qry}}_\Gamma = \widetilde{G}_\Gamma[Q^{\mathrm{ext}}_\Gamma] \setminus \hat{E}_{\Gamma,\mathrm{aff}}$ (the subgraph of $\widetilde{G}_\Gamma$ induced by vertices Q^{ext}_Γ excluding all affected edges). Because $V^{\mathrm{tmn}}(I) \subseteq Q^{\mathrm{ext}}_\Gamma$ and $y \in Q^{\mathrm{ext}}_\Gamma$, this edge e is in $\widetilde{G}^{\mathrm{qry}}_\Gamma$, so $y \in N^{\mathrm{inc}}_\gamma(I)$ by its definition (see Equation (2)), a contradiction.

  Therefore, we can use a strategy similar to the proof of Lemma 3.32. Concretely, we can obtain $(\mathsf{profile}(u), \mathsf{profile}(v))$ for all affected shortcut edges $\{u, v\}$ in $\widetilde{G}_\Gamma$, and for each of them, check if it contributes to $\delta_{\widetilde{G}_\Gamma}(V^{\mathrm{tmn}}(I), y)$ using $\mathsf{IsTerminal}$.

□

3.4.5 Space Analysis

Finally, we analyse the space of our labeling schemes. First, we bound the size of one label for all different types of labels.

- $\mathsf{id}(\cdot)$. It takes $O(\log n)$ bits.
- $\mathsf{pos}(v_{\mathrm{oc}})$ of occurrences. It takes $O(\log n)$ bits because the length of each $\mathsf{Euler}(T)$ is polynomial.
- $\mathsf{profile}(v_{\mathrm{oc}})$ of occurrences. It takes $O(\log n)$ bits because it stores two $\mathsf{id}(\cdot)$ and one $\mathsf{pos}(v_{\mathrm{oc}})$.
- $\mathsf{occurrences}(v, T)$. It takes $O(h\Delta \log n)$ bits because it stores $O(h\Delta)$ occurrence-profiles. Recall that each vertex $v \notin S$ owns at most $O(h\Delta)$ occurrences in $\mathsf{Euler}(T)$ by Lemma 3.8.
- $\mathsf{profile}(\Gamma)$ of components. It takes $O(h \log n)$ bits because it stores h $\mathsf{id}(\cdot)$ and h one-bit indicators. Note that the number of ancestors of Γ is h.
- $\mathsf{profile}(v)$ of vertices. It takes $O(h \log n)$ bits because it stores h occurrence-profiles. Note that the number of (extended) Steiner trees $T \in \{T_{\gamma_v}\} \cup \{T_{\gamma^{\mathrm{ext}}} \mid v \in \gamma^{\mathrm{ext}}\}$ is h.
- $\mathsf{SuccTerminal}(v_{\mathrm{oc}}, T)$. It takes $O(\log n)$ bits because it stores one occurrence-profile.
- $\mathsf{InnerTerminals}^{\rightarrow}_\gamma(v_{\mathrm{oc}}, T_{\gamma^{\mathrm{ext}}})$. It takes $O(hf \log n/\phi)$ bits because it stores $f/\phi{+}1$ vertex-profiles.

- $\underline{\mathsf{NeighborEdge}^{\rightarrow}_{\gamma}(v_{\text{oc}}, T) \text{ and } \mathsf{NeighborEdge}^{\leftarrow}_{\gamma}(v_{\text{oc}}, T).}$ It takes

  $$(h(f+2)+1)(h(f+2)\lambda_{\text{nb}} + f/\phi + 1) \cdot O(\log n) \text{ bits,}$$

  because the number of edges in $\mathsf{NeighborEdge}^{\rightarrow}_{\gamma}(v_{\text{oc}}, T)$ is at most $(h(f+2)+1)(h(f+2)\lambda_{\text{nb}} + f/\phi + 1)$, and for each edge we store one occurrence-position and two $\mathsf{id}(\cdot)$.

- $\underline{\mathsf{NeighborVertex}^{\rightarrow}_{\gamma}(v_{\text{oc}}, T) \text{ and } \mathsf{NeighborVertex}^{\leftarrow}_{\gamma}(v_{\text{oc}}, T).}$ It takes $O((hf\lambda_{\text{nb}} + f/\phi)h \log n)$ bits because the number of vertices in $\mathsf{NeighborVertex}^{\rightarrow}_{\gamma}(v_{\text{oc}}, T)$ is at most $h(f+2)\lambda_{\text{nb}} + f/\phi + 1$, and for each vertex we store its profile.

- $\underline{\mathsf{ArtificialEdge}(\gamma', \widetilde{G}_{\Gamma}).}$ It takes $O(h^2\lambda^2_{\text{nb}} \log n)$ bits by the following reasons. Because $\widetilde{G}_{\Gamma}$ is a subgraph of $\hat{G}_{\Gamma}$, the number of γ'-type edges in $\widetilde{G}_{\Gamma}$ is at most that in $\hat{G}_{\Gamma}$. Recall the construction of shortcut edges. The γ'-type edges in $\hat{G}_{\Gamma}$ forms a biclique between $\hat{N}(\Gamma') \cap \Gamma$ and $\hat{N}_{\gamma}(\Gamma')$. Combining $|\hat{N}(\Gamma') \cap \Gamma| \leq |\hat{N}(\Gamma')| \leq h\lambda_{\text{nb}}$ and $|\hat{N}_{\gamma}(\Gamma')| \leq \lambda_{\text{nb}}$, we have the number of γ'-type edges in $\widetilde{G}_{\Gamma}$ is $O(h\lambda^2_{\text{nb}})$. Lastly, for each such edge, we store two vertex-profiles and one $\mathsf{id}(\cdot)$.

- $\underline{\mathsf{IncidentEdge}(x, \widetilde{G}_{\Gamma}[\gamma^{\text{ext}}]).}$ It takes $O(\lambda_{\text{arbo}} \log n)$ bits because there are λ_{arbo} edges and for each edge we store two $\mathsf{id}(\cdot)$ and one polynomially bounded number.

- $\underline{\mathsf{Degree}(x, \widetilde{G}_{\Gamma}[\gamma^{\text{ext}}]), \mathsf{Enum}(\gamma', y, \widetilde{G}_{\Gamma}), \mathsf{Enum}(x, y, \widetilde{G}_{\Gamma}), \mathsf{PrefixEnum}^{\text{inc}}_{\gamma}(v_{\text{oc}}, T, y), \mathsf{PrefixEnum}^{\text{exc}}_{\gamma}(v_{\text{oc}}, T, y).}$ Each of them takes $O(\log n)$ bits because they are polynomially bounded number.

Next, fixing a vertex $v \in V(G)$, we bound the number of labels at v for each label-type.

- $\underline{\mathsf{occurrences}(v, T).}$ The number is $O(h)$ because the number of (extended) Steiner trees T s.t. $V(T)$ has nodes corresponding to $v \notin S$ is $O(h)$ (recall that this label requires $v \notin S$).

- $\underline{\mathsf{profile}(\Gamma) \text{ of components.}}$ The number is h because the number of components containing v is at most h.

- $\underline{\mathsf{profile}(v) \text{ of vertices.}}$ The number is one.

- $\underline{\mathsf{SuccTerminal}(v_{\text{oc}}, T).}$ The number is $O(h^2\Delta + h)$ because the number of (extended) Steiner trees T s.t. $V(T)$ has nodes corresponding to v is $O(h)$, and each $\mathsf{Euler}(T)$ has at most $O(h\Delta)$ occurrences owned by $v \notin S$ (recall that this label requires v_{oc} not owned by S-vertices). The vertex v will additionally store $O(h)$ such labels with $v_{\text{oc}} = \mathsf{start}(T)$ or $\mathsf{end}(T)$, because v is in at most h components, and each component Γ corresponds to two (extended) Steiner trees T_{γ} and $T_{\gamma^{\text{ext}}}$.

- $\underline{\mathsf{InnerTerminals}^{\rightarrow}_{\gamma}(v_{\text{oc}}, T_{\gamma^{\text{ext}}}).}$ The number is $O(h^2\Delta + h)$ because the number of extended Steiner trees T s.t. $V(T)$ has nodes corresponding to v is at most h, and each $\mathsf{Euler}(T)$ has at most $O(h\Delta)$ occurrences owned by $v \notin S$ (recall that this label requires v_{oc} not owned by S-vertices). The additional term $O(h)$ is due to the case $v_{\text{oc}} = \mathsf{start}(T_{\gamma^{\text{ext}}})$ or $\mathsf{end}(T_{\gamma^{\text{ext}}})$.

- $\underline{\mathsf{NeighborEdge}^{\rightarrow}_{\gamma}(v_{\mathrm{oc}},T),\ \mathsf{NeighborEdge}^{\leftarrow}_{\gamma}(v_{\mathrm{oc}},T),\ \mathsf{NeighborVertex}^{\rightarrow}_{\gamma}(v_{\mathrm{oc}},T)\text{ and }\mathsf{NeighborVertex}^{\leftarrow}_{\gamma}(v_{\mathrm{oc}},T).}$ For each of them, the number is $O((h^2\Delta + h)h)$ by the following reasons. First, the number of (extended) Steiner trees T s.t. $V(T)$ has nodes corresponding to v is $O(h)$. Second, each $\mathsf{Euler}(T)$ has at most $O(h\Delta)$ occurrences owned by $v \notin S$ (recall that this label requires v_{oc} not owned by S-vertices). The additional term $O(h^2)$ is due to the case $v_{\mathrm{oc}} = \mathsf{start}(T_{\gamma^{\mathrm{ext}}})$ or $\mathsf{end}(T_{\gamma^{\mathrm{ext}}})$. Finally, fixing v_{oc} and T, the number of eligible γ is h.

- $\underline{\mathsf{ArtificialEdge}(\gamma', \widetilde{G}_{\Gamma}).}$ The number is $O(h^2)$, because v is in at most h component Γ', and each Γ' has h ancestors.

- $\underline{\mathsf{IncidentEdge}(x, \widetilde{G}_{\Gamma}[\gamma^{\mathrm{ext}}]).}$ The number is $O(h + (h^2\Delta + h)(f/\phi + 1))$ by the following reasons. The first term h is because we store $\mathsf{IncidentEdge}(v, \widetilde{G}_{\Gamma}[\gamma^{\mathrm{ext}}])$ at v for each $\gamma^{\mathrm{ext}} \ni v$, and there are h such γ^{ext}. The second term $(h^2\Delta + h)(f/\phi + 1)$ is because we store a label $\mathsf{IncidentEdge}(x, \widetilde{G}_{\Gamma}[\gamma^{\mathrm{ext}}])$ along with each vertex in $\mathsf{InnerTerminals}^{\rightarrow}_{\gamma}(v_{\mathrm{oc}}, T_{\gamma^{\mathrm{ext}}})$. We have shown that there are $O(h^2\Delta + h)$ $\mathsf{InnerTerminals}^{\rightarrow}_{\gamma}(v_{\mathrm{oc}}, T_{\gamma^{\mathrm{ext}}})$ stored at v, and each of them has $f/\phi + 1$ vertices.

- $\underline{\mathsf{Degree}(x, \widetilde{G}_{\Gamma}[\gamma^{\mathrm{ext}}]).}$ The number is $O((h^2\Delta + h)(f/\phi + 1))$ because each $\mathsf{Degree}(x, \widetilde{G}_{\Gamma})$ is stored along with a vertex in $\mathsf{InnerTerminals}^{\rightarrow}_{\gamma}(v_{\mathrm{oc}}, T_{\gamma^{\mathrm{ext}}})$.

- $\underline{\mathsf{Enum}(\gamma', y, \widetilde{G}_{\Gamma}).}$ The number is $O(h^2\lambda_{\mathrm{nb}})$ by the following reasons. First, there are at most h components Γ' containing v. Second, each Γ' has at most h ancestor Γ. Finally, fixing Γ' and Γ, the number of eligible y is $|N_{\gamma}(\Gamma')| \leq \lambda_{\mathrm{nb}}$.

- $\underline{\mathsf{Enum}(x, y, \widetilde{G}_{\Gamma}).}$ The number is $O(h\lambda_{\mathrm{nb}})$ by the following reasons. First, v is in exactly one γ'. Second, the number of component Γ s.t. $\Gamma' \preceq \Gamma$ is at most h. Finally, fixing Γ' and Γ, the number of eligible y is $|N_{\gamma}(\Gamma')| \leq \lambda_{\mathrm{nb}}$.

- $\underline{\mathsf{PrefixEnum}^{\mathrm{inc}}_{\gamma}(v_{\mathrm{oc}}, T, y)\text{ and }\mathsf{PrefixEnum}^{\mathrm{exc}}_{\gamma}(v_{\mathrm{oc}}, T, y).}$ For each of them, the number is $O((h^2\Delta + h)h \cdot (hf\lambda_{\mathrm{nb}} + f/\phi + 1))$ by the following reasons. Take $\mathsf{PrefixEnum}^{\mathrm{inc}}_{\gamma}(v_{\mathrm{oc}}, T, y)$ as an example. Recall that we store this label along with each vertex $y \in \mathsf{NeighborVertex}^{\rightarrow}_{\gamma}(v_{\mathrm{oc}}, T)$. We have shown that there are $O((h^2\Delta + h)h)$ $\mathsf{NeighborVertex}^{\rightarrow}_{\gamma}(v_{\mathrm{oc}}, T)$ stored at v, and each of them has $hf\lambda_{\mathrm{nb}} + f/\phi + 1$ vertices.

Regarding the total space at an vertex $v \in V(G)$, observe that the bottleneck is the label $\mathsf{NeighborEdge}^{\rightarrow}_{\gamma}(v_{\mathrm{oc}}, T)$ (also $\mathsf{NeighborEdge}^{\leftarrow}_{\gamma}(v_{\mathrm{oc}}, T)$) and the label $\mathsf{IncidentEdge}(x, \widetilde{G}_{\Gamma}[\gamma^{\mathrm{ext}}])$. The former takes total bits

$$(h(f+2)+1)(h(f+2)\lambda_{\mathrm{nb}} + f/\phi + 1) \cdot O(\log n) \cdot O((h^2\Delta + h)h) = O((h^5 f^2 \lambda_{\mathrm{nb}}\Delta + h^4 f^2 \Delta/\phi)\log n),$$

and the latter takes total bits

$$O(\lambda_{\mathrm{arbo}} \log n) \cdot O(h + (h^2\Delta + h)(f/\phi + 1)) = O(h^2 f \lambda_{\mathrm{arbo}} \Delta \log n/\phi).$$

Plugging in $h = O(\log n), \Delta = O(1/\phi), \lambda_{\mathrm{nb}} = O(f \log n)$ and $\lambda_{\mathrm{arbo}} = O(f^2 \log^3 n)$, the total number of bits is $O(f^3(\log^7 n/\phi + \log^6 n/\phi^2))$.

3.4.6 The Final Labeling Scheme

Recall that the labeling scheme described above is for one $S_i \in \mathcal{S}$. In fact, our final labeling scheme will be made up of $f+1$ separated (sub-)labeling schemes for the $f+1$ groups S_i in $\mathcal{S}$. For each sub-scheme (corresponding to S_i) and each vertex $v \in V(G)$, we add the index i to the labels at v belonging to this sub-scheme (view these labels as a whole, so the index i will only be added once), so that we can locate the correct sub-scheme if we know (the index i) of a valid S_i for a query $\langle s,t,F\rangle$. Note that such indices takes $O((f+1)\log f)$ extra bits at v because each index i takes $O(\log f)$ bits and there are $f+1$ sub-schemes.

Furthermore, we store a label $\mathsf{color}(v)$ at each vertex v, where $\mathsf{color}(v)$ is the unique index i s.t. $v \in S_i$. For a query $\langle s,t,F\rangle$, to find a valid S_i (i.e. S_i is disjoint from F), it suffices to look at $\mathsf{color}(v)$ of all $v \in F$ and pick an index i different from any of such $\mathsf{color}(v)$. This label $\mathsf{color}(v)$ takes extra $O(\log f)$ bits at a vertex v.

In summary, the space of our final labeling scheme is $O(f^4(\log^7 n/\phi + \log^6/\phi^2))$. The query time is $\mathrm{poly}(f, \log n)$.

4 Randomized Edge Fault Connectivity Labels

Dory and Parter [DP21] presented two Monte Carlo labeling schemes for f edge faults. The first uses $O(f+\log n)$ bits, which is optimal for $f \le \log n$, while the second is an $O(\log^3 n)$-bit sketch based on ℓ_0-samplers, following Ahn, Guha, and McGregor [AGM12] and Kapron, King, and Mountjoy [KKM13]. In Section 4.1 we present a simpler proof of the $O(f+\log n)$-bit sketch, with a slightly faster construction time $O(m(1+f/\log n))$, rather than $O(m(f+\log n))$ [DP21], and in Section 4.2 we combine the two sketches to yield an $O(\log^2 n \log(f/\log^2 n))$-bit sketch, which improves on [DP21] whenever $f = n^{o(1)}$.

4.1 A Simple Labeling Scheme

Theorem 4.1 (Cf. Dory and Parter [DP21])**.** *Fix any undirected graph $G=(V,E)$ and integer $f \ge 1$. There are randomized labeling functions $L_V : V \to \{0,1\}^{O(\log n)}$ and $L_E : E \to \{0,1\}^{f+O(\log n)}$ such that given any query $\langle s,t,F\rangle$, $F \subset E$, $|F| \le f$, with high probability we can determine if s and t are connected in $G-F$, by inspecting the labels $L_V(s), L_V(t)$, and $\{L_E(e) \mid e \in F\}$. The labeling can be constructed in $O(m(1+f/\log n))$ time.*

Let T^* be any spanning tree of G, rooted at an arbitrary vertex $\mathsf{root}(T^*)$, and let T^*_v be the set of vertices in the subtree rooted at v. Let $L_V(v) = (\min_{u\in T^*_v} \mathsf{DFS}(u), \max_{u\in T^*_v} \mathsf{DFS}(u))$ contain the first and last DFS-numbers in the subtree rooted at v. Given $L_V(u), L_V(v)$, we can determine whether u,v have an ancestor/descendant relationship. Let $\mathsf{sk}^0 : E \to \{0,1\}^{c\log n+f}$ be a uniformly random labeling of the edges. This notation is overloaded for vertices and vertex-sets as follows.

$$\mathsf{sk}^0_{\hat{E}}(v) = \bigoplus_{\substack{e\in\hat{E}-E(T^*)\\ \text{s.t. } v\in e}} \mathsf{sk}^0(e) \qquad \text{bitwise XOR of } \hat{E}-E(T) \text{ edges incident to } v$$

$$\mathsf{sk}^0_{\hat{E}}(S) = \bigoplus_{v\in S} \mathsf{sk}^0_{\hat{E}}(v) \qquad \text{for } S \subset V.$$

Definition 4.2 (Edge Fault Tolerant Labels for Theorem 4.1)**.** Fix any edge $e = \{u, v\} \in E$. The label $L_E(e)$ contains

- $L_V(u), L_V(v)$, and a bit indicating whether $e \in E(T^*)$.
- Either $\mathsf{sk}^0(e)$, if $e \notin T^*$, or $\mathsf{sk}^0_E(T^*_v)$, if $e \in E(T^*)$ with v being the child of u in T^*.

Note that since $\mathsf{sk}^0(e) \oplus \mathsf{sk}^0(e) = \mathbf{0}$, this last component of $L_E(\{u, v\})$ is the XOR of all sk^0-labels of edges crossing the cut $(T^*_v, V - T^*_v)$. As a special case, $\mathsf{sk}^0_E(T^*_{\mathsf{root}(T^*)}) = \mathsf{sk}^0_E(V) = \mathbf{0}$.[8]

Observation 4.3 (Homomorphism from Sets to Sketches)**.** *We also let $\oplus$ be the symmetric difference of sets, i.e., $A \oplus B = (A - B) \cup (B - A)$. If $A, B \subset V$, $\mathsf{sk}^0(A \oplus B) = \mathsf{sk}^0(A) \oplus \mathsf{sk}^0(B)$.*

Proof of Theorem 4.1. To answer a query $\langle s, t, F\rangle$ we first identify those tree edges $F \cap E(T^*) = \{e_1, \ldots, e_{f_0}\}$, and let $T^*_0, \ldots, T^*_{f_0}$ be the connected components of $T^* - F$. We then compute $\mathsf{sk}^0_E(T^*_i)$ for all $i \in [0, f_0]$ as follows. Suppose the deleted tree edges incident to T^*_i are $F_i = \{\{u_1, v_1\}, \ldots, \{u_t, v_t\}\}$, with v_j the child of u_j. We claim that $\mathsf{sk}^0_E(T^*_i)$ is

$$\mathsf{sk}^0_E(T^*_i) = \bigoplus_{j\in[t]} \mathsf{sk}^0_E(T^*_{v_j}),$$

which can be calculated from the labels of F. If T^*_i is rooted at v_1, then

$$T^*_i = T^*_{v_1} - (T^*_{v_2} \cup \cdots \cup T^*_{v_t}) = \bigoplus_{j\in[t]} T^*_{v_j},$$

and if T^*_i is rooted at $\mathsf{root}(T^*)$, then $T^*_i = \overline{T^*_{v_1} \cup \cdots \cup T^*_{v_t}} = \overline{\bigoplus_{j\in[t]} T^*_{v_j}}$. Correctness follows from Observation 4.3, and the fact that $\mathsf{sk}^0(S) = \mathsf{sk}^0(S) \oplus \mathbf{0} = \mathsf{sk}^0(S) \oplus \mathsf{sk}^0(V) = \mathsf{sk}^0(\overline{S})$, for any $S \subset V$. See Fig. 6.

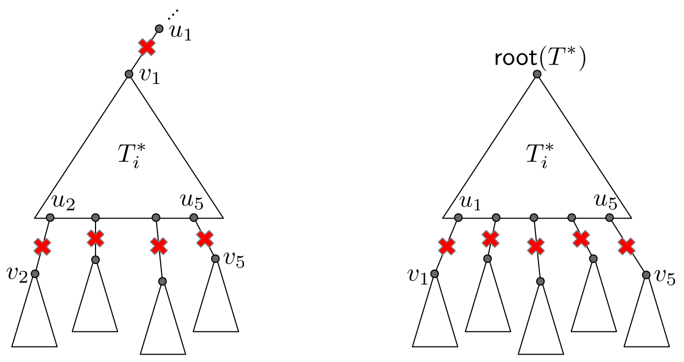


Figure 6:

We can then delete the contribution of edges from $F - E(T^*)$ by setting

$$\mathsf{sk}^0_{E-F}(T^*_i) = \mathsf{sk}^0_E(T^*_i) \oplus \bigoplus_{\substack{\{u,v\}\in F-E(T^*)\\ u\in T^*_i, v\notin T^*_i}} \mathsf{sk}^0(\{u, v\}).$$

[8]Here $\mathbf{0}$ refers to a zero-vector of the appropriate length.

Consider a set $S \subset V$, which is the union of some strict subset of $\{T_0^*, \ldots, T_{f_0}^*\}$. Then $\mathsf{sk}_{E-F}^0(S) = \bigoplus_{T_i^* \subseteq S} \mathsf{sk}_{E-F}^0(T_i^*)$ is the XOR of the sk^0-labels of edges crossing the cut $(S, \overline{S})$. Thus, if S is the union of some connected components in $G - F$, then $\mathsf{sk}_{E-F}^0(S) = \mathbf{0}$. The converse is true with high probability, since the expected number of *false positive* zeros (over all non-trivial partitions of $\{T_0^*, \ldots, T_{f_0}^*\}$) is $(2^{f_0} - 1)2^{-(c \log n + f)} < n^{-c}$.

Using Gaussian elimination, we find a subset $I \subset [f_0+1]$ for which $S = \bigcup_{i \in I} T_i^*$ and $\mathsf{sk}_{E-F}(S) = \mathbf{0}$, then recursively look for more such subsets in I and $[f_0 + 1] - I$. The leaves of this recursion tree enumerate all connected components of $G - F$, assuming no false positives. We can then answer connectivity queries w.r.t. $G - F$ in $O(\min\{\frac{\log\log n}{\log\log\log n}, \frac{\log f}{\log\log n}\})$ time using predecessor search [PT06, PT14] over the set of DFS-numbers of endpoints of edges in $F \cap E(T^*)$.

Dory and Parter's [DP21] preprocessing algorithm takes linear time $O(m)$ to generate each bit of the labeling, or $O(m(f + \log n))$ time in total. Assuming a machine with $(\log n)$-bit words, the random edge labels $\{\mathsf{sk}^0(e) \mid e \in E\}$ can be generated in $O(m(1 + f/\log n))$ time. It takes $O(m(1+f/\log n))$ to form $\{\mathsf{sk}^0(v) \mid v \in V\}$, then another $O(n(1+f/\log n))$ to generate $\{\mathsf{sk}^0(T_v^*) \mid v \in V\}$ with a postorder traversal of T^*. $\square$

4.2 A Smaller Labeling Scheme

Theorem 4.4. *Fix any undirected graph $G = (V, E)$ and integer $f \geq 2\log^2 n$. There are randomized labeling functions $L_V : V \to \{0,1\}^{O(\log n)}$ and $L_E : E \to \{0,1\}^{O(\log^2 n \log(f/\log^2 n))}$ such that given any query $\langle s, t, F\rangle$, $F \subset E$, $|F| \leq f$, with high probability one can determine whether s and t are connected in $G - F$ by inspecting only $L_V(s), L_V(t), \{L_E(e) \mid e \in F\}$.*

The vertex labeling function L_V of Theorem 4.4 is the same as Theorem 4.1; only the edge-labels will be different. As in [DP21], the edge-label sketches contain the names of a set of edges XORed together. We need to be able to decide (w.h.p.) when this set has cardinality zero, one, or greater than one. Lemma 4.5 improves the seed-length of the singleton-detection schemes of Ghaffari and Parter [GP16] and Gibb, Kapron, King, and Thorn [GKKT15] from $O(\log^2 n)$ bits to $O(\log n)$ bits. See Appendix B for proof.

Lemma 4.5. *There are functions $\mathsf{uid} : \{0,1\}^{O(\log n)} \times E \to \{0,1\}^{O(\log n)}$ and $\mathsf{singleton} : \{0,1\}^{O(\log n)} \times \{0,1\}^{O(\log n)} \to \{0,1\}^{O(\log n)} \cup \{\perp\}$ with the following properties.*

- *Given $\mathsf{uid}_s(e)$ where $e = \{u, v\}$ and any seed $s \in \{0,1\}^{O(\log n)}$, we can recover $L_V(u), L_V(v)$ with probability 1.*
- *For any single-edge set $E' = \{e^*\} \subset E$ and any seed $s \in \{0,1\}^{O(\log n)}$, $\mathsf{singleton}_s(\bigoplus_{e \in E'} \mathsf{uid}_s(e)) = \mathsf{uid}_s(e^*)$ with probability 1.*
- *When $|E'| > 1$, $\Pr[\mathsf{singleton}_s(\bigoplus_{e \in E'} \mathsf{uid}_s(e)) = \perp] = 1 - 1/\mathrm{poly}(n)$. With probability $1/\mathrm{poly}(n)$, it may return a* false positive *$\mathsf{uid}(e)$, where e may or may not be in E'. (These probabilities are over the choice of the random seed s.)*

For brevity the subscript s is always omitted.

We construct an ℓ_0-sampling sketch as in [KKM13, AGM12] as follows. Let $B = \Theta(\log(f/\log^2 n))$. For each $i \in [B]$, $\mathsf{rank}_i : E \to \mathbb{Z}^+$ is a random rank assignment such that $\Pr(\mathsf{rank}_i(e) = j) = 2^{-j}$, independent of other edges. Define $\mathsf{sk}(e)$ to be a $B \times \log m$ matrix where

$$\mathsf{sk}(e)[i,j] = \begin{cases} \mathsf{uid}(e) & \text{if } \mathsf{rank}_i(e) = j, \\ \mathbf{0} & \text{otherwise.} \end{cases}$$

Overloading the notation to vertices and vertex sets,

$$\mathsf{sk}_{\hat{E}}(v) = \bigoplus_{\substack{e \in \hat{E} - E(T) \\ \text{s.t. } v \in e}} \mathsf{sk}_{\hat{E}}(e),$$
$$\mathsf{sk}_{\hat{E}}(S) = \bigoplus_{v \in S} \mathsf{sk}_{\hat{E}}(v).$$

Here $\oplus$ is applied entrywise to the sketch array.

Definition 4.6 (Edge Fault Tolerant Labels for Theorem 4.4)**.** The label $L_E(e)$, $e = \{u,v\}$ has bit-length $O(\log^2 n \log(f/\log^2 n))$. It consists of:

- The random $O(\log n)$-bit seed s.
- The sketch from Theorem 4.1, where $\mathsf{sk}^0 : E \to \{0,1\}^{\log^2 n}$ assigns $\log^2 n$-bit labels, independent of f. Specifically, it includes $L_V(u), L_V(v)$, and either $\mathsf{sk}^0(e)$, if $e \notin T^*$, or $\mathsf{sk}^0(T^*(v))$, where v is the child of u, if $e \in T^*$.
- Either $\mathsf{sk}(e)$, if $e \notin T^*$, or $\mathsf{sk}_E(T^*_v)$, if $e = \{u,v\} \in T^*$, where v is the child of u in T^*.

Theorem 4.4 is proved in the remainder of this section.

Consider a query $\langle s,t,F\rangle$. Removing the faulty tree edges $\{e_1,\ldots,e_{f_0}\} = F \cap E(T^*)$ results in a set of trees $\{T^*_0,\ldots,T^*_{f_0}\}$. Suppose the deleted tree edges incident to T^*_i are $\{\{u_1,v_1\},\ldots,\{u_t,v_t\}\}$, with v_j the child of u_j, and let $F_i \subset F$ be the set of non-tree edges with exactly one endpoint in T^*_i. Then

$$\mathsf{sk}_{E-F}(T^*_i) = \bigoplus_{j\in[t]} \mathsf{sk}_E(T^*_{v_j}) \oplus \bigoplus_{e \in F_i} \mathsf{sk}(e), \qquad \text{and} \qquad \mathsf{sk}^0_{E-F}(T^*_i) = \bigoplus_{j\in[t]} \mathsf{sk}^0_E(T^*_{v_j}) \oplus \bigoplus_{e \in F_i} \mathsf{sk}^0(e),$$

which can be computed directly from the labels $\{L_E(e) \mid e \in F\}$.

At this point we run $B = O(\log(f/\log^2 n))$ probabilistic *Borůvka steps.* We begin with the partition $\mathcal{P}_0 = \{T^*_0,\ldots,T^*_{f_0}\}$ of V and maintain the loop invariant that after i Borůvka steps, for each part $P \in \mathcal{P}_i$, we have the sketches $\mathsf{sk}_{E-F}(P)$ and $\mathsf{sk}^0_{E-F}(P)$. Here $\mathcal{P}_i$ is a coarsening of $\mathcal{P}_{i-1}$.

In the $(i+1)$th Borůvka step we attempt, for each $P \in \mathcal{P}_i$, to extract from $\mathsf{sk}_{E-F}(P)$ the $\mathsf{uid}(e)$ of an edge $e = \{u,u'\}$, with $u \in P$ and $u' \in P' \neq P$, then unify P and P' in $\mathcal{P}_{i+1}$.

Lemma 4.7 (Cut Sketch [AGM12, KKM13])**.** *For any i and $P \in \mathcal{P}_i$, with constant probability there exists a j such that $\mathsf{singleton}(\mathsf{sk}_{E-F}(P)[i,j]) = \mathsf{uid}(e)$. Conditioned on $\mathsf{singleton}$ returning a $\mathsf{uid}(e)$, $e \in E - F$ is an edge crossing the cut $(P, \overline{P})$, with probability $1 - 1/\mathrm{poly}(n)$.*

Proof. Suppose the number of edges in $E - F$ crossing the cut $(P,\overline{P})$ is in the range $[2^{j-1}, 2^j)$, then with constant probability there is exactly one such e with $\mathsf{rank}_i(e) = j$, in which case $\mathsf{singleton}(\mathsf{sk}_{E-F}(P)[i,j]) = \mathsf{uid}(e)$. By Lemma 4.5, the probability that $\mathsf{singleton}(\mathsf{sk}_{E-F}(P)[i,j'])$ returns a false positive, for any $j' \in [\log m]$, is $\log m/\mathrm{poly}(n) = 1/\mathrm{poly}(n)$. $\square$

Let $X_P \in \{0,1\}$ be an indicator for the event that $\mathsf{sk}_{E-F}(P)$ reports a valid edge in the $(i+1)$th Borůvka step. If the $\{X_P\}_{P\in\mathcal{P}_i}$ were independent then Chernoff-Hoeffding bounds would imply that $\sum_P X_P$ is concentrated around its expectation, meaning the number of non-isolated parts in the partitions would drop by a constant factor, w.h.p., so long as there are $\Omega(\log n)$ non-isolated parts. However, the $\{X_P\}$ are *not independent*, so we require a more careful analysis.

Lemma 4.8. *Let $\mathcal{P}_i^* \subset \mathcal{P}_i$ be the parts that are not already connected components of $G-F$. With probability $1-\exp(-\Omega(|\mathcal{P}_i^*|))$, $|\mathcal{P}_{i+1}^*| < 0.94|\mathcal{P}_i^*|$.*

Before proving Lemma 4.8 let us briefly explain how queries are handled. We use the sk-sketches to implement $B = O(\log(f/\log^2 n))$ Borůvka steps. The success of these steps are independent, as step i only uses $\mathsf{sk}_{E-F}(P)[i,\cdot]$, which depends only on rank_i. Lemma 4.8 guarantees that the number of non-isolated components drops by a constant factor in each step, hence after B Borůvka steps the number of non-isolated parts is at most $f' = (\log^2 n)/2$ with probability $1-\exp(-\Omega(\log^2 n))$.

We declare P *isolated* if $\mathsf{sk}^0_{E-F}(P) = \mathbf{0}$, then determine the connected components of the remaining components using Gaussian elimination on the sk^0-sketches $\{\mathsf{sk}^0_{E-F}(P) \mid P \in \mathcal{P}_B, P \text{ non-isolated}\}$, exactly as in Theorem 4.4. This last step succeeds with probability $1-\exp(-\Omega(\log^2 n))$ as sk^0 assigns edge labels with $\log^2 n = f' + (\log^2 n)/2$ bits.

The proof of Lemma 4.8 uses the following martingale concentration inequality.

Theorem 4.9 (See [DP09] or [CL06]). *Let $f = f(\mathbf{Y}_m)$ be some function of independent random variables $\mathbf{Y}_m = (Y_1,\ldots,Y_m)$. Define $\Delta_i = \mathbb{E}(f \mid \mathbf{Y}_i) - \mathbb{E}(f \mid \mathbf{Y}_{i-1})$, $v_i = \limsup_{\mathbf{Y}_{i-1}} \mathbb{V}(\Delta_i \mid \mathbf{Y}_{i-1})$, $V = \sum_i v_i$, and M be such that $\forall i. \Delta_i \le M$. Then for any $\lambda > 0$,*

$$\Pr(\mathbb{E}(f) - f \ge \lambda), \Pr(f - \mathbb{E}(f) \ge \lambda) \le \exp\left(-\frac{\lambda^2}{2(V + \lambda M/3)}\right).$$

Proof of Lemma 4.8. We order the edges in $E-F$ with endpoints in distinct components arbitrarily as $e_1,\ldots,e_m$ and let $Y_j = \mathsf{rank}_i(e_j)$. Define $f(\mathbf{Y}_m) = \sum_{P\in\mathcal{P}_i} X_P$, where X_P is an indicator for the event that *exactly* one edge e incident to P has $\mathsf{rank}_i(e) \ge \lfloor \log \mathsf{Deg}(P) \rfloor + 1$, where $\mathsf{Deg}(P)$ is the number of $E-F$ edges with exactly one endpoint in P. Clearly f is an *underestimate* for the number of components P that isolate a single incident edge in $\mathsf{sk}_{E-F}(P)[i,\cdot]$.

For $p = \Pr(\mathsf{rank}_i(e) \ge \lfloor \log \mathsf{Deg}(P) \rfloor + 1) = 2^{-\lfloor \log \mathsf{Deg}(P) \rfloor}$, we have $\mathbb{E}(X_P) = \mathsf{Deg}(P)p(1-p)^{\mathsf{Deg}(P)-1} \ge \mathsf{Deg}(P)pe^{-\mathsf{Deg}(P)p}$. In the interval $[1,2)$ this is minimized when $\mathsf{Deg}(P)p \to 2$, so $\mathbb{E}(X_P) \ge 2e^{-2} > 0.27$.

Suppose e_j joins parts $P, P' \in \mathcal{P}_i$. Note that revealing $Y_j = \mathsf{rank}_i(e_j)$ can only change the conditional probability of X_P and $X_{P'}$. In particular, $\Delta_j \le M \overset{\text{def}}{=} 2$ and

$$\begin{aligned}\mathbb{V}(\Delta_j \mid \mathbf{Y}_{j-1}) = \mathbb{E}((X_P - \mathbb{E}(X_P \mid \mathbf{Y}_{j-1}))^2 \mid \mathbf{Y}_{j-1}) + \mathbb{E}((X_{P'} - \mathbb{E}(X_{P'} \mid \mathbf{Y}_{j-1}))^2 \mid \mathbf{Y}_{j-1}) \\ + 2\mathbb{E}((X_P - \mathbb{E}(X_P \mid \mathbf{Y}_{j-1}))(X_{P'} - \mathbb{E}(X_{P'} \mid \mathbf{Y}_{j-1})) \mid \mathbf{Y}_{j-1}),\end{aligned} \tag{3}$$

where the expectations are over choice of Y_j. Suppose that $\mathbf{Y}_{j-1}$ reveals the levels of all but g edges incident to P, and among those revealed, b are at level at least $\lfloor \log \mathsf{Deg}(P) \rfloor + 1$.

Case $b \ge 2$. Then it is already known that $X_P = 0$.

Case $b = 1$. Then $X_P = 1$ iff the remaining g edges choose levels at most $\lfloor \log \mathsf{Deg}(P) \rfloor$.

$$\begin{aligned}
&\mathbb{E}((X_P - \mathbb{E}(X_P \mid \mathbf{Y}_{j-1}))^2 \mid \mathbf{Y}_{j-1}) \\
&= p \cdot (0 - (1-p)^g)^2 + (1-p) \cdot ((1-p)^{g-1} - (1-p)^g)^2 \\
&\leq p(1-p)^2 + (1-p)p^2 = p(1-p) \qquad \text{(maximized at } g = 1.)
\end{aligned}$$

Case $b = 0$. Then $X_P = 1$ iff exactly one of the remaining g edges chooses a level at least $\lfloor \log \mathsf{Deg}(P) \rfloor + 1$.

$$\begin{aligned}
&\mathbb{E}((X_P - \mathbb{E}(X_P \mid \mathbf{Y}_{j-1}))^2 \mid \mathbf{Y}_{j-1}) \\
&= p \cdot ((1-p)^{g-1} - gp(1-p)^{g-1})^2 + (1-p) \cdot ((g-1)p(1-p)^{g-2} - gp(1-p)^{g-1})^2 \\
&\leq p \cdot (1-p)^2 + (1-p)p^2 = p(1-p) \qquad \text{(maximized at } g = 1.)
\end{aligned}$$

Let $p = 2^{-\lfloor \log \mathsf{Deg}(P) \rfloor}$ and $q = 2^{-\lfloor \log \mathsf{Deg}(P') \rfloor}$ where $q \leq p$. From the calculations above, which are maximized at $g = 1$, we can upper bound the last term of Eq. (3) as follows. Note: $Y_j \in [1, \lfloor \log \mathsf{Deg}(P) \rfloor]$, $(\lfloor \log \mathsf{Deg}(P) \rfloor, \lfloor \log \mathsf{Deg}(P') \rfloor]$, and $(\lfloor \log \mathsf{Deg}(P') \rfloor, \infty)$ with probability $1-p, p-q$, and q, respectively.

$$\begin{aligned}
&\mathbb{E}((X_P - \mathbb{E}(X_P \mid \mathbf{Y}_{j-1}))(X_{P'} - \mathbb{E}(X_{P'} \mid \mathbf{Y}_{j-1})) \mid \mathbf{Y}_{j-1}) \\
&= q(1-p)(1-q) + (p-q)(1-p)q + (1-p)pq \\
&= (1-p)q(1-2q+2p)
\end{aligned}$$

and therefore

$$\begin{aligned}
\mathbb{V}(\Delta_j \mid \mathbf{Y}_{j-1}) &\leq p(1-p) + q(1-q) + 2(1-p)q(1-2q+2p) \\
&< 4p.
\end{aligned}$$

In other words, $\mathbb{V}(\Delta_j \mid \mathbf{Y}_{j-1}) < 4/2^{\lfloor \min\{\log \mathsf{Deg}(P), \log \mathsf{Deg}(P')\} \rfloor} < 8/\min\{\mathsf{Deg}(P), \mathsf{Deg}(P')\}$, and therefore $V = \sum_j \mathbb{V}(\Delta_j \mid \mathbf{Y}_{j-1}) < 8|\mathcal{P}_i^*|$. By Theorem 4.9,

$$\Pr(f < \mathbb{E}(f) - \lambda) < \exp\left(-\frac{\lambda^2}{2(V + \lambda M/3)}\right) = \exp\left(-\frac{\lambda^2}{16|\mathcal{P}_i^*| + 4\lambda/3}\right).$$

Setting $\lambda = \mathbb{E}(f)/2 > 0.13|\mathcal{P}_i^*|$, we conclude that $\Pr(f \geq \mathbb{E}(f)/2 > 0.13|\mathcal{P}_i^*|) > 1 - \exp(-\Omega(|\mathcal{P}_i^*|))$. This implies the number of *distinct* edges reported in the ith Borůvka step is at least $0.13|\mathcal{P}_i^*|/2 = 0.065|\mathcal{P}_i^*|$, as each edge can be reported twice, by either endpoint. Hence $|\mathcal{P}_{i+1}^*| \leq 0.935|\mathcal{P}_i^*|$. □

5 Randomized Vertex Fault Connectivity Labels

We improve the size of labeling scheme under f vertex faults from $\tilde{O}(f^3)$ [PPP24] to $\tilde{O}(f^2)$ bits.

Theorem 5.1. *Fix any undirected graph $G = (V, E)$ and integer $f \geq 1$. There are randomized labeling functions $L_V : V \to \{0,1\}^{O(f^2 \log^6 n)}$ such that given any query $\langle s, t, F \rangle$, $F \subset V$, $|F| \leq f$, with high probability one can determine whether s and t are connected in $G - F$ by inspecting only $L_V(s), L_V(t), \{L_V(v) \mid v \in F\}$.*

Our scheme is a small modification to that of Parter, Petruschka, and Pettie [PPP24]. Fix a collection of vertex sets $N_1, N_2, \dots, N_k \subset V$ where $k \leq n$.[9] We say that a *neighborhood hitter* $S \subset V$ is *good* for a fault set $F \subset V$ of size $|F| \leq f$ if

$$S \cap F = \emptyset \text{ and } S \cap N_j \neq \emptyset \text{ for all } N_j \text{ where } |N_j| \geq cf \log n,$$

where c is a constant. A collection of neighborhood hitters $\mathcal{S} = \{S_1, \dots, S_s\}$ is good for F if there exists $S_i \in \mathcal{S}$ that is good for F. Parter et al. [PPP24] reduces the labeling problem to constructing a collection of neighborhood hitters as follows.

Lemma 5.2 (Section 5.2 of [PPP24])**.** *Suppose we can construct a collection of neighborhood hitters $\mathcal{S} = \{S_1, \dots, S_s\}$ such that, for each fault set F of size at most $|F| \leq f$, $\mathcal{S}$ is good for F with high probability, then there exists a randomized vertex labeling of size $O(sf^2 \log^5 n)$ satisfying the guarantee in the setting of Theorem 5.1.*

We construct $\mathcal{S} = \{S_1, \dots, S_s\}$ as follows. Set $s = O(c \log n)$. For each i, sample each vertex into S_i with probability $1/f$. Observe that S_i is good for F with constant probability. Indeed,

$$\Pr[S_i \cap F = \emptyset] = (1 - 1/f)^{|F|} \geq \Omega(1).$$

Also, for each N_j where $|N_j| \geq cf \log n$,

$$\Pr[S_i \cap N_j = \emptyset] = (1 - 1/f)^{|N_j|} \leq 1/n^{\Omega(c)}.$$

Thus, by a union bound, $\Pr[\exists j \text{ s.t. } S_i \cap N_j = \emptyset] \leq n^{-\Omega(c)}$. Hence, for any fixed F, $\mathcal{S}$ is not good for F with probability at most $(1 - \Omega(1))^s \leq n^{-\Omega(c)}$. By plugging $\mathcal{S}$ into Lemma 5.2, we obtain Theorem 5.1.[10]

6 Lower Bound for Global Connectivity under Vertex Faults

In this section, we show that any vertex labeling scheme that supports *global* connectivity queries under f vertex faults requires $\Omega(n^{1-1/f}/f)$ bits. This improves on an $\Omega(\min\{n/f, 4^f/f^{3/2}\})$ lower bound of Parter et al. [PPP24], and gives a negative answer to the open problem by [PPP24], which asks for an $\tilde{O}(1)$-size labeling scheme for global connectivity queries when $f = O(1)$. In contrast, under edge faults it is easy to answer global connectivity queries with previous schemes [DP21, IEWM23] or Theorems 2.1 and 4.4.

The lower bound is stated below. The proof closely follows [PPP24, Theorem 9.2] but with different parameters.

Theorem 6.1. *Let $L : V \to \{0,1\}^b$ be a b-bit vertex labeling scheme such that, given $\{L(v) \mid v \in F\}$ where $|F| \leq f$, reports whether $G - F$ is still connected. Then $b = \Omega(n^{1-1/f}/f)$.*

[9] In [PPP24], each $N_j = N(\Gamma_j)$ is a neighborhood for some component Γ_j in the low-degree hierarchy, which is deterministically constructed and fixed. Here, we can think of each N_j as some arbitrary vertex set.

[10] Parter et al. [PPP24] gave the same construction of $\mathcal{S}$, but set $s = f + 1$ so that the criterion that there exists $S_i \in \mathcal{S}$ with $S_i \cap F = \emptyset$ holds with probability 1, which is necessary for a *deterministic* labeling scheme. Our observation is simply that in a Monte Carlo labeling scheme, we only need this property to hold with high probability.

Proof. First, construct a "base" bipartite graph $G_0 = (L \cup R, E_0)$ as follows. Set $L = \{v^*\} \cup \{v_1, \dots, v_n\}$ and $R = \{u_1, \dots, u_r\}$ where $r = \lceil fn^{1/f} \rceil$. Connect v^* to all vertices in R. For each $i \in [n]$, $F_i \subseteq R$ is a neighbor set of v_i where $|F_i| = f$. We make sure that the $\{F_i\}$ are all distinct, i.e., $F_i \neq F_j$ for all $i \neq j \in [n]$. This is possible because $\binom{r}{f} \geq (r/f)^f \geq n$ by the choice of r. Finally, we create a family $\mathcal{G}$ of 2^n graphs from a fixed graph G_0 as follows: For each v_i, we can choose to add new edges into G_0 so that v_i is connected to all vertices in R.

Suppose an unknown graph G is promised to be from $\mathcal{G}$. Observe that $G - F_i$ is disconnected if and only if we did not add new edges incident to v_i into G_0. So, by reading the labels on F_i, we can check what choice was made for v_i. Thus, the labels of all vertices in R can determine $G \in \mathcal{G}$. Thus, $2^{rb} \geq |\mathcal{G}|$, implying that $b \geq n / \lceil fn^{1/f} \rceil$. □

7 Conclusion and Open Problems

In this paper we gave improved constructions of *expander hierarchies* (w.r.t. both vertex and edge cuts) and developed shorter *labeling schemes* for f-fault connectivity queries, in all four quadrants of {vertex faults, edge faults} × {randomized, deterministic}.

Our deterministic labeling scheme for edge faults has size $\tilde{O}(\sqrt{f})$, but it is not clear that there must be a polynomial dependence on f.

Open Question 7.1. *Is there a deterministic, f-edge-fault connectivity labeling scheme with $\tilde{O}(1)$-bit labels?*

In the case of randomized edge-labeling schemes, we improved Dory and Parter [DP21] from $O(\log^3 n)$ to $O(\log^2 n \log f)$. An interesting problem is to prove a non-trivial lower bound on edge labels, randomized or not. There are natural targets around $f = \omega(\log n)$ and $f = \mathrm{poly}(n)$.

Open Question 7.2. *Concerning the f-edge fault connectivity problem:*

- *When $f = \omega(\log n)$, are $\omega(\log n)$-bit edge labels necessary? (See [DP21] and Theorem 4.1.)*
- *When $f = \mathrm{poly}(n)$, are $\Theta(\log^3 n)$-bit edge labels optimal? (See $\Omega(\log^3 n)$-bit lower bounds of Nelson and Yu [NY19] and Yu [Yu21] for similar problems.)*

We actually know *how* to improve the randomized label length to $O(\log n \log^2(f \log n))$ *if* the following conjecture holds.

Conjecture 7.3. *Given a graph G and spanning tree T, define $\mathsf{cutsize}_T(e)$ to be 0 if $e \notin T$ and the number of (non-tree) edges crossing the cut defined by $T - e$ otherwise. For any $c_0 > 0$, there exists a $c_1 > 0$, such that for any graph G and integer $f \gg \log n$, there is a distribution $\mathcal{T}$ of its spanning trees such that for any $F = \{e_1, \dots, e_f\} \subset E$,*

$$\Pr_{T \sim \mathcal{T}}(|\{i \in [f] : \mathsf{cutsize}_T(e_i) > (f \log n)^{c_1}\}| \geq c_1 \log n) < n^{-c_0}.$$

If Conjecture 7.3 were true, we would pick the spanning tree $T \sim \mathcal{T}$ used in the labeling scheme of Theorem 4.4. Since, with probability $1 - n^{c_0}$, the cut sizes for all but $O(\log n)$ trees of $T - F$ would be bounded by $f(f \log n)^{c_1}$, we would only need to store $O(\log(f \log n))$ rows in the sketch-matrix sk, rather than $O(\log n)$, in order to implement Borůvka steps. The remaining $c_1 \log n$ trees with large cut-sizes would be handled using the sk^0 sketch, just as we handle the residual trees in Theorem 4.4. Using *spanning* trees [AN19] in Räcke's tree distribution $\mathcal{R}$ [Räc08] guarantees that for any $e \in E(G)$, $\Pr_{T\sim\mathcal{R}}(\mathsf{cutsize}_T(e) > f\text{poly}(\log n)) < 1/f$. Conjecture 7.3 can be viewed as asserting that there is a distribution where these events are sufficiently independent, for any $F \subset E(G)$ with $|F| = f$, so that we can get a Chernoff-like tail bound on the event $|\{i \in [f] : \mathsf{cutsize}_T(e_i) > (f \log n)^{c_1}\}| \geq c_1 \log n$.

The state-of-the-art for connectivity labels under vertex faults are now $\tilde{O}(f^2)$ for randomized schemes, by Theorem 5.1, and $\tilde{O}(f^4)$ for deterministic schemes, by Theorem 3.1, while there is a simple lower bound of $\Omega(f + \log n)$ [PPP24]. We believe the correct exponent is likely 2, but any non-trivial lower bound would be welcome.

Open Question 7.4. *Is there an $\Omega(f^{1.1})$ lower bound for f-vertex fault connectivity labels? What is the optimal exponent?*

Theorem 6.1 strongly separated pairwise connectivity and global connectivity for vertex-labeling schemes under vertex faults. Can we match the lower bound for global connectivity?

Open Question 7.5. *Show a labeling scheme for global connectivity under vertex faults whose size matches the lower bound by Theorem 6.1.*

The (edge) expander hierarchy of Theorem 2.3 improves the expansion of Pătraşcu and Thorup's [PT07] by a $\Theta(\log n)$-factor. The $O(f \log^2 n \log\log n)$ query time of their f-edge-failure connectivity oracle contains some $O(\log^2 n)$ terms unrelated to the expander hierarchy parameters, so it may be worth revisiting the complexity of this problem in the deterministic setting. See [DP20, GKKT15] for smaller and faster randomized data structures.

Acknowledgments

Yaowei Long and Thatchaphol Saranurak are partially funded by the Ministry of Education and Science of Bulgaria's support for INSAIT, Sofia University "St. Kliment Ohridski" as part of the Bulgarian National Roadmap for Research Infrastructure.

A Low-Degree Steiner Trees Spanning Tough Sets

In this section, we prove that there exists a $O(1/\phi)$-degree Steiner tree spanning any ϕ-vertex-expanding set.

Lemma 3.6 (Low-degree Steiner Trees)**.** *Given a graph G such that a set $A \subseteq V(G)$ is ϕ-vertex-expanding in G, there is an algorithm that computes an $O(1/\phi)$-degree Steiner tree that spans A in G. The running time is $O(mn \log n)$.*

In fact, we will show that the statement holds even for *ϕ-tough sets*, which we define now. For any graph G and vertex set $X \subseteq V(G)$, let $c_G(X)$ count the number of connected components C in G containing some vertex of X. In particular, if G is connected, then $c_G(X) = 1$ for every X. We say that X is *ϕ-tough* in G if, for every vertex set $S \subseteq V(G)$, we have

$$|S| \geq \phi \cdot c_{G-S}(X)$$

whenever $c_{G-S}(X) > 1$, i.e., X is not connected in $G - S$.

We say that G has *toughness* $\tau(G) = \phi$ if $V(G)$ is ϕ-tough. The toughness of graphs is a well-studied measure of the robustness of graphs with many connections to other graph properties (see, e.g., [Chv73, EJKS85, Win89, BBS06, Gu21]). Most literature considers the toughness of $V(G)$, but here, we will focus on the toughness of an arbitrary vertex subset X.

Observe that any vertex-expanding set is tough with the same parameter to up a constant.

Fact A.1. *If X is 3ϕ-vertex-expanding in G, then X is ϕ-tough.*

Proof. Suppose that X is not ϕ-tough, i.e. there exists S where $|S| < \phi \cdot c_{G-S}(X)$. Let $c = c_{G-S}(X)$ and $C_1, \dots, C_c$ be different connected components in $G - S$ where $|C_i \cap X| > 0$ for all i. Let C' be the union of other connected components in $G - S$ disjoint from X. Let $L = C_1 \cup \dots \cup C_{\lceil c/2 \rceil}$ and $R = C_{\lceil c/2 \rceil + 1} \cup \dots \cup C_c \cup C'$. We have that $|X \cap L| \geq \lceil c/2 \rceil \geq c/3, |X \cap R| \geq c - \lceil c/2 \rceil c/3$. So

$$|S| < \phi c \leq 3\phi \min\{|X \cap (L \cup S)|, |X \cap (R \cup S)|\},$$

meaning that X is not 3ϕ-vertex-expanding. □

The following theorem stating that there exists a $O(1/\phi)$-degree Steiner tree spanning any ϕ-tough set immediately implies Lemma 3.6.

Theorem A.2. *Given a graph G such that a set $X \subseteq V(G)$ is ϕ-tough in G, there is an algorithm that computes a $(2/\phi + 3)$-degree Steiner tree spanning X. The running time is $O(mn \log n)$.*

The weaker statement of this theorem was shown by Win [Win89], who gave a non-algorithmic version of this theorem when $A = V$, i.e., a *spanning* tree case.

To prove Theorem A.2, we apply the additive-1 approximation algorithm by [FR94] for finding a minimum degree Steiner trees. The structural guarantees of their algorithm can be summarized as follows.

Lemma A.3 ([FR94])**.** *There is an algorithm that, given a graph G with n vertices and m edges and a vertex X, in $O(mn \log n)$ time returns a tree T in G with maximum degree Δ and a vertex set $B \subseteq V(T)$ such that*

1. *Every leaf of T is a vertex in X,*
2. *Each vertex $v \in B$ has degree $\mathsf{Deg}_T(v) \geq \Delta - 1$, and*
3. *For any two vertices $s, t \in X - B$, s and t are connected in $G - B$ if and only if they are connected in $T - B$.*

The last property says that the connectivity between vertices of X in $G - B$ and $T - B$ are preserved exactly.

Proof of Theorem A.2. We claim that $|B| < 2c_{T-B}(X)/(\Delta - 3)$. Since $c_{G-B}(X) = c_{T-B}(X)$ because of Property 3 and X is ϕ-tough, it must be that $\frac{2}{(\Delta-3)} > \phi$, meaning that $\Delta < 2/\phi + 3$. That is, T is a $(2/\phi + 3)$-degree Steiner tree spanning X as desired and T can be computed in $O(mn \log n)$ time.

Now we prove the claim. Just for analysis, consider the set E_B of all edges incident to B. If we delete *edges* in E_B, observe that $T - E_B$ contains exactly $|E_B| + 1$ connected components. We classify these connected components in $T - E_B$ into three types:

1. **(Trivial components):** Components that contain of a single vertex $v \in B$,
2. **(Internal components):** Components that contains no vertex in B or X,
3. **(Leaf components):** Components that contains no vertex in B, but contains a vertex in X.

The number trivial components is clearly $|B|$. The number of internal components is at most $|E_B|/2$. This is because each deleted edge in E_B has at most one endpoint in non-trivial components (internal or leaf components.) But, crucially, each internal component must be incident to at least 2 deleted edges (otherwise, it will contain a leaf, which is a vertex in X). The number of leaf components is precisely $c_{T-B}(X)$. Since the total number of components is $|E_B| + 1$, we have that

$$|E_B| + 1 \leq |B| + |E_B|/2 + c_{T-B}(X).$$

So,

$$c_{T-B}(X) \geq |E_B|/2 - |B| + 1 \geq |B|(\Delta - 1)/2 - |B| + 1 > |B|(\Delta - 3)/2$$

where the second inequality is because B has minimum degree $\Delta - 1$. □

B Improved Singleton-Detection Scheme: Proof of Lemma 4.5

Let a be a uniformly random odd w-bit integer and t a uniformly random w-bit integer. Thorup [Tho18] proved that the function $\mathsf{Sample} : [2^w] \to \{0,1\}$, $\mathsf{Sample}_{a,t}(x) = \mathbb{1}(ax \mod 2^w < t)$ (written `a*x < t` in C++ notation) is a *distinguisher* with probability 1/8. In other words, for any non-empty set $S \subset [2^w]$,

$$\Pr\left(\sum_{x \in S} \mathsf{Sample}_{a,t}(x) \equiv 1 \pmod 2\right) \geq 1/8.$$

We identify the edge-set E with $\{0,1\}^{2\log n}$. Suppose $\mathsf{uid} : \{0,1\}^{2\log n} \to \{0,1\}^{2\log n} \times \{0,1\}^{c\log n}$ is defined so that $\mathsf{uid}(x) = (x, \mathsf{Sig}(x))$, where the signature

$$\mathsf{Sig}(x) = \left(\mathsf{Sample}_{a_1,t_1}(x), \ldots, \mathsf{Sample}_{a_{c\log n}, t_{c\log n}(x)}\right)$$

consists of $c\log n$ independent invocations of the distinguisher. In this case uid would satisfy the singleton-detection properties of Lemma 4.5, but with a $O(\log^2 n)$-bit seed. To see why, consider an arbitrary set $S \subset E$ of edges. Let $e^* = \bigoplus_{e \in S} e$ be the XOR of the names of all edges in S. If $|S| > 2$, then the probability that we mistakenly believe S to be the singleton set $\{e^*\}$ is

$$\Pr\left(\bigoplus_{e \in S} \mathsf{uid}(e) = (e^*, \mathsf{Sig}(e^*))\right) = \Pr\left(\bigoplus_{e \in (S \oplus \{e^*\})} \mathsf{uid}(e) = (\mathbf{0}, \mathbf{0})\right) \leq (7/8)^{c\log n} = 1/\mathrm{poly}(n).$$

Like [GKKT15, GP16], this scheme uses a $O(\log^2 n)$-bit seed, but it is simpler than both of [GKKT15, Gha16]. The seed-length can be reduced to $O(\log n)$ bits by taking a random walk on an expander of length $O(\log n)$.

Theorem B.1 (Hitting Property of Random Walks; see Vadhan [Vad12, Theorem 4.17])**.** *If G is a regular digraph with spectral expansion $1-\lambda$, then for any $B \subset V(G)$ of density μ, the probability that a random walk $(v_1, \ldots, v_t)$ of $t-1$ steps in G starting at a uniformly random vertex v_1 always remains in B is*

$$\Pr\left(\bigwedge_{i\in[t]} (v_i \in B)\right) \leq (\mu + \lambda(1-\mu))^t.$$

In our case $V(G)$ corresponds to the set of possible seeds (a,t) for Sample, so $|V(G)| = \text{poly}(n)$, and $\mu = 7/8$. Whenever $\lambda < 1$, $t = O(\log n)$ suffices to reduce the error probability to $1/\text{poly}(n)$. If G is d-regular, the cost of encoding the random walk is $O(\log n) + t \log d = O(\log n)$ bits.

References

[AAK$^+$06] Serge Abiteboul, Stephen Alstrup, Haim Kaplan, Tova Milo, and Theis Rauhe. Compact labeling scheme for ancestor queries. *SIAM J. Comput.*, 35(6):1295–1309, 2006. 1

[AALOG18] Vedat Levi Alev, Nima Anari, Lap Chi Lau, and Shayan Oveis Gharan. Graph clustering using effective resistance. In *9th Innovations in Theoretical Computer Science Conference (ITCS 2018)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2018. 5

[ABR05] Stephen Alstrup, Philip Bille, and Theis Rauhe. Labeling schemes for small distances in trees. *SIAM J. Discret. Math.*, 19(2):448–462, 2005. 1

[ACGP16] Ittai Abraham, Shiri Chechik, Cyril Gavoille, and David Peleg. Forbidden-set distance labels for graphs of bounded doubling dimension. *ACM Trans. Algorithms*, 12(2):22:1–22:17, 2016. 1

[ADK17] Stephen Alstrup, Søren Dahlgaard, and Mathias Bæk Tejs Knudsen. Optimal induced universal graphs and adjacency labeling for trees. *J. ACM*, 64(4):27:1–27:22, 2017. 1

[AG11] Ittai Abraham and Cyril Gavoille. On approximate distance labels and routing schemes with affine stretch. In *Proceedings 25th International Symposium on Distributed Computing (DISC)*, pages 404–415, 2011. 1

[AGHP16a] Stephen Alstrup, Cyril Gavoille, Esben Bistrup Halvorsen, and Holger Petersen. Simpler, faster and shorter labels for distances in graphs. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 338–350, 2016. 1

[AGHP16b] Stephen Alstrup, Inge Li Gørtz, Esben Bistrup Halvorsen, and Ely Porat. Distance labeling schemes for trees. In *Proceedings 43rd International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 55 of *LIPIcs*, pages 132:1–132:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. 1

[AGM12] Kook J. Ahn, Supdipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 459–467, 2012. 44, 46, 47

[AHL14] Stephen Alstrup, Esben Bistrup Halvorsen, and Kasper Green Larsen. Near-optimal labeling schemes for nearest common ancestors. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 972–982, 2014. 1

[AKTZ19] Stephen Alstrup, Haim Kaplan, Mikkel Thorup, and Uri Zwick. Adjacency labeling schemes and induced-universal graphs. *SIAM J. Discret. Math.*, 33(1):116–137, 2019. 1

[AN19] Ittai Abraham and Ofer Neiman. Using petal-decompositions to build a low stretch spanning tree. *SIAM J. Comput.*, 48(2):227–248, 2019. 52

[ARV09] Sanjeev Arora, Satish Rao, and Umesh V. Vazirani. Expander flows, geometric embeddings and graph partitioning. *J. ACM*, 56(2), 2009. 6

[BBS06] Douglas Bauer, Hajo Broersma, and Edward Schmeichel. Toughness in graphs–a survey. *Graphs and Combinatorics*, 22:1–35, 2006. 53

[BCG$^+$22] Aviv Bar-Natan, Panagiotis Charalampopoulos, Pawel Gawrychowski, Shay Mozes, and Oren Weimann. Fault-tolerant distance labeling for planar graphs. *Theor. Comput. Sci.*, 918:48–59, 2022. 1

[BCHR20] Surender Baswana, Keerti Choudhary, Moazzam Hussain, and Liam Roditty. Approximate single-source fault tolerant shortest path. *ACM Trans. Algorithms*, 16(4):44:1–44:22, 2020. 1

[BF67] Melvin A Breuer and Jon Folkman. An unexpected result in coding the vertices of a graph. *Journal of Mathematical Analysis and Applications*, 20(3):583–600, 1967. 1

[BGP22] Marthe Bonamy, Cyril Gavoille, and Michal Pilipczuk. Shorter labeling schemes for planar graphs. *SIAM J. Discret. Math.*, 36(3):2082–2099, 2022. 1

[Bre66] Melvyl Breuer. Coding the vertexes of a graph. *IEEE Transactions on Information Theory*, 12(2):148–153, 1966. 1

[CGKT08] Bruno Courcelle, Cyril Gavoille, Mamadou Moustapha Kanté, and Andrew Twigg. Connectivity check in 3-connected planar graphs with obstacles. *Electron. Notes Discret. Math.*, 31:151–155, 2008. 1

[Cho16] Keerti Choudhary. An optimal dual fault tolerant reachability oracle. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2016. 1

[Chv73] Vasek Chvátal. Tough graphs and hamiltonian circuits. *Discret. Math.*, 5(3):215–228, 1973. 53

[CL06] Fan R. K. Chung and Lincoln Lu. Survey: Concentration inequalities and martingale inequalities: A survey. *Internet Math.*, 3(1):79–127, 2006. 48

[CMW23] Shiri Chechik, Shay Mozes, and Oren Weimann. $\tilde{O}$ptimal fault-tolerant reachability labeling in planar graphs. *arXiv preprint arXiv:2307.07222*, 2023. 1

[CPR11] Timothy M. Chan, Mihai Pătraşcu, and Liam Roditty. Dynamic connectivity: Connecting to networks and geometry. *SIAM J. Comput.*, 40(2):333–349, 2011. 20

[CT07] Bruno Courcelle and Andrew Twigg. Compact forbidden-set routing. In *Proceedings 24th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 4393 of *Lecture Notes in Computer Science*, pages 37–48. Springer, 2007. 1

[DP09] Devdatt P. Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms.* Cambridge University Press, 2009. 48

[DP20] Ran Duan and Seth Pettie. Connectivity oracles for graphs subject to vertex failures. *SIAM J. Comput.*, 49(6):1363–1396, 2020. 3, 4, 14, 17, 20, 52

[DP21] Michal Dory and Merav Parter. Fault-tolerant labeling and compact routing schemes. In *Proceedings of the 40th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 445–455, 2021. , 1, 2, 3, 4, 44, 46, 50, 51

[EJKS85] Hikoe Enomoto, Bill Jackson, Panagiotis Katerinis, and Akira Saito. Toughness and the existence of k-factors. *Journal of Graph Theory*, 9(1):87–95, 1985. 53

[FHL05] Uriel Feige, MohammadTaghi Hajiaghayi, and James R Lee. Improved approximation algorithms for minimum-weight vertex separators. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 563–572, 2005. 16

[FR94] Martin Fürer and Balaji Raghavachari. Approximating the minimum-degree Steiner tree to within one of optimal. *J. Algor.*, 17(3):409–423, 1994. 17, 53

[Gha16] Mohsen Ghaffari. An improved distributed algorithm for maximal independent set. In *Proceedings 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 270–277, 2016. 54

[GKKT15] David Gibb, Bruce M. Kapron, Valerie King, and Nolan Thorn. Dynamic graph connectivity with improved worst case update time and sublinear space. *CoRR*, abs/1509.06464, 2015. 3, 46, 52, 54

[GP16] Mohsen Ghaffari and Merav Parter. MST in log-star rounds of congested clique. In *Proceedings of the 35th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 19–28, 2016. 46, 54

[GPPR04] Cyril Gavoille, David Peleg, Stéphane Pérennes, and Ran Raz. Distance labeling in graphs. *J. Algorithms*, 53(1):85–112, 2004. 1

[GRST21] Gramoz Goranci, Harald Räcke, Thatchaphol Saranurak, and Zihan Tan. The expander hierarchy and its applications to dynamic graph algorithms. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2212–2228. SIAM, 2021. 3

[Gu21] Xiaofeng Gu. A proof of brouwer's toughness conjecture. *SIAM Journal on Discrete Mathematics*, 35(2):948–952, 2021. 53

[GU23] Pawel Gawrychowski and Przemyslaw Uznanski. Better distance labeling for unweighted planar graphs. *Algorithmica*, 85(6):1805–1823, 2023. 1

[HKNS15] Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of the 47th Annual ACM Symposium on Theory of Computing (STOC)*, pages 21–30, 2015. 3

[HL09] Tai-Hsin Hsu and Hsueh-I Lu. An optimal labeling for node connectivity. In *Proceedings of 20th International Symposium on Algorithms and Computation (ISAAC)*, volume 5878 of *Lecture Notes in Computer Science*, pages 303–310. Springer, 2009. 1

[IEWM23] Taisuke Izumi, Yuval Emek, Tadashi Wadayama, and Toshimitsu Masuzawa. Deterministic fault-tolerant connectivity labeling scheme with adaptive query processing time. In *Proceedings of the 42nd ACM Symposium on Principles of Distributed Computing (PODC)*, 2023. , 1, 2, 3, 4, 6, 50

[IN12] Rani Izsak and Zeev Nutov. A note on labeling schemes for graph connectivity. *Inf. Process. Lett.*, 112(1-2):39–43, 2012. 1

[KKKP04] Michal Katz, Nir A. Katz, Amos Korman, and David Peleg. Labeling schemes for flow and connectivity. *SIAM J. Comput.*, 34(1):23–40, 2004. 1

[KKM13] Bruce M. Kapron, Valerie King, and Ben Mountjoy. Dynamic graph connectivity in polylogarithmic worst case time. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1131–1142, 2013. 44, 46, 47

[KNR92] Sampath Kannan, Moni Naor, and Steven Rudich. Implicit representation of graphs. *SIAM J. Discret. Math.*, 5(4):596–603, 1992. 1

[Kos23] Evangelos Kosinas. Connectivity queries under vertex failures: Not optimal, but practical. In *31st Annual European Symposium on Algorithms (ESA 2023)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2023. 3

[KP21] Karthik C. S. and Merav Parter. Deterministic replacement path covering. In *Proceedings of the 32nd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 704–723, 2021. , 3, 25

[KPP16] Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3SUM conjecture. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1272–1287, 2016. 3

[LS22] Yaowei Long and Thatchaphol Saranurak. Near-optimal deterministic vertex-failure connectivity oracles. In *Proceedings 63rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 1002–1010, 2022. 3, 4, 14, 15, 17, 20

[LW24] Yaowei Long and Yunfan Wang. Better Decremental and Fully Dynamic Sensitivity Oracles for Subgraph Connectivity. In *Proceedings 51st International Colloquium on Automata, Languages, and Programming (ICALP 2024)*, volume 297 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 109:1–109:20, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. 3

[MS18] Guy Moshkovitz and Asaf Shapira. Decomposing a graph into expanding subgraphs. *Random Structures & Algorithms*, 52(1):158–178, 2018. 5

[NI92] Hiroshi Nagamochi and Toshihide Ibaraki. A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. *Algorithmica*, 7(5&6):583–596, 1992. 3, 24, 25

[NSWN17] Danupon Nanongkai, Thatchaphol Saranurak, and Christian Wulff-Nilsen. Dynamic minimum spanning forest with subpolynomial worst-case update time. In *Proceedings 58th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 950–961, 2017. 3

[NY19] Jelani Nelson and Huacheng Yu. Optimal lower bounds for distributed and streaming spanning forest computation. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1844–1860, 2019. 51

[PP22] Merav Parter and Asaf Petruschka. $\tilde{\text{O}}$ptimal dual vertex failure connectivity labels. In *Proceedings of the 36th International Symposium on Distributed Computing (DISC)*, volume 246 of *LIPIcs*, pages 32:1–32:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. , 1, 2

[PPP24] Merav Parter, Asaf Petruschka, and Seth Pettie. Connectivity labeling and routing with multiple vertex failures. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing (STOC)*, pages 823–834, 2024. , 1, 2, 3, 14, 17, 19, 20, 24, 25, 49, 50, 52

[PSS$^+$22] Michal Pilipczuk, Nicole Schirrmacher, Sebastian Siebertz, Szymon Torunczyk, and Alexandre Vigny. Algorithms and data structures for first-order logic with connectivity under vertex failures. In *Proceedings of the 49th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 229 of *LIPIcs*, pages 102:1–102:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. 3

[PSY22] Seth Pettie, Thatchaphol Saranurak, and Longhui Yin. Optimal vertex connectivity oracles. In *Proceedings of the 54th Annual ACM Symposium on Theory of Computing (STOC)*, pages 151–161, 2022. 1

[PT06] Mihai Pătraşcu and Mikkel Thorup. Time-space trade-offs for predecessor search. In *Proceedings of the 38th ACM Symposium on Theory of Computing (STOC)*, pages 232–240, 2006. 9, 46

[PT07] Mihai Pătraşcu and Mikkel Thorup. Planning for fast connectivity updates. In *Proceedings of the 48th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 263–271, 2007. 3, 4, 5, 6, 52

[PT14] Mihai Pătraşcu and Mikkel Thorup. Dynamic integer sets with optimal rank, select, and predecessor search. In *Proceedings 55th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 166–175, 2014. 9, 46

[Räc08] Harald Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In *Proceedings 40th Annual ACM Symposium on Theory of Computing (STOC)*, pages 255–264, 2008. 52

[RST14] Harald Räcke, Chintan Shah, and Hanjo Täubig. Computing cut-based hierarchical decompositions in almost linear time. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 227–238. SIAM, 2014. 5

[Tho18] Mikkel Thorup. $\texttt{Sample}(\texttt{x}) = (\texttt{a} * \texttt{x} <= \texttt{t})$ is a distinguisher with probability 1/8. *SIAM Journal on Computing*, 47(6):2510–2526, 2018. 54

[TZ05] Mikkel Thorup and Uri Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24, 2005. 1

[Vad12] Salil P. Vadhan. Pseudorandomness. *Foundations and Trends in Theoretical Computer Science*, 7(1–3):1–336, 2012. 55

[vdBS19] Jan van den Brand and Thatchaphol Saranurak. Sensitive distance and reachability oracles for large batch updates. In *Proceedings of the 60th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 424–435, 2019. 3

[Win89] Sein Win. On a connection between the existence of k-trees and the toughness of a graph. *Graphs and Combinatorics*, 5(1):201–205, 1989. 53

[Yu21] Huacheng Yu. Tight distributed sketching lower bound for connectivity. In *Proceedings of the 32nd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1856–1873, 2021. 51