Shortcut Partitions in Minor-Free Graphs: Steiner Point Removal, Distance Oracles, Tree Covers, and More

Hsien-Chih Chang[*] Jonathan Conroy[†] Hung Le[‡] Lazar Milenković[§]
Shay Solomon[¶] Cuong Than[‖]

Abstract

The notion of *shortcut partition*, introduced recently by Chang, Conroy, Le, Milenković, Solomon, and Than [CCL+23], is a new type of graph partition into low-diameter clusters. Roughly speaking, the shortcut partition guarantees that for every two vertices u and v in the graph, there exists a path between u and v that intersects only a few clusters. They proved that any planar graph admits a shortcut partition and gave several applications, including a construction of tree cover for arbitrary planar graphs with stretch $1+\varepsilon$ and $O(1)$ many trees for any fixed $\varepsilon \in (0,1)$. However, the construction heavily exploits planarity in multiple steps, and is thus inherently limited to planar graphs.

In this work, we breach the "planarity barrier" to construct a shortcut partition for K_r-minor-free graphs for any r. To this end, we take a completely different approach — our key contribution is a novel deterministic variant of the *cop decomposition* in minor-free graphs [And86, AGG+14]. Our shortcut partition for K_r-minor-free graphs yields several direct applications. Most notably, we construct the first *optimal* distance oracle for K_r-minor-free graphs, with $1+\varepsilon$ stretch, linear space, and constant query time for any fixed $\varepsilon \in (0,1)$. The previous best distance oracle [AG06] uses $O(n \log n)$ space and $O(\log n)$ query time, and its construction relies on Robertson-Seymour structural theorem and other sophisticated tools. We also obtain the first tree cover of $O(1)$ size for minor-free graphs with stretch $1+\varepsilon$, while the previous best $(1+\varepsilon)$-tree cover has size $O(\log^2 n)$ [BFN19].

As a highlight of our work, we employ our shortcut partition to resolve a major open problem — the *Steiner point removal (SPR)* problem: Given any set K of *terminals* in an arbitrary edge-weighted planar graph G, is it possible to construct a minor M of G whose vertex set is K, which preserves the shortest-path distances between all pairs of terminals in G up to a *constant* factor? Positive answers to the SPR problem were only known for very restricted classes of planar graphs: trees [Gup01], outerplanar graphs [BG08], and series-parallel graphs [HL22]. We resolve the SPR problem in the affirmative for any planar graph, and more generally for any K_r-minor-free graph for any fixed r. To achieve this result, we prove the following general reduction and combine it with our new shortcut partition: For any graph family closed under taking subgraphs, the existence of a shortcut partition yields a positive solution to the SPR problem.

[*]Department of Computer Science, Dartmouth College. Email: hsien-chih.chang@dartmouth.edu.
[†]Department of Computer Science, Dartmouth College. Email: jonathan.conroy.gr@dartmouth.edu
[‡]Manning CICS, UMass Amherst. Email: hungle@cs.umass.edu
[§]Tel Aviv University. Email: lazarm@mail.tau.ac.il
[¶]Tel Aviv University. Email: shayso@tauex.tau.ac.il
[‖]Manning CICS, UMass Amherst. Email: cthan@cs.umass.edu

1 Introduction

Partitioning a graph into clusters is a fundamental primitive for designing algorithms. Perhaps the most basic requirement of a partition is that every cluster would have a small diameter. However, to be useful, most partitions require one or more additional constraints, and achieving these constraints is the key to the power of those partitions. For example, *probabilistic partition* [Bar96], a principal tool in the metric embedding literature, guarantees that the probability of any two vertices being placed into different clusters is proportional to their distance. *Sparse partition* [AP90, JLN+05] guarantees that each cluster has neighbors in only a few other clusters. *Scattering partition* [Fil20b] guarantees that each shortest path up to a certain length only intersects a small number of clusters. These partitions have found a plethora of applications in a wide variety of areas, such as metric embeddings, distributed computing, routing, and algorithms for network design problems, to name a few.

Recently, Chang, Conroy, Le, Milenković, Solomon, and Than [CCL+23] introduced a new notion of partition called *shortcut partition*. Roughly speaking, a shortcut partition guarantees that for every two vertices u and v in the graph, there exists a low-hop path in the cluster graph between C_u and C_v, where C_u and C_v are the clusters containing u and v, respectively. More formally, a *clustering* of a graph G is a partition of the vertices of G into connected *clusters*. The *cluster graph* $\breve{G}$ of a clustering $\mathcal{C}$ of G is the graph where each vertex of $\breve{G}$ corresponds to a cluster in $\mathcal{C}$, and there is an edge between two vertices in $\breve{G}$ if there is an edge in G whose endpoints are in the two corresponding clusters. We always treat $\breve{G}$ as an unweighted graph; to emphasize this, we use the terms *hop-length* and *hop-distance* to refer to path lengths and distances in $\breve{G}$.

Definition 1.1. *An (ε, h)-shortcut partition is a clustering $\mathcal{C} = \{C_1, \ldots, C_m\}$ of G such that:*

- [Diameter.] *the strong*[1] *diameter of each cluster C_i is at most $\varepsilon \cdot \mathrm{diam}(G)$;*
- [Low-hop.] *for any vertices u and v in G, there is a path $\breve{\pi}$ in the cluster graph $\breve{G}$ between the clusters containing u and v such that:*
  - *(1) $\breve{\pi}$ has hop-length at most $h \cdot \max\left\{\frac{\delta_G(u,v)}{\mathrm{diam}(G)}, \varepsilon\right\}$,*
  - *(2) there exists a shortest path π in G between u and v, such that $\breve{\pi}$ only contains (a subset of) clusters that have nontrivial intersections with π.*

Notice that the hop-length of $\breve{\pi}$ is always at most h, as $\delta_G(u,v)$ is at most $\mathrm{diam}(G)$. In the other extreme, if $\delta_G(u,v) \le \varepsilon \cdot \mathrm{diam}(G)$, then we guarantee that the hop-length is at most εh.

The shortcut partition is similar to the scattering partition introduced by Filtser [Fil20b]. A key difference is that in a scattering partition, *every* shortest path of length $\alpha\varepsilon \cdot \mathrm{diam}(G)$ intersects at most $O(\alpha)$ clusters, while in a shortcut partition, it only requires that there is a low-hop path in the cluster graph between the two clusters containing the path's endpoints. The fact that scattering partition requires a stronger guarantee on shortest paths makes it very difficult to construct; it remains an open problem whether scattering partition for every planar graph exists [Fil20b, Conjecture 1]. Although shortcut partition provides a weaker guarantee, it is already sufficient for many applications as shown in previous work [CCL+23], including the first tree cover in planar graphs with stretch $1+\varepsilon$ using $O(1)$ many trees for any fixed $\varepsilon \in (0,1)$, a simpler proof to the existence of a $+\varepsilon \cdot \mathrm{diam}(G)$ additive embedding of planar graph into bounded-treewidth graph, distance oracles, labeling schemes, (hop-)emulators, and more.

For any given $\varepsilon \in (0,1)$, the authors of [CCL+23] constructed an $(\varepsilon, O(\varepsilon^{-2}))$-shortcut partition for any planar graph. This naturally motivates the question of constructing a shortcut partition for broader classes

[1] The *strong* diameter of cluster C is the one of induced subgraph $G[C]$. In contrast, the *weak* diameter of C is $\max_{u,v\in C} \delta_G(u,v)$. Here, and throughout this paper, $\delta_G(u,v)$ denotes the distance between u and v in graph G.

of graphs, specifically K_r-minor-free graphs.[2] This will open the door to seamlessly extend algorithmic results from planar graphs to K_r-minor-free graphs. However, the construction of [CCL+23] heavily exploits planarity in multiple steps. It starts from the outerface of G, and works toward the interior of G in a recursive manner, similar in spirit to Busch, LaFortune, and Tirthapura [BLT14]. Specifically, the construction first finds a collection of subgraphs of G call *columns* such that every vertex near the outerface of G belongs to one of the columns. The construction then recurs on subgraphs induced by vertices that are not in any of the columns. The overall construction produces a structure called the *grid-tree hierarchy*, which is then used to construct a shortcut partition. The construction relies on the fact that each column contains a shortest path between two vertices on the outer face, which splits the graph into two subgraphs using Jordan curve theorem. As a result, constructing a shortcut partition for K_r-minor-free graphs requires breaking away from the planarity-exploiting framework of [CCL+23].

In this work, we overcome this barrier and construct a shortcut partition for K_r-minor-free graphs.

Theorem 1.2. *Any edge-weighted K_r-minor-free graph admits an $(\varepsilon, 2^{O(r \log r)}/\varepsilon)$-shortcut partition for any $\varepsilon \in (0,1)$.*

Remark. *Definition 1.1 is slightly stronger than the corresponding definition of shortcut partition for planar graphs in [CCL+23] (Definition 2.1 in their paper). Specifically, their definition states that the hop-length of $\check{\pi}$ is at most h, regardless of $\delta_G(u,v)$, while our definition allows smaller hop-lengths for smaller distances. (For example, when $\delta_G(u,v) = \varepsilon \cdot \mathrm{diam}(G)$, the hop-length of $\check{\pi}$ is $O(\varepsilon h)$ instead of h.) Another difference is that, in the current definition, the nontrivial intersections of clusters contained by $\check{\pi}$ stated in condition (2) of the "low-hop" property are with respect to a shortest path in the graph, whereas in [CCL+23] they are with respect to an approximate shortest path; we discuss this point further in Section 5. In particular, the shortcut partition provided by Theorem 1.2 for minor-free graphs subsumes the one in [CCL+23] for planar graphs.*

The hop length of $2^{O(r \log r)}/\varepsilon$ of the shortcut partition in Theorem 1.2 is optimal for every constant r up to a constant factor: any shortcut partition of a path would have hop length $1/\varepsilon$ between the two endpoints of the path. Also, in the particular case of planar graphs, our shortcut partition in fact improves over [CCL+23]; the hop length of their partition is $O(\varepsilon^{-2})$.

Techniques. We base our construction on a modified *cop decomposition* for K_r-minor-free graphs, first introduced by Andreae [And86] in the context of the *cops-and-robbers* game. A cop decomposition is a rooted tree decomposition; loosely speaking, while a standard tree decomposition guarantees that each bag contains a bounded number of vertices, a cop decomposition instead guarantees that each bag contains vertices from at most $r-2$ single-source shortest path (SSSP) trees. These SSSP trees are called *skeletons*. Abraham, Gavoille, Gupta, Neiman, and Talwar [AGG+14], in their construction of a padded decomposition for K_r-minor-free graphs, adapted the cop decomposition by allowing each bag to contain up to $r-2$ clusters[3], each of which is the set of vertices within radius $\sigma \cdot \Delta$ from a skeleton in the bag; here σ is a parameter in $(0,1)$ randomly sampled from a truncated exponential distribution, and $\Delta := \varepsilon \cdot \mathrm{diam}(G)$.

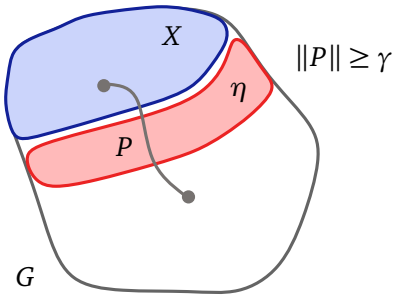

Figure 1. A cluster X "cut off" by η from part of graph G. There is a buffer of width γ between X and the part of the graph that it is cut off from.

[2] We sometimes drop the prefix "K_r-" in K_r-minor-free graphs when the clique minor has constant size r.

[3] Later in the paper, we rename the clusters as *supernodes*; we reserve the former term for the actual clusters in the shortcut partition to be constructed.

One of their goals is to ensure that for every vertex v in the graph, the process of constructing the cop decomposition guarantees that the number of (random) clusters that could contain v is small *in expectation*; this is the bulk of their analysis, through clever and sophisticated use of potential function and setting up a (sub)martingale. Their result can be interpreted as guaranteeing what we call a *buffer property*[4]:

> If one cluster X is "cut off" from a piece of the graph by another cluster, then any path from X to that piece has length at least γ, which we call the *buffer width*.

In particular, the construction of [AGG+14] intuitively implies that the *expected* buffer width is about γ. (Their end goal is a stochastic partition, and hence they could afford the buffer property in expectation.) Their expected bound on the buffer width is insufficient for our shortcut partition, as well as for all other applications considered in our paper. One either has to guarantee that the buffer width holds with high probability or, ideally, holds deterministically.

In this work, we achieve the buffer property *deterministically*. To do this, we add a layer of recursion on top of the cop decomposition by [AGG+14] to directly fix the buffer property whenever it is violated, thereby bypassing the need for the complicated analysis of the potential function. In more detail, we build a cop decomposition by iteratively creating clusters. At each point in the construction, we create an SSSP tree connecting to some existing clusters, and initialize a new cluster with that tree as the cluster's skeleton. Our idea to enforce the buffer property is natural: we (recursively) assign those vertices that violate the property to join previously-created clusters. Specifically, whenever a cluster X is cut off by some cluster η, we assign every vertex within distance γ of X to be a part of some existing clusters. However, enforcing the buffer property directly comes at the cost of increasing the radius of some existing clusters — recall that we want all points in a cluster to be at most $O(\Delta)$ distance away from its skeleton. Therefore, our implementation of vertex assignment is very delicate; otherwise, the diameter of a cluster could continue growing, passing the diameter bound prescribed by the shortcut partition. Our key insight is to show that during the course of our construction, each cluster can only be expanded a single time for each of the $O(r)$ clusters that it can "see". This lets us achieve a deterministic buffer width of $\gamma = O(\Delta/r)$.

1.1 Steiner Point Removal Problem

In the Steiner Point Removal (SPR) problem, we are given an undirected weighted graph $G = (V, E, w)$ with vertex set V, edge set E, nonnegative weight function w over the edges, and a subset K of V. The vertices in K are called *terminals* and the vertices in $V \setminus K$ are called *non-terminal* or *Steiner* vertices. The goal in the SPR problem is to find a graph *minor* M of G such that $V(M) = K$, and for every pair t_1, t_2 of terminals in K, $\delta_M(t_1, t_2) \leq \alpha \cdot \delta_G(t_1, t_2)$, for some constant $\alpha \geq 1$; such a graph minor M of G is called a *distance-preserving minor* of G with *distortion* α.[5]

Gupta [Gup01] was the first to consider the problem of removing Steiner points to preserve all terminal distances. He showed that every (weighted) *tree* can be replaced by another tree on the terminals where the shortest path distances are preserved up to a factor of 8; another proof of this result is given by [FKT18]. Chan, Xia, Konjevod, and Richa [CXKR06] observed that Gupta's construction in fact produces a distance-preserving minor of the input tree, and showed a matching lower bound: there exists a tree and a set of terminals, such that any distance-preserving minor of that tree must have distortion at least $8(1-o(1))$. Both Chan *et al.* [CXKR06] and Basu and Gupta [BG08] considered the following question:

[4]There is a technical difference between our buffer property and that of [AGGM06], which we clarify in Section 3 (see Remark 3.3).

[5]In the literature [KNZ14] the term *distance-preserving minor* allows the existence of Steiner vertices as well, with the goal to minimize their usage. For our purpose we do not allow any Steiner vertices.

Question 1.3. *Does every K_r-minor-free graph for any fixed r admit a distance-preserving minor with constant distortion?*

Question 1.3 has attracted significant research attention over the years, and numerous works have attempted to attack it from different angles. Some introduced new frameworks [FKT18, Fil19, Fil20a] that simplify known results; others considered the problem for general graphs, establishing the distortion bound of $O(\log|K|)$ after a sequence of works [KKN15, Che18, Fil19]; there are also variants where Steiner points are allowed, but their number should be minimized [KNZ14, CGH16, CKT22]; and yet another achieved a constant *expected* distortion [EGK+14].

Nevertheless, Question 1.3 remains wide open: a positive solution for K_r-minor-free graphs is not known for any $r \geq 5$. Gupta's result for trees [Gup01] can be seen as providing a solution for K_3-minor-free graphs. Basu and Gupta [BG08] gave a positive answer for outerplanar graphs (which is $(K_{2,3}, K_4)$-minor-free). Recently, Hershkowitz and Li [HL22] provided a solution for K_4-minor-free graphs, also known as *series-parallel graphs*. Even for planar graphs, a subclass of K_5-minor-free graphs, the answer is not known. Both outerplanar and series-parallel graphs are very restricted classes of planar graphs: they have treewidth at most 2. For slightly larger graph classes, such as treewidth-3 planar graphs or k-outerplanar graphs for any constant k, the SPR problem has remained open to date.

We resolve Question 1.3 in the affirmative, thus solving the SPR problem for minor-free graphs in its full generality:

Theorem 1.4. *Let $G = (V, E, w)$ be an arbitrary edge-weighted K_r-minor-free graph and let $K \subseteq V$ be an arbitrary set of terminals. Then, there is a solution to the SPR problem on G with distortion $2^{O(r \log r)}$.*

We prove Theorem 1.4 by devising a general reduction from SPR to shortcut partition. Specifically:

Theorem 1.5. *If every subgraph of G admits an $(\varepsilon, f(r)/\varepsilon)$-shortcut partition for every $\varepsilon \in (0,1)$, then G admits a solution to the SPR problem with distortion $O(f(r)^{13})$.*

The proof of Theorem 1.5 builds on a reduction by Filtser [Fil20b], from the SPR problem to that of finding *scattering partitions*, which require every shortest path between two vertices to intersect only a small number of clusters. We introduce an inherently relaxed notion which we call the *approximate scattering partition* (Definition 2.1) — which among other changes uses *approximate* shortest paths rather than exact shortest paths — and adapt Filtser's reduction using the new notion. The first challenge underlying this adaptation is that, unlike shortest paths, an approximate shortest path does not have the optimal substructure property (any subpath of a shortest path is also a shortest path). The second and perhaps more significant challenge stems from the fact that the partition only guarantees the existence of *some* low-hop path in the cluster graph, and the distortion to its length is *not* with respect to the distance between the two endpoints. We explain the differences in detail in Section 2. Consequently, we have to make some crucial changes in the reduction, and more so in its analysis.

We observe that Theorem 1.5 together with a shortcut partition in Theorem 1.2 gives us a solution to the SPR problem with $O(1)$ distortion in K_r-minor-free graphs, since in this case, $f(r) = r^{O(r)}$ and r is fixed.

1.2 Other Applications of Our Results

Distance oracle. An *α-approximate distance oracle* is a compact data structure for graph G that given any two vertices u and v, return the distance between u and v in G up to a factor of α. In constructing a distance oracle, we would like to minimize the *distortion* parameter α, the *space* usage, and the *time* it takes to answer a query; there is often a tradeoff between the three parameters.

Constructing $(1+\varepsilon)$-approximate distance oracles for planar graphs has been extensively studied. A long line of work [Tho04, Kle02, KKS11, WN16, GX19, CS19] recently culminated in an optimal distance oracle with linear space and constant query time by Le and Wulff-Nilsen [LWN22]. On the other hand, the only known $(1+\varepsilon)$-approximate distance oracle for K_r-minor-free graphs achieving $O(n \log n)$ space and $O(\log n)$ query time (for any constant $\varepsilon \in (0,1)$ and constant r) was by Abraham and Gavoille [AG06]. The main reason is that the topology of K_r-minor-free graphs is much more complicated, and many techniques from planar graphs — such as reduction to additive distance oracles [KKS11, LWN22] or more sophisticated use of planar shortest path separators [WN16] — do not extend to K_r-minor-free graphs. Even the shortest path separator [AG06] in K_r-minor-free graphs does not behave as well as its planar counterpart [GKR01, Tho04]: each path in the separator in K_r-minor-free graphs is not a shortest path of the *input graph*, but a shortest path of its *subgraph* after some previous paths were removed. As a result, despite significant recent progress on approximate distance oracles for planar graphs, the following problem remains open:

Problem 1.6. *Design a $(1+\varepsilon)$-approximate distance oracle for K_r-minor-free graphs with linear space and constant query time for fixed ε and r.*

In this work, we resolve Theorem 1.7 affirmatively. Our oracle can also be implemented in the pointer-machine model, matching the best-known results for planar graphs [CCL+23].

Theorem 1.7. *Given any parameter $\varepsilon \in (0,1)$, and any edge-weighted undirected K_r-minor-free graphs with n vertices, we can design a $(1+\varepsilon)$-approximate distance oracle with the following guarantees:*

- *Our distance oracle has space $n \cdot 2^{r^{O(r)}/\varepsilon}$ and query time $2^{r^{O(r)}/\varepsilon}$ in the word RAM model with word size $\Omega(\log n)$. Consequently, for fixed ε and r, the space is $O(n)$ and query time is $O(1)$.*
- *Our distance oracle has space $O(n \cdot 2^{r^{O(r)}/\varepsilon})$ and query time $O(\log\log n \cdot 2^{r^{O(r)}/\varepsilon})$ in the pointer machine model.*

Our oracle is constructed via tree covers, which we will discuss next.

Tree cover. An *α-tree cover* $\mathcal{T}$ of a metric space (X, δ_X) for some $\alpha \geq 1$ is a collection of trees such that: (1) every tree $T \in \mathcal{T}$ has $X \subseteq V(T)$ and $d_T(x,y) \geq \delta_X(x,y)$ for every two points $x, y \in X$, and (2) for every two points $x, y \in X$, there exists a tree $T \in \mathcal{T}$ such that $d_T(x,y) \leq \alpha \cdot \delta_X(x,y)$. We call α the *distortion* of the tree cover $\mathcal{T}$. The size of the tree cover is the number of trees in $\mathcal{T}$.

Tree covers have been extensively studied for many different metric spaces [AP92, AKP94, ADM+95, GKR01, BFN19, FL22, KLMS22]. Gupta, Kumar, and Rastogi [GKR01] showed among other things that planar metrics admit a tree cover of distortion 3 and size $O(\log n)$. Bartal, Fandina, and Neiman [BFN19] reduced the distortion to $1+\varepsilon$ for any fixed $\varepsilon \in (0,1)$ at the cost of a higher number of trees, $O(\log^2 n)$. Their result also holds for any K_r-minor-free graphs with a fixed r; however, because of the usage of shortest path separator [AG06], the final tree cover size contains a hidden dependency on r which is the Robertson-Seymour constant [RS03], known to be bigger than the tower function of r. Their work left several questions open: (a) Can we construct a $(1+\varepsilon)$ tree cover of $O(1)$ size for planar graphs, and more generally K_r-minor-free graphs? (b) Can we avoid Robertson-Seymour decomposition and achieve a more practical construction?

The shortcut partition introduced by Chang *et al.* [CCL+23] partially resolved the first question: they constructed a $(1+\varepsilon)$-tree cover for *planar graphs* of $O(1)$ size. Using our new shortcut partition in Theorem 1.2, we resolve the question of Bartal *et al.* for all K_r-minor-free graphs. As our construction is rooted in the cop decomposition, the construction might behave reasonably well even when the graph is

not strictly K_r-minor-free, as the performance ultimately depends on the width of the buffer and the number of times a cluster can expand. This provides a more practical alternative to the Robertson-Seymour decomposition.

Theorem 1.8. *Let G be any edge-weighted undirected K_r-minor-free graph with n vertices. For any parameter $\varepsilon \in (0,1)$, there is a $(1+\varepsilon)$-tree cover for the shortest path metric of G using $2^{r^{O(r)}/\varepsilon}$ trees.*

Given a tree cover $\mathcal{T}$ in Theorem 1.8, we can obtain a $(1+\varepsilon)$-approximate distance oracle in Theorem 1.7 as follows. The distance oracle consists of $\mathcal{T}$ and an LCA data structure for each tree in $\mathcal{T}$. For each query pair (u,v), we iterate through each tree, compute the distance on the tree using LCA data structure, and then return $\min_{T\in\mathcal{T}} d_T(u,v)$. The query time and space are as described in Theorem 1.7 because $|\mathcal{T}| = 2^{r^{O(r)}/\varepsilon}$; the distortion is $1+\varepsilon$ since the distortion of the tree cover is $1+\varepsilon$.

Additive embeddings for apex-minor-free graphs. Graph A is an *apex graph* if there exists a vertex $a \in V(A)$, called the *apex*, such that $A \setminus \{a\}$ is a planar graph. A graph G is *apex-minor-free* if it excludes some apex graph A of $O(1)$ size as a minor. We note that apex-minor-free graphs include planar graphs and, more generally, *bounded-genus graphs* as subclasses. We show that our shortcut partition also gives the first deterministic additive embeddings of apex-minor-free graphs into bounded-treewidth graphs.

Given a weighted graph G of diameter Δ, we say that a (deterministic) embedding $f : V(G) \to H$ of G into H has *additive distortion* $+\varepsilon\Delta$ if $d_G(x,y) \le d_H(f(x),f(y)) \le d_G(x,y)+\varepsilon\Delta$ for every $x,y \in V(G)$. The goal is to construct an embedding f such that the treewidth of H, denoted by $\mathrm{tw}(H)$, is minimized. Ideally, we would like $\mathrm{tw}(H)$ to depend only on ε and not on the number of vertices of G.

Additive embeddings have been studied recently for planar graphs [FKS19, FL22, CCL+23] and for minor-free graphs [CFKL20]. A key result in this line of work is an additive embedding for planar graphs where the treewidth of H is polynomially dependent on ε [FKS19]; specifically, they achieved $\mathrm{tw}(H) = O(1/\varepsilon^c)$ for some constant $c \ge 58$, which was recently improved to $\mathrm{tw}(H) = O(1/\varepsilon^4)$ [CCL+23]. Cohen-Addad *et al.* [CFKL20] constructed a family of apex graphs and showed that any deterministic embedding with additive distortion $+\Delta/12$ ($\varepsilon = 1/12$) for the family must have treewidth $\Omega(\sqrt{n})$. Their result left an important question regarding additive embeddings of apex-minor-free graphs. Here we use the shortcut partition in Theorem 1.2 to resolve this problem, and thereby completing our understanding of deterministic additive embeddings of graphs excluding a fixed minor into bounded-treewidth graphs.

Theorem 1.9. *Let G be any given edge-weighted graph of n vertices excluding a fixed apex graph as a minor. Let Δ be the diameter of G. For any given parameter $\varepsilon \in (0,1)$, we can construct in polynomial time a deterministic embedding of G into a graph H such that the additive distortion is $+\varepsilon\Delta$ and $\mathrm{tw}(H) = 2^{O(\varepsilon^{-1})}$.*

In addition to the aforementioned results, we also obtain generalizations to minor-free graphs of results from [CCL+23]; we simply use our tree cover from Theorem 1.8 in place of their tree cover theorem for planar graphs. The results include (1) the first $(1+\varepsilon)$-emulator of linear size for minor-free graphs, (2) low-hop emulators for minor-free metrics, (3) a compact distance labeling scheme for minor-free graphs, and (4) routing in minor-free metrics. We refer readers to [CCL+23] for more details.

Organization. In Section 2 we resolve the SPR problem by constructing approximate scattering partition using the shortcut partition in Theorem 1.2. In Section 3, we introduce and describe in full detail the construction of the buffered cop decomposition, which we will use in Section 4 to construct the shortcut partition. In Section 5, we give the details of the applications of shortcut partition in constructing tree cover, distance oracle, and additive embedding into bounded treewidth graphs.

2 Reduction to Shortcut Partition

As mentioned, Filtser [Fil20b] presented a reduction from the SPR problem to that of finding *scattering partitions*. To prove Theorem 1.5, we introduce an inherently relaxed notion of *approximate* scattering partition (refer to Definition 2.1), and adapt the reduction of [Fil20b, Theorem 1] using that notion. Due to the usage of our relaxed notion of partition, we have to make some crucial changes in the reduction, and alter various parts of the analysis.

Definition 2.1 (Approximate Scattering Partition). *Let $G = (V, E, w)$ be an edge-weighted graph. A β-approximate (τ, Δ)-scattering partition of G is a partition $\mathcal{C}$ of V such that:*

- [Diameter.] *For each cluster C in $\mathcal{C}$, the induced subgraph $G[C]$ has weak diameter at most Δ; that is, $\delta_G(u, v) \leq \Delta$ for any vertices u and v in C.*
- [Scattering.] *For any two vertices u and v in V such that $\delta_G(u, v) \leq \Delta$, there exists a path π in G between u and v where (1) π has length at most $\beta \cdot \Delta$, (2) every edge in π has length at most Δ, and (3) π intersects at most τ clusters in $\mathcal{C}$. We say π is a β-approximate (τ, Δ)-scattered path.*

We remark that scattering properties (2) and (3) together imply property (1): the length of π is at most $O(\tau) \cdot \Delta$. Nevertheless, we prefer to keep property (1) separately from properties (2) and (3) in the definition to emphasize the fact that π is an approximate path.

Notice that the notion of approximate scattering partition is more relaxed than the original notion of scattering partition [Fil20b]. A scattering partition requires *every* shortest path with length at most Δ to be τ-scattered. However in an approximate scattering partition there are three relaxations:

1. we only require that *one* such path exists;
2. that path may be an approximate shortest path (rather than an exact shortest path);
3. the β-approximation to the length of such path π is *not* with respect to the distance between the endpoints; rather, the length of π is bounded by β times Δ, the diameter bound of clusters.

The following lemma, which we prove in §2.1 and §2.2, is analogous to Theorem 1 by Filtser [Fil20b], except for the key difference that we employ approximate scattering partitions. We show that such partitions, with the three aforementioned relaxations introduced, still suffice for solving the SPR problem.

Lemma 2.2. *Let G be a graph such that for every $\Delta > 0$, every induced subgraph of G admits a β-approximate (τ, Δ)-scattering partition, for some constants $\beta, \tau \geq 1$. Then, there is a solution to the SPR problem on G with distortion $O(\tau^8 \cdot \beta^5) = O(1)$.*

To construct approximate scattering partitions, we use *shortcut partitions*. Recall the *cluster graph* of G with respect to $\mathcal{C}$, denoted $\check{G}$, is the graph obtained by contracting each cluster in $\mathcal{C}$ into a *supernode*. The *hop-length* of a path is the number of edges in the path.

Lemma 2.3. *Let G be a graph and let $\Delta > 0$ be a parameter. If any subgraph of G has an (ε, h)-shortcut partition for any $\varepsilon \in (0, 1)$ and some number h, then G has a $2\varepsilon h$-approximate $(\varepsilon h, \Delta)$-scattering partition.*

Proof: Construct graph G' from G by removing all edges of length greater than Δ. Notice that if any pair u, v of vertices satisfies $\delta_G(u, v) \leq \Delta$, then it also satisfies $\delta_{G'}(u, v) \leq \Delta$. Thus, any partition of vertices that satisfies the approximate scattering property for G' also satisfies that property for G.

Let $\mathcal{C}$ be an (ε, h)-shortcut partition for the graph G', with parameter $\varepsilon := \Delta / \operatorname{diam}(G')$. Notice that $\mathcal{C}$ is a clustering of the vertices of G, where for any cluster C in $\mathcal{C}$, the induced subgraphs $G[C]$ and $G'[C]$

have strong diameter at most $\varepsilon \cdot \mathrm{diam}(G') = \Delta$; thus, $\mathcal{C}$ satisfies the diameter property of approximate scattering partition.

We now show that $\mathcal{C}$ satisfies the scattering property. Let u and v be two vertices in G with $\delta_G(u,v) \leq \Delta$. Note that $\delta_{G'}(u,v) \leq \Delta$. By the properties of shortcut partition, there is a path $\check{\pi}$ in the cluster graph $\check{G}$ between the clusters containing u and v, such that $\check{\pi}$ has hop-length at most $h \cdot \max\left\{\frac{\delta_{G'}(u,v)}{\mathrm{diam}(G')}, \varepsilon\right\} = \varepsilon h$. In other words, the hop-length of $\check{\pi}$ is t, for some t that is upper-bounded by εh. Write $\check{\pi} = (C_1, C_2, \ldots, C_t)$ as a sequence of t adjacent clusters in $\check{G}$. Notice that two clusters C and C' in $\check{G}$ are *adjacent* if and only if there is an edge in G' between a vertex in C and a vertex in C'. For every pair of consecutive clusters C_i and C_{i+1} in $\check{\pi}$, let x'_i be a vertex in C_i and x_{i+1} be a vertex in C_{i+1} such that there is an edge e_i in G' between x'_i and x_{i+1}. To simplify notation, define $x_1 := u$ and define $x'_t := v$. With this definition, x_i and x'_i are defined for all i in $\{1, \ldots, t\}$. Notice that for every i in $\{1, \ldots, t\}$, vertices x_i and x'_i are both in cluster C_i. By the strong diameter property of $\mathcal{C}$, there is a path P_i in G' between x_i and x'_i, such that P_i is contained in C_i and has length at most Δ.

We define the path π in G' (and thus also in G) between u and v to be the concatenation $P_1 \circ e_1 \circ P_2 \circ e_2 \circ \ldots \circ P_t$. Notice that (1) π has length at most $2t \cdot \Delta \leq 2\varepsilon h \cdot \Delta$; indeed, each subpath P_i has length at most Δ (by the strong diameter property), and each edge e_i has length at most Δ (as e_i is in G'). Further, (2) every edge of π has length at most Δ, and (3) π intersects at most εh clusters (namely, the clusters $C_1, \ldots, C_t$ along $\check{\pi}$). □

Theorem 1.5 follows from Lemma 2.2 and Lemma 2.3. As a direct corollary of Theorem 1.2 and Lemma 2.3, we obtain the following.

Corollary 2.4. *There are constants β and τ such that, for any K_r-minor-free graph G and any $\Delta > 0$, there exists a β-approximate (τ, Δ)-scattering partition of G. Specifically, $\beta = \tau = 2^{O(r \log r)}$.*

In what follows we prove Lemma 2.2.

2.1 Algorithm

Our construction for proving Lemma 2.2 is similar to that of [Fil20b], but deviates from it in several crucial points (see Remark 2.5 for details). For completeness, we next provide the entire construction of [Fil20b], adapted appropriately to our purposes.

We will assume without loss of generality that the minimum pairwise distance is 1. We shall partition V into $|K|$ connected subgraphs, each of which corresponds to a single terminal in K. Each vertex in V will be *assigned* to a connected subgraph by the *assignment function* $f : V \to K$, such that at the end of the process, we can create a graph minor M of G by contracting each connected subgraph $f^{-1}(t)$ into a supernode for every terminal $t \in K$. By setting $w_M(t,t') := \delta_G(t,t')$ for each edge $(t,t') \in E(M)$, the edge-weighted graph $M = (K, E(M), w_M)$ is our solution to the SPR problem on G. For a path P, we denote by $||P||$ the length of P.

We compute the assignment function f in iterations. In iteration i we shall compute a function $f_i : V \to K \cup \{\perp\}$, where $\perp$ symbolizes that the vertex remains unassigned. The function f will be obtained as the function f_i computed at the last iteration of the algorithm. We will maintain the set of *relevant vertices* $\mathcal{R}_i := \left\{v \in V \mid \zeta^{i-1} \leq \delta_G(v,K) < \zeta^i\right\}$ and the set of *assigned vertices* V_i by the function f_i to some terminals, for each iteration i, where $\zeta := c \cdot \beta \cdot \tau$, for β and τ being the constants provided by Corollary 2.4 and c being some large constant. Initialize $f_0(t) := t$ for each $t \in K$, and $f_0(v) := \perp$ for each $v \in V \setminus K$. Define both $\mathcal{R}_0$ and V_0 to be K. Inductively, we maintain the properties that $V_{i-1} \subseteq V_i$ and $\bigcup_{j \leq i} \mathcal{R}_j \subseteq V_i$, hence the algorithm terminates when all vertices have been assigned.

At the i-th iteration of the algorithm, we compute β-approximate (τ, ζ^{i-1})-scattering partition $\mathcal{P}_i$, provided by Corollary 2.4, on the subgraph induced on the unassigned vertices $G_i := G[V \setminus V_{i-1}]$. Let $\mathcal{C}_i$

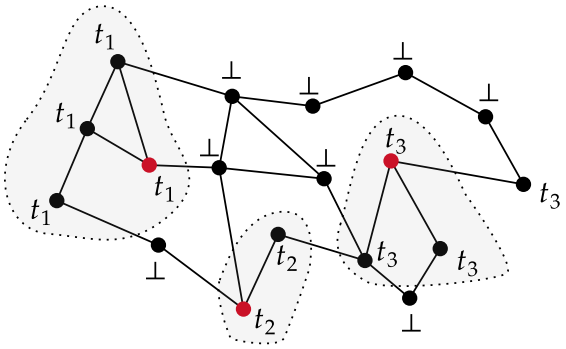

Figure 2. An SPR instance with 3 Steiner points. Values of assignment function f are shown next to vertices.

be the set of clusters in $\mathcal{P}_i$ that contain at least one vertex in $\mathcal{R}_i$. All vertices in the clusters of $\mathcal{C}_i$ will be assigned by f_i at iteration i.

We classify the clusters in $\mathcal{C}_i$ into *levels*, starting from level 0, viewing V_{i-1} as a *level-0* cluster. We say that a cluster $C \in \mathcal{C}_i$ is at *level j* if j is the minimum index such that there is an edge of weight at most ζ^i connecting a vertex u in C and another vertex v in some level-$(j-1)$ cluster C'. If there are multiple such edges, we fix one of them arbitrarily; we call vertex v in C' the *linking vertex* of C. Let $lv_i(C)$ denote the level of C. Observe that every $\mathcal{C}_i$ contains a vertex in $\mathcal{R}_i$, i.e., there exists a vertex $v \in \mathcal{C}_i$ such that $\zeta^{i-1} \leq \delta_G(v,K) < \zeta^i$. Hence, it is readily verified that every cluster $\mathcal{C}_i$ has a linking vertex, and thus $lv_i(C)$ is a valid level.

For every vertex $v \in V_{i-1}$, we set $f_i(v) \coloneqq f_{i-1}(v)$. For every vertex not in $\bigcup \mathcal{C}_i$ (or V_{i-1}), we set $f_i(v) = \perp$. Next, we scan all clusters in $\mathcal{C}_i$ by non-decreasing order of level, starting from level 1. For each vertex u in each cluster C, we set $f_i(u)$ to be $f_i(v_C)$, where v_C is the linking vertex of C. If some unassigned vertices remain, we proceed to the next iteration; otherwise, the algorithm terminates.

Remark 2.5. *The algorithm presented here is different than that of [Fil20b] in three important points:*

- *As mentioned, we use approximate scattering partitions (as in Definition 2.1) rather than the scattering partitions of [Fil20b]. This change poses several technical challenges in the argument.*
- *In the proof of Lemma 2.2, one has to consider β-approximate path I' and its replacement I'' at the ith and $(i+1)$th iterations, respectively. [Fil20b] uses the same path in both iterations.*
- *Moreover, β-approximate path I' at the ith iteration has its length increased by a constant β; one has to guarantee that every subpath P_i is still short enough to be subject to the approximate scattering partition at the $(i+1)$th iteration (see Eq. 8 and the paragraph that follows). Thus we do not use constant 2 as in [Fil20b] but rather use a bigger constant ζ (as defined above).*

2.2 Distortion Analysis

From the algorithm, any vertex within distance between ζ^{i-1} to ζ^i from K is assigned at iteration at most i. However, the following claim narrows the possibilities down to two choices. The claim is analogous to Claim 5 in [Fil20b], where we use ζ instead of 2, and its proof is similar.

Claim 2.6 ([Fil20b, Claim 5]). *Any vertex v satisfying $\zeta^{i-1} \leq \delta_G(v,K) < \zeta^i$ is assigned during iteration $i-1$ or i. Consequently, any vertex v assigned during iteration i must satisfy $\zeta^{i-1} \leq \delta_G(v,K) < \zeta^{i+1}$.*

Proof: If v remains unassigned until iteration i, it will be assigned during iteration i by construction. Suppose that v was assigned during iteration j. Then v belongs to a cluster $C \in \mathcal{C}_j$, and there is a vertex $u \in C$ with $\delta_G(u,K) \leq \zeta^j$. As C has strong diameter at most ζ^{j-1} and $\zeta > 2$, we obtain

$$\zeta^{i-1} \leq \delta_G(v,K) \leq \delta_G(u,K) + \delta_G(u,v) \leq \zeta^j + \zeta^{j-1} < \zeta^2 \cdot \zeta^{j-1},$$

implying that $i-1 < 2+(j-1)$, or equivalently $j \geq i-1$. □

The following claim is analogous to Corollary 1 in [Fil20b], but we introduce a few changes in the proof.

Claim 2.7 ([Fil20b, Corollary 1]). *For every $v \in V$, $\delta_G(v, f(v)) \leq 3\tau \cdot \zeta^2 \cdot \delta_G(v,K)$.*

Proof: Let i be the iteration in which v is assigned, and let C_v be the cluster in $\mathcal{C}_i$ containing v. We shall prove that

$$\delta_G(v, f(v)) \leq 3\tau \cdot \zeta^{i+1}. \tag{1}$$

Combining this bound with Claim 2.6 yields

$$\delta(v, f(v)) \leq 3\tau \cdot \zeta^{i+1} \leq 3\tau \cdot \zeta^2 \cdot \delta_G(v,K),$$

as required. The proof is by induction on the iteration i in which v is assigned. The base case $i = 0$ is trivial, as then v is a terminal, and we have $\delta_G(v, f(v)) = 0 \leq 3\tau \cdot \zeta^{0+1}$. We henceforth consider the induction step when $i \geq 1$.

First, we argue that $lv_i(C_v) \leq \zeta \cdot \tau$. Since cluster C_v is in $\mathcal{C}_i$, there exists a vertex $u \in C_v$ such that $\delta_G(u,K) < \zeta^i$. Let $P_u := (u_1, u_2, \ldots u_s)$ be a shortest path from $u = u_1$ to K (with $\|P_u\| < \zeta^i$), let ℓ be the largest index such that $u_1, u_2, \ldots u_\ell \in V \setminus V_{i-1}$, and define the prefix $Q := (u_1, u_2, \ldots u_\ell)$ of P_u; note that $\ell < s$ and $u_{\ell+1} \in V_{i-1}$. Since $\|Q\| < \|P_u\| < \zeta^i$, we can greedily partition Q into $\zeta' \leq \zeta$ sub-paths $Q_1, \ldots, Q_{\zeta'}$, each of length at most ζ^{i-1}, connected via edges of weight less than ζ^i; that is, Q is obtained as the concatenation $Q_1 \circ e_1 \circ Q_2 \circ e_2 \ldots \circ e_{\zeta'-1} \circ Q_{\zeta'}$, where $\|Q_j\| < \zeta^{i-1}$ and $\|e_j\| < \zeta^i$ for each j. Consider the β-approximate (τ, ζ^{i-1})-scattering partition $\mathcal{P}_i$ (provided by Corollary 2.4), used in the ith iteration, on the subgraph $G_i = G[V \setminus V_{i-1}]$ induced on the unassigned vertices. For each j, the sub-path Q_j of Q is contained in G_i and it satisfies $\|Q_j\| \leq \zeta^{i-1}$, thus there exists a β-approximate path Q'_j between the endpoints of Q_j that is scattered by τ' clusters, with $\tau' \leq \tau$, and each edge of Q'_j is of weight at most ζ^{i-1}. The path $Q'_1 \circ e_1 \circ Q'_2 \circ e_2 \ldots \circ e_{\zeta'-1} \circ Q'_{\zeta'}$ obtained from Q by replacing each sub-path Q_j by its scattered path Q'_j, is a path from u_1 to u_ℓ intersecting at most $\zeta \cdot \tau$ clusters in $\mathcal{C}_i$. Since u_ℓ is in a cluster of level 1 (because $u_{\ell+1}$ is in V_{i-1}, which is of level 0), $lv_i(C_v) \leq \zeta \cdot \tau$, as required.

We then show that $\delta_G(v, f(v)) \leq lv_i(C_v) \cdot 2 \cdot \zeta^i + 3\tau \cdot \zeta^i$ by induction on the (ith-iteration) level of C_v. We employ a double induction, one on the iteration i and the other on the level of C_v; aiming to avoid confusion, we shall refer to the former as the "outer induction" and to the latter as the "inner induction".

Let x be the linking vertex of C_v; in particular, we have $f(v) = f(x)$. Let x_v be the vertex in C_v such that $(x, x_v) \in E$ and $w(x, x_v) \leq \zeta^i$. For the basis $lv_i(C_v) = 1$ of the inner induction, x is assigned during iteration $i' < i$. By the outer induction hypothesis for iteration i' (i.e., substituting i with i' in Eq. 1), we obtain $\delta_G(x, f(x)) \leq 3\tau \cdot \zeta^{i'+1} \leq 3\tau \cdot \zeta^i$. By the triangle inequality and since $\zeta > 1$:

$$\begin{aligned} \delta_G(v, f(v)) &\leq \delta_G(v, x_v) + \delta_G(x_v, x) + \delta_G(x, f(v)) \\ &\leq \zeta^{i-1} + \zeta^i + \delta_G(x, f(x)) \leq \zeta^{i-1} + \zeta^i + 3\tau \cdot \zeta^i \leq 2 \cdot \zeta^i + 3\tau \cdot \zeta^i. \end{aligned} \tag{2}$$

For the inner induction step, consider the case $lv_i(C_v) > 1$. Let C_x be the cluster in $\mathcal{P}_i$ containing x; in particular, we have $lv_i(C_x) = lv_i(C_v) - 1$. By the inner induction hypothesis on the level of C_x, we have

$\delta(x, f(x)) \leq lv_i(C_x) \cdot 2 \cdot \zeta^i + 3\tau \cdot \zeta^i$. Using the triangle inequality again, we have:

$$\begin{aligned}\delta_G(v, f(v)) &\leq \delta_G(v, x_v) + \delta_G(x_v, x) + \delta_G(x, f(v)) \\ &\leq \zeta^{i-1} + \zeta^i + \delta_G(x, f(x)) \leq \zeta^{i-1} + \zeta^i + lv_i(C_x) \cdot 2 \cdot \zeta^i + 3\tau \cdot \zeta^i \\ &\leq 2 \cdot \zeta^i + (lv_i(C_v) - 1) \cdot 2 \cdot \zeta^i + 3\tau \cdot \zeta^i = lv_i(C_v) \cdot 2 \cdot \zeta^i + 3\tau \cdot \zeta^i,\end{aligned} \tag{3}$$

which completes the inner induction step.

Since $lv_i(C_v) \leq \zeta \cdot \tau$ and as $\zeta > 3$, it follows that $\delta(v, f(v)) \leq 3\tau \cdot \zeta^{i+1}$, which completes the outer induction step. The claim follows. □

Now we are ready to prove Lemma 2.2.

Proof (of Lemma 2.2): We prove that our algorithm returns a minor of G that satisfies the SPR conditions. By the description of the algorithm, it is immediate that the subgraph induced by the vertex set $f^{-1}(t)$ is connected, for each $t \in K$. Thus, it remains to prove that the minor M induced by f is a distance preserving minor of G with distortion $O(\tau^8 \cdot \beta^5)$.

Consider an arbitrary pair of terminals t and t'. Let $P := (v_1, v_2, \ldots, v_{|P|})$ be a shortest path between $v_1 := t$ and $v_{|P|} := t'$. For each subpath $I := (v_\ell, v_{\ell+1}, \ldots v_r)$ of P, let I^+ denote the *extended subpath* $(v_{\ell-1}, v_\ell, v_{\ell+1}, \ldots v_r, v_{r+1})$; we define $v_0 := v_1$ and $v_{|P|+1} := v_{|P|}$ for technical convenience. Partition P into a set $\mathcal{I}$ of subpaths called *intervals* such that for each subpath $I \in \mathcal{I}$ between v_ℓ and v_r:

$$\|I\| \leq \eta \cdot \delta_G(v_\ell, K) \leq \|I^+\|, \tag{4}$$

where $\eta := \frac{1}{4\zeta}$. It is easy to verify that $\mathcal{I}$ can be constructed greedily from P.

Consider an arbitrary interval $I = (v_\ell, v_{\ell+1}, \ldots v_r) \in \mathcal{I}$. Let $u \in I$ be a vertex that is assigned in iteration i, and assume no vertex of I was assigned prior to iteration i. Since u is assigned in iteration i, u belongs to a cluster C in $\mathcal{C}_i$, which is the subset of clusters that contain at least one vertex in $\mathcal{R}_i$, among the β-approximate (τ, ζ^{i-1})-scattering partition $\mathcal{P}_i$ computed at the ith iteration. Hence, by definition, C has strong diameter at most ζ^{i-1} and there exists a vertex $u' \in C$ such that $\delta_G(u', K) < \zeta^i$, implying that

$$\delta_G(u, K) \leq \delta_G(u, u') + \delta_G(u', K) < \zeta^{i-1} + \zeta^i < 2\zeta^i. \tag{5}$$

By Eq. 4 and the triangle inequality,

$$\delta_G(v_\ell, K) \leq \delta_G(v_\ell, u) + \delta_G(u, K) \leq \|I\| + \delta_G(u, K) \leq \eta \cdot \delta_G(v_\ell, K) + \delta_G(u, K),$$

which together with Eq. 5 and the fact that $\eta < 1/2$ yields

$$\delta_G(v_\ell, K) \leq \frac{\delta_G(u, K)}{1 - \eta} < \frac{2\zeta^i}{1 - \eta} < 4\zeta^i. \tag{6}$$

By Eq. 4 and Eq. 6,

$$\delta_G(v_\ell, v_r) = \|I\| \leq \eta \cdot \delta_G(v_\ell, K) < \eta \cdot 4\zeta^i = \zeta^{i-1}, \tag{7}$$

where the last inequality holds as $\eta = \frac{1}{4\zeta}$.

At the beginning of iteration i, all vertices of I are unassigned, i.e., I is in $G_i = G[V \setminus V_{i-1}]$, and Eq. 7 yields $\delta_{G_i}(v_\ell, v_r) = \delta_G(v_\ell, v_r) < \zeta^{i-1}$. At the ith iteration a β-approximate (τ, ζ^{i-1})-scattering partition $\mathcal{P}_i$ on G_i is computed, thus there exists a β-approximate (τ, ζ^{i-1})-scattered path I' in G_i from v_ℓ to v_r that is scattered by at most τ clusters in $\mathcal{P}_i$, with $\|I'\| \leq \beta \cdot \zeta^{i-1}$. A path is called a *detour* if its first and last vertices are assigned to the same terminal. Since vertices in the same cluster will be assigned to the same

terminal, at the end of iteration i, I' can be greedily partitioned into at most τ detours and $\tau+1$ subpaths that contain only unassigned vertices; in other words, we can write $I' := P_1 \circ Q_1 \circ \ldots \circ P_\rho \circ Q_\rho \circ P_{\rho+1}$, where $\rho \leq \tau$, $Q_1, Q_2, \ldots Q_\rho$ are detours, and each of the (possibly empty) sub-paths $P_1, P_2, \ldots P_{\rho+1}$ contains only unassigned vertices at the end of iteration i.

Fix an arbitrary index $j \in [1\,..\,\rho+1]$. Let a_j and b_j be the first and last vertices of P_j; it is possible that $a_j = b_j$. Since $\|I'\| \leq \beta \cdot \zeta^{i-1}$ and as $\beta < \zeta$, we have

$$\delta_G(a_j, b_j) \leq \|P_j\| \leq \|I'\| \leq \beta \cdot \zeta^{i-1} < \zeta^i. \tag{8}$$

At the beginning of iteration $i+1$, all vertices of P_j are unassigned by definition, hence P_j is in $G_{i+1} = G[V \setminus V_i]$ and by Eq. 8, $\delta_{G_{i+1}}(a_j, b_j) \leq \|P_j\| < \zeta^i$. At the $(i+1)$th iteration a β-approximate (τ, ζ^i)-scattering partition $\mathcal{P}_{i+1}$ on G_{i+1} is computed, thus there exists a β-approximate (τ, ζ^i)-scattered path P'_j in G_{i+1} from a_j to b_j that is scattered by at most τ clusters in $\mathcal{P}_{i+1}$, with $\|P'_j\| \leq \beta \cdot \zeta^i$.

Next, consider the path $I'' := P'_1 \circ Q_1 \circ \ldots \circ P'_\rho \circ Q_\rho \circ P'_{\rho+1}$. By Eq. 7 we have

$$\|I''\| \leq \|I\| + \sum_{j=1}^{\rho+1} \|P'_j\| \leq \zeta^{i-1} + (\tau+1)\beta \cdot \zeta^i \leq (\tau+2)\beta \cdot \zeta^i \tag{9}$$

Since no vertex in I (in particular, v_ℓ) was assigned prior to iteration i, Claim 2.6 yields $\delta_G(v_\ell, K) \geq \zeta^{i-1}$. Eq. 4 yields $\|I^+\| \geq \eta \cdot \delta_G(v_\ell, K) \geq \eta \cdot \zeta^{i-1}$, and as $\eta = \frac{1}{4\zeta}$ we obtain

$$\|I''\| \leq (\tau+2)\beta \cdot \zeta^i \leq 4\zeta^2(\tau+2)\beta \cdot \|I^+\|. \tag{10}$$

Next, we argue that all vertices in I'' are assigned at the end of iteration $i+1$. Let w be an arbitrary vertex in I''; by Claim 2.6, it suffices to show that $\delta_G(w, K) < \zeta^{i+1}$. Recall that u is a vertex of I that is assigned in iteration i. By Eq. 5, Eq. 7, Eq. 9 and the triangle inequality,

$$\begin{aligned} \delta_G(w,K) &\leq \delta_G(v_\ell, K) + \delta_G(v_\ell, w) \leq \delta_G(v_\ell, u) + \delta_G(u, K) + \delta_G(v_\ell, w) \\ &\leq \|I\| + \delta_G(u,K) + \|I''\| < \zeta^{i-1} + 2\zeta^i + (\tau+2)\beta \cdot \zeta^i < \zeta^{i+1}, \end{aligned} \tag{11}$$

where the last inequality holds since $\zeta = c \cdot \beta \cdot \tau$ for a sufficiently large constant c.

Hence, every vertex in P'_j is assigned by iteration $i+1$, for every $j \in [1\,..\,\rho+1]$. Then, P'_j could be greedily partitioned into at most τ detours, as before with I', but we have no subpaths of unassigned vertices in I'', since every vertex in I'' must be assigned by the end of iteration $i+1$. We have thus shown that I'' can be partitioned into at most $O(\tau^2)$ detours $D_1, D_2, \ldots D_g$, with $g = O(\tau^2)$. For each $j \in [1\,..\,g]$, let x_j and y_j be the first and last vertices in D_j. Because I'' are partitioned greedily into *maximal* detours, one has $f(y_j) \neq f(x_{j+1})$ for all j. Observe that there exists an edge between $f(x_j)$ and $f(x_{j+1})$ in the SPR minor M for each $j \in [1\,..\,g-1]$, since $f(x_j) = f(y_j) \in K$ and $(y_j, x_{j+1}) \in E$. Consequently, by the triangle inequality, Corollary 2.7 and Eq. 10,

$$\begin{aligned} \delta_M(f(v_\ell), f(v_r)) &\leq \sum_{j=1}^{g-1} \delta_M(f(x_j), f(x_{j+1})) = \sum_{j=1}^{g-1} \delta_G(f(x_j), f(x_{j+1})) \\ &\leq \sum_{j=1}^{g-1} \left[\delta_G(x_j, f(x_j)) + \delta_G(x_j, x_{j+1}) + \delta_G(x_{j+1}, f(x_{j+1}))\right] \\ &\leq 2\sum_{j=1}^{g} \delta_G(x_j, f(x_j)) + \sum_{j=1}^{g-1} \delta_G(x_j, x_{j+1}) \leq 2\sum_{j=1}^{g} \delta_G(x_j, f(x_j)) + \|I''\| \\ &\leq 6\tau\zeta^2 \sum_{j=1}^{g} \delta_G(x_j, K) + 4\zeta^2(\tau+2)\beta \cdot \|I^+\|. \end{aligned} \tag{12}$$

For every vertex $v'' \in I''$, we have

$$\begin{aligned}\delta_G(v'',K) &\le \delta_G(v'',v_\ell) + \delta_G(v_\ell,K) \le \|I''\| + \delta_G(v_\ell,K)\\ &\le 4\zeta^2(\tau+2)\beta\cdot\|I^+\| + \frac{\|I^+\|}{\eta} \le 4\zeta^2(\tau+3)\beta\cdot\|I^+\|,\end{aligned} \tag{13}$$

where the penultimate inequality holds by Eq. 4 and Eq. 10 and the last inequality holds since $\eta = \frac{1}{4\zeta}$. We remark that Eq. 13 also holds for any vertex $v' \in I$, which will be used below for deriving Eq. 17. Hence, for every $j \in [1\,..\,g]$, $\delta_G(x_j,K) \le 4\zeta^2(\tau+3)\beta\cdot\|I^+\|$; plugging this in Eq. 12 yields:

$$\delta_M(f(v_\ell),f(v_r)) \le 24\zeta^4\tau(\tau+3)\beta g\cdot\|I^+\| + 4\zeta^2(\tau+2)\beta\cdot\|I^+\| = O(\zeta^4\cdot\tau^4\cdot\beta)\cdot\|I^+\|. \tag{14}$$

Next, we bound the distance between t and t' in M. So far we fixed an arbitrary interval $I = (v_\ell, v_{\ell+1}, \ldots v_r) \in \mathcal{I}$. Writing $\mathcal{I} = \{I_1, I_2, \ldots I_s\}$, we have $\sum_{j=1}^{s}\|I_j\| = \|P\| = \delta_G(t,t')$, hence

$$\sum_{j=1}^{s}\|I_j^+\| \le 2\|P\| = 2\cdot\delta_G(t,t'). \tag{15}$$

For each I_j, let v_ℓ^j and v_r^j be the first and last vertices of I_j. For each $j \in [1\,..\,s-1]$, since $(v_r^j, v_\ell^{j+1}) \in E$, either $(f(v_r^j), f(v_\ell^{j+1})) \in E(M)$ or $f(v_r^j) = f(v_\ell^{j+1})$, thus we have $\delta_M(f(v_r^j), f(v_\ell^{j+1})) = \delta_G(f(v_r^j), f(v_\ell^{j+1}))$. Hence, using the triangle inequality:

$$\begin{aligned}\delta_M(t,t') &\le \sum_{j=1}^{s-1}\Big(\delta_M(f(v_\ell^j),f(v_r^j)) + \delta_M(f(v_r^j),f(v_\ell^{j+1}))\Big) + \delta_M(f(v_\ell^s),f(v_r^s))\\ &\le O(\zeta^4\cdot\tau^4\cdot\beta)\cdot\sum_{j=1}^{s}\|I_j^+\| + \sum_{j=1}^{s-1}\delta_M(f(v_r^j),f(v_\ell^{j+1})) \qquad \text{(by Eq. 14)}\\ &\le O(\zeta^4\cdot\tau^4\cdot\beta)\cdot\delta_G(t,t') + \sum_{j=1}^{s-1}\delta_M(f(v_r^j),f(v_\ell^{j+1})). \qquad \text{(by Eq. 15)}\\ &= O(\zeta^4\cdot\tau^4\cdot\beta)\cdot\delta_G(t,t') + \sum_{j=1}^{s-1}\delta_G(f(v_r^j),f(v_\ell^{j+1})).\end{aligned} \tag{16}$$

Using the triangle inequality again, we have:

$$\begin{aligned}\sum_{j=1}^{s-1}\delta_G(f(v_r^j),f(v_\ell^{j+1})) &\le \sum_{j=1}^{s-1}\Big(\delta_G(f(v_r^j),v_r^j) + \delta_G(v_r^j,v_\ell^{j+1}) + \delta_G(v_\ell^{j+1},f(v_\ell^{j+1}))\Big)\\ &\le \sum_{j=1}^{s-1}\delta_G(v_r^j,v_\ell^{j+1}) + \sum_{j=1}^{s}\Big(\delta_G(v_\ell^j,f(v_\ell^j)) + \delta_G(v_r^j,f(v_r^j))\Big)\\ &\le \|P\| + 3\tau\zeta^2\cdot\sum_{j=1}^{s}\Big(\delta_G(v_\ell^j,K) + \delta_G(v_r^j,K)\Big) \qquad \text{(by Corollary 2.7)}\\ &\le \delta_G(t,t') + 3\tau\zeta^2\cdot 4\zeta^2(\tau+3)\beta\cdot\sum_{j=1}^{s}(\|I_j^+\| + \|I_j^+\|) \qquad \text{(by Eq. 13)}\\ &\le O(\zeta^4\cdot\tau^2\cdot\beta)\cdot\delta_G(t,t') \qquad \text{(by Eq. 15).}\end{aligned} \tag{17}$$

Plugging Eq. 17 into Eq. 16, we obtain $\delta_M(t,t') = O(\zeta^4\cdot\tau^4\cdot\beta)\cdot\delta_G(t,t')$. Since $\zeta = O(\beta\cdot\tau)$, we conclude that $\delta_M(t,t') = O(\tau^8\cdot\beta^5)$, as required. □

3 Buffered cop decompositions for minor-free graphs

notation	*meaning*
$\mathcal{T}$	*partition tree*: nodes of $\mathcal{T}$ are supernodes
$\hat{\mathcal{T}}$	*expansion of* $\mathcal{T}$
η	*supernode*: *induced subgraph* on its vertices (one may identify η with these vertices); η contains T_η initially and may only grow
$\mathrm{dom}(\eta)$	*domain of* η: subgraph induced by the union of all supernodes in the subtree of $\mathcal{T}$ rooted at η
T_η	*tree skeleton*: SSSP tree in $\mathrm{dom}(\eta)$ (remain fixed)
$\mathcal{S}$	*set of supernodes*: $\mathcal{S}$ starts empty and grows, and each supernode may grow
witness v_S	vertex adjacent to some vertex in supernode S
H sees S	subgraph H has a witness vertex v_S to S
$\mathcal{S}\|_H$	set of supernodes in $\mathcal{S}$ subgraph H can see
$\mathrm{dom}_{\mathcal{S}}(\eta)$	*domain of* η *with respect to* $\mathcal{S}$: may shrink; the final $\mathrm{dom}_{\mathcal{S}}(\eta)$ is $\mathrm{dom}(\eta)$
$\partial H'_{\downarrow X}$	*boundary vertices*: vertices in $G \setminus H'$ that are (1) adjacent to H', and (2) in $\mathrm{dom}_{\mathcal{S}}(X)$
$\mathcal{N}H'_X$	*buffer vertices*: unassigned vertices in H' within distance (in $\mathrm{dom}_{\mathcal{S}}(X)$) Δ/r of $\partial H'_{\downarrow X}$
$\eta_{\mathcal{S}}$	vertices assigned to η by the current $\mathcal{S}$

Table 1. Glossary for the construction of buffered cop decompositions.

Buffered cop decomposition. Let G be a graph. A *supernode* η with *skeleton* T_η and *radius* Δ is an induced subgraph η of G containing a tree T_η where every vertex in η is within distance Δ of T_η for some real number Δ, where distance is measured with respect to η. We occasionally abuse notation and use η to refer to the set of vertices in η, rather than the subgraph. A *buffered cop decomposition* for G is a partition of $V(G)$ into vertex-disjoint supernodes, together with a tree $\mathcal{T}$ called the *partition tree*, whose nodes $V(\mathcal{T})$ are the supernodes of G. For any supernode η, the *domain* $\mathrm{dom}(\eta)$ denotes the subgraph induced by the union of all vertices in supernodes in the subtree of $\mathcal{T}$ rooted at η.

Definition 3.1. *A* (Δ, γ, w)*-buffered cop decomposition for* G *is a buffered cop decomposition* $\mathcal{T}$ *that satisfies the following properties:*

- [Supernode radius.] *Every supernode* η *has radius at most* Δ*.*
- [Shortest-path skeleton.] *For every supernode* η*, the skeleton* T_η *is an SSSP tree in* $\mathrm{dom}(\eta)$*, with at most* $w-1$ *leaves.*
- [Supernode buffer.] *Let* η *be a supernode, and let* X *be another supernode that is an ancestor of* η *in the partition tree* $\mathcal{T}$*. Then either* η *and* X *are adjacent*[6] *in* G*, or for every vertex* v *in* $\mathrm{dom}(\eta)$*, we have* $\delta_{\mathrm{dom}(X)}(v, X) \geq \gamma$*.*
- [Tree decomposition.] *For every supernode* η*, define the bag of* η*, denoted* $B_\eta \subseteq V(\mathcal{T})$*, to be a set containing* η *and all ancestor supernodes adjacent to* η *in* G*. Then* $|B_\eta| \leq w$ *for every* η*. Further, define* $\hat{B}_\eta \subseteq V(G)$ *to be the set of vertices contained in some supernode in* B_η*, that is,* $\hat{B}_\eta = \bigcup_{X \in B_\eta} V(X)$*; and define the expansion of* $\mathcal{T}$*, denoted* $\hat{\mathcal{T}}$*, to be a tree isomorphic to* $\mathcal{T}$ *with vertex set* $\{\hat{B}_\eta\}_{\eta \in V(\mathcal{T})}$*. Then* $\hat{\mathcal{T}}$ *is a tree decomposition of* G*.*

[6] that is, there is some $a \in \eta$ and $b \in X$ such that (a, b) is an edge in G

We say that $\mathcal{T}$ has radius Δ, buffer γ, and width w.

Our definition of buffered cop decomposition is somewhat different from the cop decomposition in the prior work [And86, AGG+14]. Recall that the cop decomposition in prior work is a tree decomposition, where each bag of the tree decomposition can be partitioned into $r-1$ supernodes. Here each node of the partition tree $\mathcal{T}$ in our definition is exactly one supernode. The expansion $\check{\mathcal{T}}$ in our definition is a cop decomposition in the sense of prior work. We find that this alternative definition helps to simplify the presentation of our buffer-creating algorithm significantly. See Figure 3 for an illustration of buffered cop decomposition, and Table 1 for a glossary of terminologies related to the construction of buffered cop decompositions.

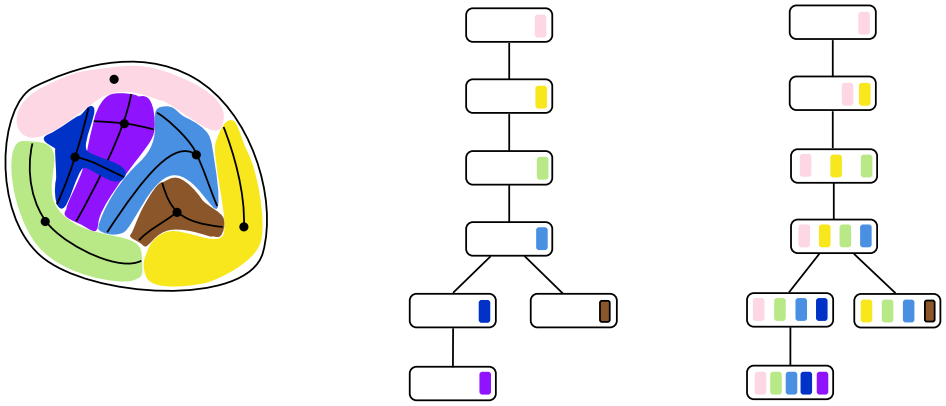

Figure 3. Left: A *non-planar* graph G with a partition into supernodes. Notice that the purple cluster is connected and goes behind the dark blue supernode. Middle: The partition tree $\mathcal{T}$ of a buffered cop decomposition for G. The supernode buffer property guarantees that any path between the brown and pink supernodes is of length at least γ. Right: The expansion of $\mathcal{T}$, where each bag contains at most 5 supernodes.

This section is devoted to proving the following theorem.

Theorem 3.2. *Let G be a K_r-minor-free graph, and let Δ be a positive number. Then G admits a $(\Delta, \Delta/r, r-1)$-buffered cop decomposition.*

We emphasize that in the construction of buffered cop decomposition, the most interesting property that we need to guarantee is the supernode buffer property. This property says that if a supernode X gets "cut off" from part of the graph, there is a "buffer region" of at least γ between X and that part of the graph. More precisely, let G' be the subgraph of G induced by vertices in descendant supernodes of X that are not adjacent to X. (That is, X is "cut off" from G' by the descendant supernodes that are adjacent to X.) The supernode buffer property in Definition 3.1 implies that $\delta_{\mathrm{dom}(X)}(v, X) \geq \gamma$ for every $v \in V(G')$. The construction of [AGG+14] produces a cop decomposition with the other three properties; that is, a buffered cop decomposition with radius Δ and width $r-1$. A delicate argument in [AGG+14] shows that their construction achieves something similar to a supernode buffer of Δ/r *in expectation*.

Review of the construction of [AGG+14]. The construction of [AGG+14] iteratively builds a collection $\mathcal{S}$ of supernodes of a graph G. At each point in the algorithm, they process a subgraph H of G by creating a new supernode η in H, and then recursing on the connected components of $H \setminus \eta$.

To describe how to create each new supernode η, we introduce some terminology. A subgraph H *sees* a supernode S if (1) S is disjoint from H, and (2) there exists some *witness vertex* v_S in H that is adjacent to a vertex in S. For any subgraph H, let $\mathcal{S}|_H$ be the set of supernodes that H sees. The algorithm of

[AGG+14] guarantees that, if G excludes a K_r-minor, the subgraph H (at any point in the algorithm) sees at most $r-2$ previously-created supernodes. Their algorithm has the following steps:

1. *Initialize a new supernode η.*

   Choose an arbitrary vertex v in H. Build a shortest-path tree T_η in H that connects v to an arbitrary witness vertex for every supernode seen by H. Initialize supernode $\eta \leftarrow T_\eta$ with skeleton T_η.

2. *Expand η, to guarantee supernode buffer property in expectation.*

   Let γ be a random number between 0 and $C \cdot \Delta$ (for some constant C) drawn from a truncated exponential distribution with rate $O(r/\Delta)$, meaning that $\mathbb{E}[\gamma] = O(\Delta/r)$. Assign every vertex within distance γ of T_η to be a part of supernode η (where distances are with respect to H).

3. *Recurse.*

   Recurse on each connected component in the graph in $H \setminus \eta$.

The subgraph H is initially selected to be G. The buffered cop decomposition is implicit from the recursion tree. They show that at any point in the algorithm, the set of supernodes seen by H forms a model of a complete graph (see Lemma 3.7); this proves the bag width property. The tree decomposition, radius, and shortest-path skeleton properties are all straightforward to verify. The proof of the "expected" supernode buffer property is quite complicated, and requires dealing with the fact that γ is drawn from a truncated exponential distribution rather than a normal exponential distribution.

Remark 3.3. *While the buffer guarantee of [AGG+14] is in expectation, the nature of their buffer property is somewhat stronger than ours: they guarantee that whenever a new skeleton T_η is added and cuts off the shortest path from a vertex v to another skeleton X (which could still be adjacent to T_η), the distance from v to T_η is smaller than the distance from v to X by $O(\Delta/r)$ in expectation. Here we only guarantee the distance reduction from v to X when X and T_η are not adjacent.*

3.1 Construction

We modify the algorithm of Abraham *et al.* [AGG+14] to obtain the (deterministic) supernode buffer property. Throughout our algorithm, we maintain the global variables $\mathcal{S}$, indicating the set of supernodes, and $\mathcal{T}$, indicating the partition tree. At any moment during the execution of our algorithm, some vertices of graph G will already be assigned to supernodes, and some vertices will be unassigned. At the end of the execution, all vertices will be assigned by $\mathcal{S}$. At each stage of the algorithm, we (1) select some unassigned vertices to become a new supernode η, (2) assign some unassigned vertices to existing supernodes (*not necessarily* η) to guarantee the supernode buffer property, and (3) recurse on connected components induced by the remaining unassigned vertices.

Our main procedure is BUILDTREE($\mathcal{S}, H$), which takes as input a connected subgraph H induced by unassigned vertices in G. It assigns vertices in H to supernodes in $\mathcal{S}$, and returns a buffered cop decomposition. Figure 4 gives an example; Figure 5 gives the complete pseudocode. The algorithm consists of the following steps:

1. *Initialize a new supernode.*

   Choose an arbitrary vertex v in H. Build a shortest path tree T_η in H that connects v to an arbitrary witness vertex for every supernode seen by H. Initialize supernode η to be the subgraph of G induced by all vertices of T_η; set T_η to be the skeleton of η; and add η to $\mathcal{S}$. Define the domain of η with respect to $\mathcal{S}$, $\mathrm{dom}_{\mathcal{S}}(\eta)$, to be the set of all vertices in H that are not assigned (by $\mathcal{S}$) to any supernode above η in the partition tree $\mathcal{T}$; initially $\mathrm{dom}_{\mathcal{S}}(\eta) = H$, and at the end of the algorithm

it will hold that $\mathrm{dom}_{\mathcal{S}}(\eta) = \mathrm{dom}(\eta)$. (Notice that η will grow and $\mathrm{dom}_{\mathcal{S}}(\eta)$ will shrink over the course of the algorithm as $\mathcal{S}$ changes, though T_η will remain unchanged. See Claim 3.6(3).)

2. *Assign vertices to existing supernodes, to guarantee the supernode buffer property.*

   For each connected component H' of $H \setminus \eta$, consider the set of supernodes $\mathcal{X}$ that *can* be seen by H but *cannot* be seen by H'. These supernodes are "cut off" from H'. In this step, we identify every *currently unassigned* vertex that could be close to a cut-off supernode, and assign those vertices to some existing supernode (possibly to the newly-created η).

   In more detail: For each X in $\mathcal{X}$, define the *boundary vertices* $\partial H'_{\downarrow X}$ to be the set of vertices in $G \setminus H'$ that are (1) adjacent to H', and (2) in $\mathrm{dom}_{\mathcal{S}}(X)$. Our algorithm will maintain the invariant that all vertices adjacent to H' (in particular, all vertices in $\partial H'_{\downarrow X}$) have already been assigned to a supernode by $\mathcal{S}$; see Invariant 3.4 for the formal statement. Define the set of *buffer vertices* $\mathcal{N}H'_X$ to be the set of unassigned vertices in H' within distance Δ/r of $\partial H'_{\downarrow X}$, where distance is measured with respect to $\mathrm{dom}_{\mathcal{S}}(X)$. Assign each vertex in $\mathcal{N}H'_X$ to the same supernode as a closest vertex in $\partial H'_{\downarrow X}$, breaking ties consistently and measuring distance with respect to $\mathrm{dom}_{\mathcal{S}}(X)$; notice that "the supernode of a vertex in $\partial H'_{\downarrow X}$" is well-defined because of Invariant 3.4.

   This procedure may cut off H' from another supernode, even if H' may originally have been able to see that supernode (even η itself could become cut off at this point); and it may break H' into multiple connected components. Repeat this assignment process on each connected component until we have dealt with all supernodes that have been cut off.

   In Lemma 3.12, we show that this procedure guarantees that the supernode buffer property holds. It will suffice to show that, in this step, we assign every vertex in H' that could become close to some cut-off supernode X, *even if X grows in the future*. Crucially, in this step we assign every vertex in H' that is within Δ/r distance of the boundary $\partial H'_{\downarrow X}$. It would *not* suffice to just assign vertices within Δ/r distance of X in the current step, because X could potentially grow in the future, and the distance from a vertex in H' to X could shrink. We show that even if X expands in the future, it remains disjoint from X'; further, we show that every path in $\mathrm{dom}(X)$ from a vertex in H' to a vertex outside of H passes through some boundary vertex in $\partial H'_{\downarrow X}$. Thus, every vertex in H' is closer[7] to $\partial H'_{\downarrow X}$ than to X (which are always outside H'), even if X expands in the future. This means that the vertices of $\mathcal{N}H'_X$ form a buffer of Δ/r between X and the unassigned vertices of H'. Note that we assign each vertex in $\mathcal{N}H'_X$ to some supernode that is not X (as H' does not see X, no vertex in $\mathcal{N}H'_{\downarrow X}$ is in X).

   This procedure is called GrowBuffer$(\mathcal{S}, \mathcal{X}, H')$; see pseudocode in Figure 6. It takes as input a subgraph H' and a list $\mathcal{X}$ of supernodes that have been cut off from H'. It assigns some vertices in H' to existing supernodes in $\mathcal{S}$.

3. *Recurse.*

   For each connected component H' in the graph induced by unassigned vertices, recursively call BuildTree$(\mathcal{S}, H')$.

To initialize, let $\mathcal{S} \leftarrow \varnothing$, and call BuildTree$(\mathcal{S}, G)$ to produce a buffered cop decomposition $\mathcal{T}$ for G. Throughout the algorithm, we maintain the following invariant:

Invariant 3.4. *Suppose that call C, whether it is* GrowBuffer$(\mathcal{S}, \mathcal{X}, H)$ *or* BuildTree$(\mathcal{S}, H)$*, is made at some point in the algorithm. At the time call C is made, every vertex in H is unassigned, and every vertex in $G \setminus H$ that is adjacent to H is already assigned to some supernode.*

[7] We assume the weight of every edge in G is nonzero.

(We say that a vertex v of graph G is *adjacent* to a subgraph H of G if (1) v is in $G \setminus H$, and (2) there is an edge in G between v and some vertex in H.) The invariant is clearly true when we make the initial call BUILDTREE$(\varnothing, G)$, and it is preserved throughout the algorithm: a recursive call to GROWBUFFER$(\mathcal{S}, \mathcal{X}, H)$ or BUILDTREE$(\mathcal{S}, H)$ is only made if H is a maximal connected component induced by unassigned vertices. Because we maintain this invariant, the procedure GROWBUFFER is well-defined.

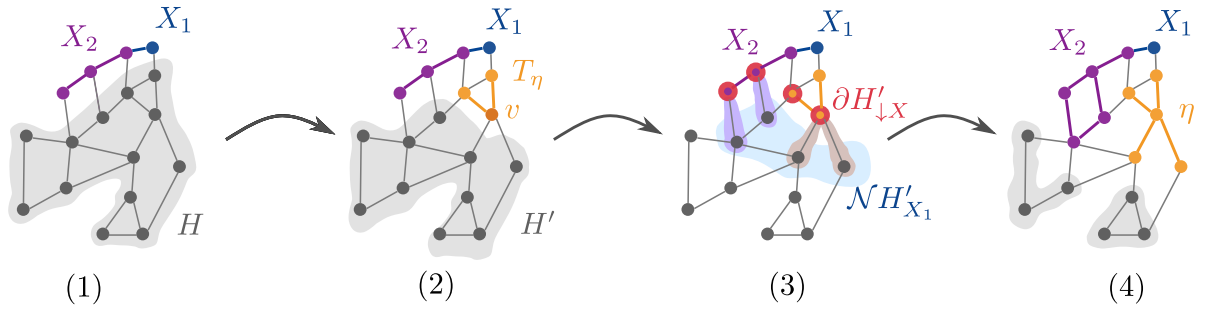

Figure 4. Example of one iteration of BuildTree(H). From left to right: (1) The graph G before an iteration, with H being a connected component of unassigned vertices. (2) Pick an arbitrary vertex v in H and compute T_η by taking shortest paths from v to X_1 and X_2. (3) Supernode X_1 is cut off from H', so find the set $\mathcal{N}H'_{X_1}$ (vertices in H' within distance Δ/r of the boundary of $\partial H'_{\downarrow X_1}$) and assign each vertex of $\mathcal{N}H'_{X_1}$ to the supernode containing the closest boundary vertex, to ensure the buffer property. (4) There are now two connected components, one of which is cut off from X_2, so we must grow a buffer for X_2 in that component.

A remark on the global variable. The procedure BUILDTREE$(\mathcal{S}, H)$ is recursive: it initializes a supernode, calls GROWBUFFER, and then recursively calls BUILDTREE$(\mathcal{S}, H'_i)$ on disjoint subgraphs H'_i. Each of these recursive calls modifies the same global variable $\mathcal{S}$. However, the modifications to $\mathcal{S}$ that are made by each call $C_i :=$ BUILDTREE$(\mathcal{S}, H'_i)$ *do not* affect the execution of any *sibling* calls $C_j :=$ BUILDTREE$(\mathcal{S}, H'_j)$. Only the ancestors of C_i in the recursion tree affect the execution of C_i.

Before proving this observation, we introduce the following important terminology. We say that a call $C :=$ BUILDTREE$(\mathcal{S}, H)$ occurs *above* (resp. *below*) a call $C' :=$ BUILDTREE$(\mathcal{S}, H')$ if C is an ancestor (resp. descendent) of C' in the recursion tree. If two calls to BUILDTREE are in different branches of the recursion tree, then they are neither above nor below each other. Intuitively, "C is above C'" means that C was a *relevant* call that happened *before* C'. Similarly, we say "supernode η_1 is initialized *above* supernode η_2" if the instance of BUILDTREE that initialized η_1 occurred above the instance of BUILDTREE that initialized η_2. Finally, we say that "supernode η is initialized *above* a call $C :=$ GROWBUFFER$(\mathcal{S}, \mathcal{X}, H)$" if the call to BUILDTREE that initialized η is above *or is the same as* the call to BUILDTREE that caused C to be called. Note that the algorithm never initializes a new supernode during a GROWBUFFER call.

With this terminology, we can state a stronger version of Invariant 3.4.

Invariant 3.5. *Suppose that call C, whether it is* GROWBUFFER$(\mathcal{S}, \mathcal{X}, H)$ *or* BUILDTREE$(\mathcal{S}, H)$*, is made at some point in the algorithm. At the time call C is made, every vertex in H is unassigned, and every vertex in $G \setminus H$ that is adjacent to H was already assigned during a call above C to some supernode initialized above C.*

Indeed, when some call $\tilde{C}$ (whether it is GROWBUFFER or BUILDTREE) makes call C on some subgraph H, the graph H is a maximal connected component of unassigned vertices — and crucially, this connected component is with respect to the assignments $\mathcal{S}$ *before any calls to* GROWBUFFER *or* BUILDTREE *are made by* $\tilde{C}$. Thus, every vertex adjacent to H has been assigned to some existing supernode (before any sibling calls of C are made), meaning it was initialized above C.

We now prove that the execution of a call C, whether it is GROWBUFFER$(\mathcal{S}, \mathcal{X}, H)$ or BUILDTREE$(\mathcal{S}, H)$, depends only on H and the vertices assigned to supernodes above C. Indeed, a call to BUILDTREE$(\mathcal{S}, H)$ uses $\mathcal{S}$ to determine the supernodes seen by H, which is determined by the vertices adjacent to H (and thus, by Invariant 3.5, by calls above C and not by sibling calls). A call to GROWBUFFER$(\mathcal{S}, \mathcal{X}, H)$ uses $\mathcal{S}$ in two places. First, it uses $\mathcal{S}$ to determine $\partial H_{\downarrow X}$ (where X denotes the supernode selected from $\mathcal{X}$ to be processed during the execution of C), which is a subset of the vertices adjacent to H (and thus determined by calls above C). Second, it uses $\mathcal{S}$ to determine, for every vertex v in H, a closest vertex $\partial H_{\downarrow X}$ with respect to $\mathrm{dom}_{\mathcal{S}}(X)$. Notice that a shortest path P in $\mathrm{dom}_{\mathcal{S}}(X)$ from v to $\partial H_{\downarrow X}$ is contained in $H \cup \partial H_{\downarrow X}$: the first vertex along P that leaves H is in $\partial H_{\downarrow X}$, and thus is an endpoint of P. This means that P is determined by the graph $H \cup \partial H_{\downarrow X}$ (and as argued earlier, Invariant 3.5 implies that $\partial H_{\downarrow X}$ is determined by calls above C).

This shows that calls only affect each other if one is above the other; sibling calls do not affect each other. We will not explicitly use this fact in our proofs, instead depending solely on Invariant 3.5 — but it is nevertheless important intuition.

```
BUILDTREE(S, H):
  ⟨⟨Initialize supernode η⟩⟩
  S|_H ← supernodes in S seen by H
  v ← arbitrary vertex in H
  T_η ← SSSP tree in H, connecting v to a witness vertex for every supernode in S|_H
  initialize supernode η ← subgraph induced by vertices of T_η
  set T_η to be the skeleton of η, and add η to S ⟨⟨Currently, dom_S(η) = H⟩⟩
  initialize tree T with root η

  ⟨⟨Grow buffer and recurse⟩⟩
  for each connected component H' of H \ η:
      X ← list containing supernodes seen by H but not by H'
      GROWBUFFER(S, X, H')
  for each connected component H' of H \ ⋃S:
      T' ← BUILDTREE(S, H')
      attach tree T' as a child to the root of T
  return tree T
```

Figure 5. Pseudocode for procedure BuildTree$(\mathcal{S}, H)$

3.2 Analysis: Basic properties

Let $\mathcal{T}$ be the tree produced by BUILDTREE$(\emptyset, G)$. We will show that if G excludes a K_r-minor, then $\mathcal{T}$ is a $(\Delta, \Delta/r, r-1)$-buffered cop decomposition for G. In this section, we prove a collection of basic properties about $\mathcal{T}$, including the shortest-path skeleton and tree decomposition properties. The proofs of the supernode buffer and supernode radius properties are deferred to the next two sections.

Notation for supernodes changing over time. When we write "supernode η" without any subscript or description, we refer to the supernode in $\mathcal{T}$, at the end of the execution of the entire algorithm. In some proofs, we will need to refer to the global variable $\mathcal{S}$ at a specific point in the algorithm's execution. We adopt the following convention: If we say a call $C' :=$ BUILDTREE$(\mathcal{S}', H')$ is made during the algorithm, we use the variable $\mathcal{S}'$ to denote the global variable at the *start* of call C'. We use the notation "*supernode* $\eta_{\mathcal{S}'}$" to refer to the vertices of G that have already been assigned to η by $\mathcal{S}'$. It does *not* refer to those vertices that will be assigned to η in the future. The phrase "the set of supernodes *in* $\mathcal{S}'$" refers to the set of supernodes $\eta_{\mathcal{S}'}$ that are assigned by $\mathcal{S}'$.

GrowBuffer($\mathcal{S}, \mathcal{X}, H$):
  ⟨⟨*Note: All supernodes in $\mathcal{X}$ are not seen by H*⟩⟩
  If $\mathcal{X}$ is the empty list, do nothing and return.

  ⟨⟨*Grow buffer around a supernode in $\mathcal{X}$*⟩⟩
  $X \leftarrow$ arbitrary supernode in $\mathcal{X}$ ⟨⟨X *is processed during this call*⟩⟩
  $\partial H_{\downarrow X} \leftarrow$ set of vertices in $G \setminus H$ that are (1) adjacent to H, and (2) in $\mathrm{dom}_{\mathcal{S}}(X)$
  $\mathcal{N}H_X \leftarrow$ vertices v in H such that $\delta_{\mathrm{dom}_{\mathcal{S}}(X)}(v, \partial H_{\downarrow X}) \le \Delta/r$
  ⟨⟨*We show that* $\delta_{\mathrm{dom}_{\mathcal{S}}(X)}(v, \partial H_{\downarrow X}) < \delta_{\mathrm{dom}_{\mathcal{S}}(X)}(v, X)$ ⟩⟩
  for each v in $\mathcal{N}H_X$:
    $\eta_v \leftarrow$ supernode containing the vertex x in $\partial H_{\downarrow X}$ that minimizes $\delta_{\mathrm{dom}(X)}(v, x)$,
      breaking ties consistently
    assign v to supernode η_v, and update $\mathcal{S}$
    ⟨⟨*This changes η_v, and the domains of supernodes initialized below η_v*⟩⟩
    ⟨⟨*Since H does not change, $\partial H_{\downarrow X}$ and $\mathcal{N}H_X$ remain fixed*⟩⟩

  ⟨⟨*Growing a buffer may cut off more supernodes, so update $\mathcal{X}$*⟩⟩
  for each connected component H' of $H \setminus \bigcup \mathcal{S}$:
    $\mathcal{X}' \leftarrow \mathcal{X} \setminus \{X\}$
    add to $\mathcal{X}'$ all supernodes in $\mathcal{S}$ that are seen by H but not by H'
    GrowBuffer($\mathcal{S}, \mathcal{X}', H'$)

Figure 6. Pseudocode for procedure GrowBuffer($\mathcal{S}, \mathcal{X}, H$)

Terminology for GrowBuffer. Suppose that some call $C :=$ GrowBuffer($\mathcal{S}, \mathcal{X}, H$) occurs during the algorithm. This call begins by selecting an arbitrary supernode X from $\mathcal{X}$; we say that X is the supernode *processed during C*. The call C then defines the set $\mathcal{N}H_X$; we say that every point in $\mathcal{N}H_X$ is *assigned during C*.

Claim 3.6. *The following properties hold.*

1. *For every $\mathcal{S}$ that appears in the algorithm, every supernode $\eta_{\mathcal{S}}$ induces a connected subgraph of G.*
2. *Suppose that call C, whether it is* GrowBuffer($\mathcal{S}, \mathcal{X}, H$) *or* BuildTree($\mathcal{S}, H$)*, is made at some point in the algorithm. Over the course of the algorithm, every vertex in H is assigned either to a supernode initialized by C, or to a supernode initialized below C, or to a supernode in $\mathcal{S}$ that H sees (at the time C is called).*
3. *Supernode $\eta_{\mathcal{S}}$ will grow and $\mathrm{dom}_{\mathcal{S}}(\eta)$ will shrink over the course of the algorithm as $\mathcal{S}$ changes. Further, after the algorithm terminates, we have $\mathrm{dom}_{\mathcal{S}}(\eta) = \mathrm{dom}(\eta)$.*

Proof: **(1)** When supernode η is initialized, it is connected (because the skeleton T_η is connected). Whenever a vertex v is assigned to η by a call to GrowBuffer($\mathcal{S}, \mathcal{X}, H$), we claim that connectivity is preserved. Let X denote the supernode processed during GrowBuffer($\mathcal{S}, \mathcal{X}, H$), let $\partial H_{\downarrow X}$ denote the set of boundary vertices, and let $\mathcal{N}H_X$ denote the vertices assigned during GrowBuffer($\mathcal{S}, \mathcal{X}, H$). Let P be a shortest path in $\mathrm{dom}_{\mathcal{S}}(X)$ from v to the closest point $\partial H_{\downarrow X}$. Every vertex in P is closer to $\partial H_{\downarrow X}$ than v. Further, we claim that every vertex in P (excluding the endpoint, which is a boundary vertex) is in H. Indeed, every vertex in P is in $\mathrm{dom}_{\mathcal{S}}(X)$, so the first vertex along x that leaves H is in $\partial H_{\downarrow X}$, and thus is the endpoint of P. Thus, every point in P (excluding the endpoint) is in $\mathcal{N}H_X$. As every vertex in $\mathcal{N}H_X$ is assigned according to the closest vertex in $\partial H_{\downarrow X}$ (and ties are broken consistently), every vertex in path P is assigned to the same supernode η, and the connectivity of η is preserved.

(2) Let v be a vertex in H assigned to supernode η. Suppose that $\eta \subseteq H$ (and suppose that C itself did not initialize η). In this case, we claim that η was initialized below C. This follows from the fact that, for

any call to BUILDTREE, the children calls to BUILDTREE in the recursion tree operate on pairwise disjoint subgraphs. This implies that η is initialized either above or below C; as H consists of unassigned nodes at the time C is called, η is initialized below C.

Now suppose that η is not contained in H. As η is connected (Item 1), there is a path P in η between v and a vertex outside of H. By Invariant 3.4, the first vertex along P that leaves H has already been assigned in $\mathcal{S}$, at the time C is called. Thus, H sees $\eta_{\mathcal{S}}$.

(3) The fact that supernodes only grow over time is immediate from the algorithm. Let H denote the domain of η at the time η is initialized. By definition, $\mathrm{dom}_{\mathcal{S}}(\eta)$ is the set of vertices in H that are *now* assigned to supernodes above η; as supernodes only grow over time, $\mathrm{dom}_{\mathcal{S}}(\eta)$ only shrinks. It remains to show that the final $\mathrm{dom}_{\mathcal{S}}(\eta)$ is equal to $\mathrm{dom}(\eta)$. Indeed, by Item 2 (applied to the call to BUILDTREE that initialized η), every vertex in H is assigned to a supernode initialized either above or below η (or is assigned to η itself). A supernode is below η in the partition tree $\mathcal{T}$ if and only if it is initialized below η. Thus, after the algorithm terminates, $\mathrm{dom}_{\mathcal{S}}(\eta)$ is the set of vertices that are in η or in supernodes below η, which is precisely $\mathrm{dom}(\eta)$. □

Lemma 3.7. *Suppose that* BUILDTREE$(\mathcal{S}, H)$ *is called during the algorithm. Let* $\mathcal{S}|_H$ *be the set of supernodes in* $\mathcal{S}$ *seen by* H*. Then* $\mathcal{S}|_H$ *contains at most* $r-2$ *supernodes; furthermore, the supernodes in* $\mathcal{S}|_H$ *are pairwise adjacent.*

Proof: We first prove that the supernodes in $\mathcal{S}|_H$ are pairwise adjacent. Consider a pair (X, Y) of supernodes in $\mathcal{S}|_H$, and assume without loss of generality[8] that Y is initialized below X. Let x and y be the vertices chosen to be the roots of the skeletons of X and Y, respectively. Since H sees both $X_{\mathcal{S}}$ and $Y_{\mathcal{S}}$, and as $X_{\mathcal{S}}$ and $Y_{\mathcal{S}}$ are connected individually, there exists a path P from x to y containing only vertices in H, $X_{\mathcal{S}}$, and $Y_{\mathcal{S}}$.

Consider the time just before Y is initialized. Let $\tilde{\mathcal{S}}$ denote the assignments of vertices at that time, and let $\tilde{H}$ be the connected component of the subgraph induced by the unassigned vertices that contains y at that time. Observe that, although $X_{\tilde{\mathcal{S}}}$ may expand later, all vertices in P are either unassigned (and thus belong to $\tilde{H}$) or belong to $X_{\tilde{\mathcal{S}}}$. Hence, there is a path from y to $X_{\tilde{\mathcal{S}}}$ containing only unassigned vertices, meaning that $\tilde{H}$ sees $X_{\tilde{\mathcal{S}}}$. Thus, when $Y_{\tilde{\mathcal{S}}}$ is initialized, it must be adjacent to $X_{\tilde{\mathcal{S}}}$ by construction. By Claim 3.6(3), supernodes only grow, and thus X and Y must be adjacent.

By Claim 3.6(1), every supernode is connected. Thus, the above claim implies that $\mathcal{S}|_H \cup \{\eta\}$ forms a model for a K_{k+1}-minor, where $k = |\mathcal{S}|_H|$. As G excludes K_r-minors, we have $|\mathcal{S}|_H| \leq r-2$. □

Lemma 3.8 (Shortest-path skeleton property). *Every supernode* η *has a skeleton that is an SSSP tree in* $\mathrm{dom}(\eta)$ *with* $r-2$ *leaves.*

Proof: Notice that, throughout the course of the algorithm, $\mathrm{dom}_{\mathcal{S}}(\eta)$ may shrink but it never expands by Claim 3.6(3). As T_η is an SSSP tree in its original domain $\mathrm{dom}_{\mathcal{S}}(\eta)$, it is also an SSSP tree in $\mathrm{dom}(\eta)$. (We remark that η only grows over the course of the algorithm, so T_η is a subgraph of η, and thus is a subgraph of $\mathrm{dom}(\eta)$). By Lemma 3.7, tree T_η has at most $r-2$ leaves. □

Claim 3.9. *Let* C *be a call, whether to* BUILDTREE$(\mathcal{S}, H)$ *or to* GROWBUFFER$(\mathcal{S}, \mathcal{X}, H)$*, and let* η *be a supernode initialized above* C*. Let* H' *be a subgraph of* H*. If* H' *is adjacent to a vertex in* η *(after the algorithm terminates), then* H *sees* $\eta_{\mathcal{S}}$*.*

Proof: Let v be a vertex in $\eta \setminus H'$ adjacent to H'. As η is connected, there is a path P from v to T_η contained in $\{v\} \cup \eta$. As η is initialized above C (and, at the time C is called, subgraph H contains only

[8] By Invariant 3.5, both X and Y are initialized above BUILDTREE$(\mathcal{S}, H)$, so one of X or Y was initialized below the other.

unassigned vertices), the skeleton T_η is disjoint from H. Thus, P starts at a vertex in H and ends at a vertex outside of H. Consider the first vertex along P that leaves H. By Invariant 3.4, this vertex had already been assigned (to η) at the time C was called. We conclude that H sees $\eta_{\mathcal{S}}$. □

Lemma 3.10 (Tree decomposition property). *$\hat{\mathcal{T}}$ satisfies the tree decomposition property.*

Proof: First, note that Lemma 3.7 directly implies that each supernode sees at most $r-2$ ancestor supernodes, meaning that each bag in $\hat{\mathcal{T}}$ contains at most $r-1$ supernodes. Next, we prove that $\hat{\mathcal{T}}$ is a tree decomposition.

(1) The union of all vertices in all bags of $\hat{\mathcal{T}}$ is V by construction.

(2) Let (x,y) be an edge in G. We need to show that there is a bag in $\hat{\mathcal{T}}$ that contains both x and y. Let X be the supernode containing x, and let Y be the supernode containing y. We will prove that either $X = Y$ or one of them is an ancestor of the other (recall that, by definition, the bag of X contains all supernodes above X that are adjacent to X).

Assume that $X \neq Y$. We claim that X and Y are in an ancestor-descendent relationship in $\mathcal{T}$. Otherwise, consider the lowest common ancestor η of X and Y, initialized by a call $C \coloneqq \textsc{BuildTree}(\mathcal{S}, H)$. As X and Y are in different subtrees of η, vertices x and y are both unassigned and belong to different connected components of unassigned vertices, at the time when C begins to recursively make calls to BUILDTREE. But this is impossible, as there is an edge between x and y.

(3) We prove that for any supernode η, if there are two bags B_X and B_Y containing η, every bag in the path between them in $\hat{\mathcal{T}}$ contains η.

Let P be the path between B_X and B_Y in $\hat{\mathcal{T}}$. Assume that there exists some bag in P not containing η. Observe that the bag B_η is a common ancestor of both B_X and B_Y. Consider two paths: P_X from B_η to B_X and P_Y from B_η to B_Y. One of them, say P_X, must have a bag that does not contain η. Let $B_{\eta'}$ be the lowest bag in P_X such that $B_{\eta'}$ does not contain η, and let $B_{\eta''}$ be the child of $B_{\eta'}$ in P_X. Notice that $B_{\eta''}$ contains η. We remark that $B_{\eta'}$ is a descendent of B_η. From the construction of $\hat{\mathcal{T}}$, we get that supernode η'' is adjacent to η but supernode η' is not. Suppose that η' is initialized during the call $C' \coloneqq \textsc{BuildTree}(\mathcal{S}', H')$, and η'' is initialized during the call $C'' \coloneqq \textsc{BuildTree}(\mathcal{S}'', H'')$. As η is initialized above C', and $H'' \subseteq H'$, and H'' is adjacent to η, Claim 3.9 implies that H' sees $\eta_{\mathcal{S}'}$ at the time C' is called. Thus, by construction η' is adjacent to η, a contradiction. □

3.3 Analysis: Supernode buffer property

The following observation is almost immediate from the construction. It says that, if some subgraph H' is cut off from an old supernode X, there was some call to GROWBUFFER that processed X.

Observation 3.11. *Suppose that call $C' \coloneqq \textsc{BuildTree}(\mathcal{S}', H')$ is made during the algorithm. If X is a supernode initialized above C', and if H' does not see $X_{\mathcal{S}'}$ at the time C' is called, then there is some call $C \coloneqq \textsc{GrowBuffer}(\mathcal{S}, \mathcal{X}, H)$ such that (1) $H \supseteq H'$, and in particular C is above C', (2) H does not see $X_{\mathcal{S}}$, and (3) X was processed during C.*

To see why the observation holds, denote by $\tilde{C} \coloneqq \textsc{BuildTree}(\tilde{\mathcal{S}}, \tilde{H})$ the lowest call above C' such that $\tilde{H}$ sees $X_{\tilde{\mathcal{S}}}$ (or, if no such call exists, let $\tilde{C}$ be the call that initializes X). After making some calls to GROWBUFFER, the call $\tilde{C}$ must recurse on some subgraph that does not see X. Since the algorithm calls GROWBUFFER whenever a supernode gets cut off, there must be some (recursive) call to GROWBUFFER caused by $\tilde{C}$ that processed X, as claimed by the observation. We shall use this observation to prove the supernode buffer property.

Lemma 3.12 (Supernode buffer property). *Let X and η be supernodes, with η initialized below X. If η is not adjacent to X in G, then for every vertex v in* $\mathrm{dom}(\eta)$*, we have $\delta_{\mathrm{dom}(X)}(v,X) > \Delta/r$.*

Proof: We prove the following claim by induction (starting immediately below the call that initialized X, and working downward in the recursion tree):

> Let $C' \coloneqq \textsc{BuildTree}(\mathcal{S}', H')$ be a call that is below the call that initialized X. Either H' sees $X_{\mathcal{S}'}$, or $\delta_{\mathrm{dom}(X)}(v,X) > \Delta/r$ for every vertex v in H'.

We emphasize that the guarantee $\delta_{\mathrm{dom}(X)}(v,X) > \Delta/r$ refers to the *final* X, after all expansions are made. This suffices to prove the lemma. Indeed, the call to $\textsc{BuildTree}(\mathcal{S}', H')$ that initialized η comes below the call that initialized X, and $\mathrm{dom}(\eta) \subseteq H'$ by Claim 3.6(3); thus, either H' sees $X_{\mathcal{S}'}$ (in which case η is adjacent to X by definition of T_η), or every point v in $\mathrm{dom}(\eta)$ satisfies $\delta_{\mathrm{dom}(X)}(v,X) > \Delta/r$.

Inductive step. Suppose that H' does not see $X_{\mathcal{S}'}$. As we are in the inductive step, we may assume that the parent of C' in the recursion tree, $\tilde{C} \coloneqq \textsc{BuildTree}(\tilde{\mathcal{S}}, \tilde{H})$, is below the call that initialized X. If $\tilde{H}$ does not see $X_{\tilde{\mathcal{S}}}$, then we are done: since graph $\tilde{H}$ is a supergraph of H', the inductive hypothesis implies that $\delta_{\mathrm{dom}(X)}(v,X) > \Delta/r$ for every vertex v in H'.

The interesting case occurs when $\tilde{H}$ sees $X_{\tilde{\mathcal{S}}}$, but H' does not see $X_{\mathcal{S}'}$: that is, X becomes "cut off" from H' some time in between. In this case, by Observation 3.11 there is some call $C \coloneqq \textsc{GrowBuffer}(\mathcal{S}, \mathcal{X}, H)$ above C' that processes X, with $H \supseteq H'$ and H does not see $X_{\mathcal{S}}$. Consider any vertex v in H such that $\delta_{\mathrm{dom}(X)}(v,X) \le \Delta/r$ (where, again, we emphasize that X refers to the *final* X, after all expansions). If no such vertex exists, we are done.

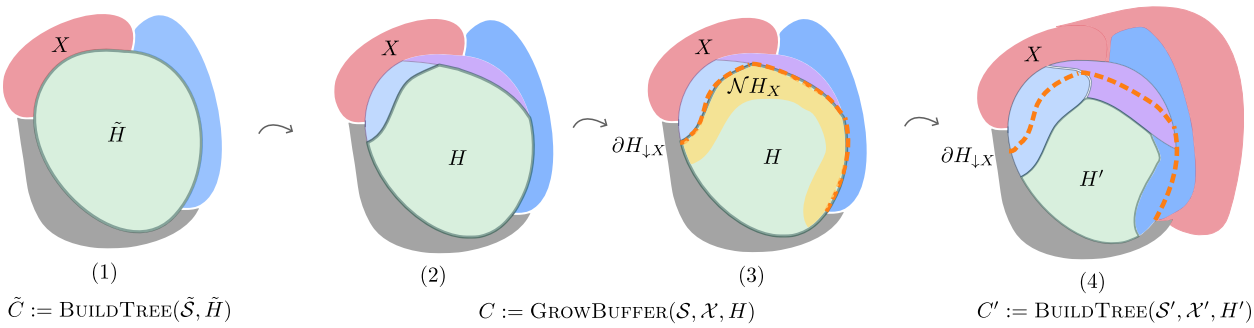

Figure 7. From left to right: (1) During call $\tilde{C}$, subgraph $\tilde{H}$ sees supernode X. The grey supernode is *above* X, and is not in $\mathrm{dom}_{\tilde{\mathcal{S}}}(X)$. (2–3) During call C, supernode X is cut off from H, and every point in $\mathcal{N}H_X$ (i.e. every point close to $\partial H_{\downarrow X}$) is assigned. (4) For every subgraph H' of H, every path in $\mathrm{dom}(X)$ from H' to X passes through $\partial H'_{\downarrow X}$.

We argue that *vertex v is at most Δ/r away from $\partial H_{\downarrow X}$ with respect to* $\mathrm{dom}_{\mathcal{S}}(X)$; see Figure 7. Let P be a shortest path from v to X in $\mathrm{dom}(X)$, where by assumption $\|P\| \le \Delta/r$. As the domain of X only shrinks over time (Claim 3.6(3)), path P is in $\mathrm{dom}_{\mathcal{S}}(X)$.[9] By Claim 3.6(2) on the call C, every vertex in H is assigned either to a supernode initialized below C, or to a supernode in $\mathcal{S}$ that H sees. Because X already existed in $\mathcal{S}$ (and thus X is not initialized below C) and H does not see $X_{\mathcal{S}}$, the other endpoint of P which is eventually assigned to X cannot be in H. So P passes through some vertex x outside of H that is adjacent to H. As P is contained in $\mathrm{dom}_{\mathcal{S}}(X)$, vertex x is in $\partial H_{\downarrow X}$. Thus, $\delta_{\mathrm{dom}_{\mathcal{S}}(X)}(v, \partial H_{\downarrow X}) \le \|P\| \le \Delta/r$.

This means that v is assigned to some supernode in $\mathcal{S}$ by the $\textsc{GrowBuffer}$ algorithm. Recall that call C' is below call C by Observation 3.11; thus, as calls are only made on connected components of

[9] However, notice that at the time when call C was made, X might not have grown into its final shape and $X_{\mathcal{S}}$ could be much smaller; in particular, P may not be a path from v to $X_{\mathcal{S}}$ and the distance from v to $X_{\mathcal{S}}$ can be larger than Δ/r.

unassigned vertices, we conclude that H' is a subgraph of H that includes only unassigned vertices. Thus, vertex v is not in H'. This completes the proof of the induction step.

Base case. In the base case, the parent of C' in the recursion tree is the call $\tilde{C} := \textsc{BuildTree}(\tilde{\mathcal{S}}, \tilde{H})$ that initialized X. If H' sees $X_{\mathcal{S}'}$, then we are done. If H' does not see $X_{\mathcal{S}'}$, then we are in the "interesting case" described above (except that here $\tilde{H}$ doesn't see $X_{\tilde{\mathcal{S}}}$): by Observation 3.11, there is some call $C := \textsc{GrowBuffer}(\mathcal{S}, \mathcal{X}, H)$ above C' and below $\tilde{C}$, during which X is processed. The argument for this case is identical to the one in the inductive case. □

3.4 Analysis: Supernode radius property

We now prove that every supernode η satisfies the radius property. To this end we prove three claims:

- Every time a supernode is cut off from a subgraph, the radius of η expands by at most Δ/r (Claim 3.13).
- There are at most $r-2$ supernodes that can cause η to expand (Claim 3.15).
- Each of the $r-2$ supernodes can cause η to expand at most once (Claim 3.14).

Combining these three claims in an inductive argument shows that the total expansion of η is bounded by Δ (Lemma 3.16).

Claim 3.13. *Suppose that v is assigned to a supernode η during a call $C := \textsc{GrowBuffer}(\mathcal{S}, \mathcal{X}, H)$. Let X denote the supernode processed during C, and let $\partial H_{\downarrow X}$ denote the boundary vertices. Let $\tilde{v}$ be the closest vertex in $\partial H_{\downarrow X}$ to v (with respect to $\mathrm{dom}_{\mathcal{S}}(X)$). Then $\delta_\eta(v, \tilde{v}) \le \Delta/r$ (with respect to the final η).*

Proof: Let $\mathcal{N}H_X$ denote the set of points assigned during C. Let P be a shortest path between v and $\tilde{v}$ in $\mathrm{dom}_{\mathcal{S}}(X)$. Every vertex in P (other than $\tilde{v}$) is in $\mathcal{N}H_X$. Because we assign every vertex in $\mathcal{N}H_X$ according to the closest vertex in $\partial H_{\downarrow X}$, every vertex in P is assigned to η. Further, P has length at most Δ/r, because every vertex in $\mathcal{N}H_X$ is within distance Δ/r of some vertex in $\partial H_{\downarrow X}$ (with respect to $\mathrm{dom}_{\mathcal{S}}(X)$), and $\tilde{v}$ is the closest vertex in $\partial H_{\downarrow X}$ to v. Thus, $\delta_\eta(v, \tilde{v}) \le \Delta/r$. □

We next show that each supernode seen by supernode η may cause η to be expanded at most once: if supernode $\tilde{X}$ causes η to expand because $\tilde{X}$ is cut off, supernode $\tilde{X}$ cannot be cut off *again* later on in the recursion. Later (in Claim 3.15) we will show that only supernodes seen by η may cause it to expand. Let $\tilde{X}$ be a supernode, and let H be a subgraph. We say that $\tilde{X}$ is *spent* with respect to H if there exists some call $\textsc{GrowBuffer}(\tilde{\mathcal{S}}, \tilde{\mathcal{X}}, \tilde{H})$ where $\tilde{H} \supseteq H$, and $\tilde{X}$ is processed during the call. In other words, $\tilde{X}$ is cut off from $\tilde{H}$ and H (even as $\tilde{X}$ grows), and it has already been "dealt with" during the previous call $\tilde{C}$.

Claim 3.14. *Suppose that call $\textsc{GrowBuffer}(\mathcal{S}, \mathcal{X}, H)$ is made during the algorithm. If supernode $\tilde{X}$ is spent with respect to H, then $\tilde{X}$ is not in $\mathcal{X}$.*

Proof: By definition of "spent", there is some call $\tilde{C} := \textsc{GrowBuffer}(\tilde{\mathcal{S}}, \tilde{\mathcal{X}}, \tilde{H})$ where $\tilde{H} \supseteq H$, and $\tilde{X}$ is processed during $\tilde{C}$. Notice that, because $\tilde{X}$ is in $\tilde{\mathcal{X}}$, subgraph $\tilde{H}$ does not see $\tilde{X}_{\tilde{\mathcal{S}}}$. Observe that:

- Call $\tilde{C}$ makes some calls to $\textsc{GrowBuffer}(\mathcal{S}', \mathcal{X}', H')$. For each of these calls made by $\tilde{C}$, notice that the set $\mathcal{X}'$ contains only supernodes in $\tilde{\mathcal{X}} \setminus \{\tilde{X}\}$ (the "leftover" ones from $\tilde{C}$) or supernodes in $\tilde{\mathcal{S}}$ that can be seen by $\tilde{H}$ but not by H' (those newly added ones). In particular, $\mathcal{X}'$ does not contain $\tilde{X}$. Further, *H' does not see $\tilde{X}_{\mathcal{S}'}$*. This follows from Claim 3.9: if H' could see $\tilde{X}_{\mathcal{S}'}$, then (because $\tilde{X}$ was initialized above $\tilde{C}$, and $H' \supseteq \tilde{H}$), Claim 3.9 would imply that $\tilde{H}$ could see $\tilde{X}_{\tilde{\mathcal{S}}}$, a contradiction.

  An inductive argument shows that, for every call to $\textsc{GrowBuffer}(\mathcal{S}', \mathcal{X}', H')$ made recursively as a result of $\tilde{C}$, the set $\mathcal{X}'$ does not contain $\tilde{X}$.

- After the recursion from $\tilde{C}$ terminates, the algorithm calls BUILDTREE on subgraphs of $\tilde{H}$, which may recursively result in more calls to BUILDTREE. Let BUILDTREE$(\mathcal{S}', H')$ be one of these calls, where $H' \subseteq \tilde{H}$. We claim that H' *does not see* $\tilde{X}_{\mathcal{S}'}$. As in the previous bullet point, this follows from Claim 3.9: if H' could see $\tilde{X}_{\mathcal{S}'}$, then Claim 3.9 would imply that $\tilde{H}$ could see $\tilde{X}_{\tilde{\mathcal{S}}}$, a contradiction.

  This means that whenever BUILDTREE$(\mathcal{S}', H')$ makes a call GROWBUFFER$(\mathcal{S}'', \mathcal{X}'', H'')$, the set $\mathcal{X}''$ does not include $\tilde{X}$; indeed, the set $\mathcal{X}''$ only includes supernodes that are seen by H'.

It follows from these two cases that, for every call to GROWBUFFER$(\mathcal{S}', \mathcal{X}', H')$ with $H' \subseteq \tilde{H}$, the supernode $\tilde{X}$ is not in $\mathcal{X}'$. In particular, the call GROWBUFFER$(\mathcal{S}, \mathcal{X}, H)$ satisfies $H \subseteq \tilde{H}$, and so $\tilde{X}$ is not in $\mathcal{X}$. □

The following claim, in conjunction with Lemma 3.7, implies that for any supernode η, at most $r-2$ supernodes can cause it to expand. We crucially rely on the fact that when supernode X is cut off, we only expand supernodes initialized *below* X; we do this because we only need to guarantee the supernode buffer property with respect to $\mathrm{dom}(X)$.

Claim 3.15. *Suppose that v is assigned to supernode η during a call $C :=$ GROWBUFFER$(\mathcal{S}, \mathcal{X}, H)$, and let X be the supernode in $\mathcal{X}$ processed during C. Suppose that η was initialized by a call $\hat{C} :=$ BUILDTREE$(\hat{\mathcal{S}}, \hat{H})$. Then $\hat{H}$ sees $X_{\hat{\mathcal{S}}}$.*

Proof: We first show that η is initialized below X. As v is assigned to supernode η during C, there is some vertex in $\partial H_{\downarrow X} \subseteq \mathrm{dom}_{\mathcal{S}}(X)$ that was assigned to η (in $\mathcal{S}$). Suppose that X was initialized during the call $\bar{C} :=$ BUILDTREE$(\bar{\mathcal{S}}, \bar{H})$, and notice that $\mathrm{dom}_{\mathcal{S}}(X) \subseteq \bar{H}$. Applying Claim 3.6(2) to call $\bar{C}$ shows that every vertex in $\bar{H}$ is (eventually) assigned to a supernode above X, or to X, or to a supernode below X. By definition, $\mathrm{dom}_{\mathcal{S}}(X)$ contains the vertices in $\bar{H}$ that are *not* assigned to supernodes above X (in $\mathcal{S}$). Thus, every vertex in $\mathrm{dom}_{\mathcal{S}}(X)$ is assigned to X or to a supernode below X, and so either $\eta = X$ or η is below X. It cannot be that $\eta = X$ (indeed, H sees η because $v \in \partial H_{\downarrow X}$, and H does not see X because X is in $\mathcal{X}$), so η is below X.

We also observe that C is below $\hat{C}$: because H is adjacent to a vertex in η, Invariant 3.5 implies that η was initialized above C. Thus, $\hat{H} \supseteq H$.

Now, for the sake of contradiction suppose that $\hat{H}$ does not see $X_{\hat{\mathcal{S}}}$. As η is below X and $\hat{H}$ does not see $X_{\hat{\mathcal{S}}}$, Observation 3.11 implies that there is some call $\tilde{C} :=$ GROWBUFFER$(\tilde{\mathcal{S}}, \tilde{\mathcal{X}}, \tilde{H})$ such that (1) $\tilde{H} \supseteq \hat{H}$, and (2) X is processed by $\tilde{C}$. As $\hat{H} \supseteq H$, this means that X is spent with respect to H, and Claim 3.14 implies that X is not in $\mathcal{X}$, a contradiction. □

Lemma 3.16. *Every supernode η has radius Δ with respect to skeleton T_η.*

Proof: Let BUILDTREE$(\hat{\mathcal{S}}, \hat{H})$ be the call that initialized η, and let $\hat{\mathcal{S}}|_{\hat{H}}$ denote the set of supernodes in $\hat{\mathcal{S}}$ that can be seen by $\hat{H}$. In other words, by Claim 3.15, $\hat{\mathcal{S}}|_{\hat{H}}$ is the set of supernodes that can cause η to expand. We prove the following statement by induction on k.

> Let v be a vertex assigned to η during a call $C :=$ GROWBUFFER$(\mathcal{S}, \mathcal{X}, H)$. If there are at most k supernodes in $\hat{\mathcal{S}}|_{\hat{H}}$ that are spent with respect to H, then $\delta_\eta(v, T_\eta) \le (k+1) \cdot \Delta/r$.

Let X be the supernode processed during the call C. Let $\tilde{v}$ be the closest vertex to v in $\eta_{\mathcal{S}}$ (with respect to $\mathrm{dom}_{\mathcal{S}}(X)$), as defined in Claim 3.13.

Inductive step ($k > 0$). If $\tilde{v}$ is in T_η, then Claim 3.13 implies that v is within distance Δ/r of T_η (in the final subgraph η), satisfying the claim. Otherwise, $\tilde{v}$ was assigned to η by some call $\tilde{C} :=$ GROWBUFFER$(\tilde{\mathcal{S}}, \tilde{\mathcal{X}}, \tilde{H})$.

We now show that the number of supernodes spent with respect to H is strictly greater than the number of supernodes spent with respect to $\tilde{H}$, aiming to apply the induction hypothesis on $\tilde{H}$.

- *Every supernode in $\hat{\mathcal{S}}|_{\hat{H}}$ that is spent with respect to $\tilde{H}$ is also spent with respect to H.* Indeed, for every such supernode $\bar{X}$ spent with respect to $\tilde{H}$, there is some call $\bar{C} := \textsc{GrowBuffer}(\bar{\mathcal{S}}, \bar{\mathcal{X}}, \bar{H})$ such that $\bar{X} \supseteq \tilde{X}$ and $\bar{C}$ processes $\bar{X}$; as $\tilde{H} \supseteq H$, call $\bar{C}$ also serves as a witness that $\bar{X}$ is spent with respect to H.

Now, consider the supernode $\tilde{X}$ that was processed during $\tilde{C}$. Observe that:

- *$\tilde{X}$ is not spent with respect to $\tilde{H}$.* This follows from Claim 3.14 and the fact that $\tilde{X} \in \tilde{\mathcal{X}}$.
- *$\tilde{X}$ is spent with respect to H.* First we argue that $\tilde{H} \supsetneq H$. Observe that call $\tilde{C}$ is above call C, because $\tilde{v}$ was already assigned when C is called. Thus, vertices that are unassigned when C is called are also unassigned when $\tilde{C}$ was called. In particular, when $\tilde{C}$ was called, every vertex in $H \cup \{\tilde{v}\}$ is unassigned. As $\tilde{H}$ is a maximal connected component of unassigned vertices and $\tilde{v}$ is adjacent to H by definition, $\tilde{H} \supsetneq H$.

  The existence of call $\tilde{C}$ (which processes $\tilde{X}$) together with the fact that $\tilde{H} \supseteq H$ implies that $\tilde{X}$ is spent with respect to H.

Moreover, Claim 3.15 implies that $\tilde{X}$ is in $\hat{\mathcal{S}}|_{\hat{H}}$, because a vertex was assigned to η during a call to GrowBuffer in which $\tilde{X}$ is processed. We conclude that there is at least one more supernode in $\hat{\mathcal{S}}|_{\hat{H}}$ that is spent with respect to H than those with respect to $\tilde{H}$. Thus, we can apply the inductive hypothesis to conclude that $\tilde{v}$ is within distance $k \cdot \Delta/r$ of T_η. By Claim 3.13, v is within distance Δ/r of $\tilde{v}$, and so v is within distance $(k+1) \cdot \Delta/r$ of T_η.

Base case ($k = 0$): In this case, we claim $\tilde{v}$ must be in T_η, and so $\delta_\eta(v, T_\eta) \le \Delta/r$. Suppose otherwise. Then $\tilde{v}$ is assigned by a call to GrowBuffer, and the argument above implies that there is at least one supernode in $\hat{\mathcal{S}}|_{\hat{H}}$ that is spent with respect to H. This contradicts our assumption that $k = 0$.

By Lemma 3.7, there are at most $r-2$ supernodes in $\hat{\mathcal{S}}|_{\hat{H}}$, and so we conclude that every vertex in η is within distance Δ of T_η. □

We conclude Theorem 3.2.

4 Shortcut partition from buffered cop decomposition

We first rephrase the definition of shortcut partition. Let G be a graph, let ε be a number in $(0,1)$, and let $\mathcal{C}$ be a partition of the vertices of G into clusters of strong diameter $\varepsilon \cdot \mathrm{diam}(G)$. Recall that the cluster graph $\check{G}$ is obtained from the original graph G by contracting each cluster in $\mathcal{C}$ into a single supernode. Let P be an arbitrary path in G. We define $\mathrm{cost}_{\mathcal{C}}(P)$ to be the minimum hop-length of a path $\check{P}$ in $\check{G}$, where (1) $\check{P}$ is a path between the clusters containing the endpoints of P, and (2) $\check{P}$ only contains clusters with nontrivial intersection with P. When $\mathcal{C}$ is clear from context, we omit the subscript and simply write $\mathrm{cost}(P)$. Notice that $\mathcal{C}$ is an (ε, h)-shortcut partition if, for every path P in G, we have $\mathrm{cost}(P) \le h \cdot \max\left\{\frac{\|P\|}{\mathrm{diam}(G)}, \varepsilon\right\}$; indeed, for any pair u, v of vertices in G, applying this condition for any *shortest* path P between u and v yields $\mathrm{cost}(P) \le h \cdot \max\left\{\frac{\|P\|}{\mathrm{diam}(G)}, \varepsilon\right\} = h \cdot \max\left\{\frac{\delta_G(u,v)}{\mathrm{diam}(G)}, \varepsilon\right\}$, as required.

In the rest of this section, we prove the following lemma:

Lemma 4.1. *Let G be a K_r-minor-free graph, and let Δ be a positive number. Then there is a partition $\mathcal{C}$ of G into connected clusters, such that (1) each cluster has strong diameter at most 4Δ, and (2) every path P in G with $\|P\| < \Delta/r$ has $\mathrm{cost}(P) \le r^{O(r)}$.*

We now show that Lemma 4.1 implies Theorem 1.2, which we restate below.

Theorem 1.2. *Any edge-weighted K_r-minor-free graph admits an $(\varepsilon, 2^{O(r\log r)}/\varepsilon)$-shortcut partition for any $\varepsilon \in (0,1)$.*

Proof: Let $\mathcal{C}$ be the partition guaranteed by Lemma 4.1 for $\Delta := \frac{\varepsilon\cdot\mathrm{diam}(G)}{4}$. Every cluster of $\mathcal{C}$ has strong diameter at most $4\Delta = \varepsilon\cdot\mathrm{diam}(G)$. To prove that $\mathcal{C}$ is an (ε,h)-shortcut partition with $h = r^{O(r)}/\varepsilon = 2^{O(r\log r)}/\varepsilon$, it suffices to show that $\mathrm{cost}(P) \le r^{O(r)}/\varepsilon \cdot \max\left\{\frac{\|P\|}{\mathrm{diam}(G)}, \varepsilon\right\}$, for an arbitrary path P in G.

We greedily partition P into a sequence of $O\left(\left\lceil \frac{r\|P\|}{\Delta}\right\rceil\right)$ vertex-disjoint subpaths, where each subpath has length at most Δ/r. That is, we can write P as the concatenation $P_1 \circ e_1 \circ P_2 \circ e_2 \ldots \circ Q_\tau$ for some $\tau = O\left(\left\lceil \frac{r\|P\|}{\Delta}\right\rceil\right)$, such that each P_i has length at most Δ/r. We can upper-bound the cost of P:

$$\mathrm{cost}(P) \le \sum_{1=1}^{\tau} \mathrm{cost}(P_i) + \sum_{i=1}^{\tau-1} \mathrm{cost}(e_i).$$

Each edge has cost at most 1, and (by Lemma 4.1) each subpath P_i has cost at most $r^{O(r)}$. It follows that $\mathrm{cost}(P) \le r^{O(r)} \cdot \left\lceil \frac{\|P\|}{\varepsilon\cdot\mathrm{diam}(G)}\right\rceil = r^{O(r)}/\varepsilon \cdot \max\left\{\frac{\|P\|}{\mathrm{diam}(G)}, \varepsilon\right\}$, which concludes the proof. □

4.1 Construction

Let $\mathcal{T}$ be a $(\Delta, \Delta/r, r-1)$-buffered cop decomposition for G. We partition each supernode η into clusters as follows. Fix an arbitrary supernode η.

Let N be a Δ-*net* of the skeleton T_η of η, which is an SSSP tree in $\mathrm{dom}(\eta)$; that is, N is a subset of vertices in T_η, such that (1) every vertex v in T_η satisfies $\delta_{T_\eta}(v,N) = \delta_{\mathrm{dom}(\eta)}(v,N) \le \Delta$, and (2) for every pair of vertices x_1 and x_2 in N, we have $\delta_{T_\eta}(x_1,x_2) = \delta_{\mathrm{dom}(\eta)}(x_1,x_2) > \Delta$. (The net N can be constructed greedily.) For each net point in N, we initialize a separate cluster.

We partition the rest of the vertices in η based on their closest point in the net N. In more detail, consider each vertex v in η in increasing order of their distance to N. Find the shortest path P_v from v to the closest point in N (if there are multiple such paths, we fix P_v arbitrarily). Let v' be the vertex adjacent to v in P_v. Set the cluster of v to be the same as the cluster of v'. Observe that each cluster has a single net point in N, which we refer to as the *center* of the cluster; the centers of clusters constitute N.

Lemma 4.2. *For each supernode, each of its clusters has strong diameter at most 4Δ.*

Proof: Let η be an arbitrary supernode and N be the set of cluster centers of η. First, we claim by induction on $\delta_\eta(v,N)$ that for every $v \in \eta$, the cluster C_v that contains v also contains a shortest path from v to N. For the basis, the claim clearly holds if $v \in N$. For the induction step, suppose that $v \notin N$. Let P_v be the shortest path from v to N that is fixed in our construction, and let v' be the vertex after v in P_v. Hence, v' is assigned to C_v. Since $\delta_\eta(v',N) < \delta_\eta(v,N)$, cluster C_v contains a shortest path from v' to N, denoted by Q'_v, by our induction hypothesis. Hence, the path $(v,v') \circ Q_{v'}$ is a shortest path from v to N, which is contained in C_v. This completes the induction step.

Consider an arbitrary cluster C in η and any vertex $v \in C$. By the supernode radius property, there is a vertex v_T in T_η such that $\delta_\eta(v,v_T) \le \Delta$; as N is a Δ-net, we have $\delta_\eta(v_T,N) \le \Delta$. By the triangle inequality, $\delta_\eta(v,N) \le 2\Delta$. By the above claim, C contains a shortest path from v to N, and is thus of length at most 2Δ. As C contains a single cluster center, the diameter of C must be at most 4Δ. □

We now bound the cost of a path P. We first prove that "the highest supernode η that P intersects" is well-defined, then show that P intersects few clusters in η, and finally give an inductive argument to bound $\mathrm{cost}(P)$.

Claim 4.3. *Let P be a path in G. P is contained in* $\mathrm{dom}(\eta)$ *for some supernode η that P intersects.*

Proof: Let $\hat{\mathcal{T}}$ denote the expansion of the partition tree $\mathcal{T}$. For every supernode η_1, the tree decomposition property implies that the set of bags in $\hat{\mathcal{T}}$ containing η_1 induces a connected subtree of $\hat{\mathcal{T}}$, which we denote $\hat{\mathcal{T}}[\eta_1]$. Further, for every pair of supernodes η_1 and η_2 that are adjacent in G, there is some bag shared by both $\hat{\mathcal{T}}[\eta_1]$ and $\hat{\mathcal{T}}[\eta_2]$; it follows that $\hat{\mathcal{T}}[\eta_1] \cup \hat{\mathcal{T}}[\eta_2]$ is a connected subtree of $\hat{\mathcal{T}}$. As P is connected, the set of bags containing *any* supernode that P intersects induces a connected subtree of $\hat{\mathcal{T}}$, which we denote $\hat{\mathcal{T}}[P]$.

Let B_η denote the root bag of the subtree $\hat{\mathcal{T}}[P]$, where B_η is in one-to-one correspondence with the supernode η in $\mathcal{T}$. Bag B_η contains the supernode η, as well as some supernodes above η in $\mathcal{T}$. Path P does not intersect any supernode above η, as otherwise $\hat{\mathcal{T}}[P]$ would include a bag above B_η. Thus, P is in $\mathrm{dom}(\eta)$. Further, P intersects some supernode in B_η, so P intersects η. □

Claim 4.4. *If P is a path of length less than Δ, and η is a supernode such that P is contained in* $\mathrm{dom}(\eta)$*, then P intersects at most $9r$ clusters in η.*

Proof: Suppose for contradiction that P satisfies the conditions of the claim, yet it intersects at least $9r+1$ clusters in η. By the shortest-path skeleton property, the skeleton T_η of η is an SSSP tree in $\mathrm{dom}(\eta)$ with at most $r-2$ leaves, thus the vertices of T_η can be partitioned into at most $r-2 \le r$ shortest paths. Since each cluster in η has its center chosen from one of at most r shortest paths, P intersects at least 10 clusters with centers in the same shortest path, denoted by Q. Let C_u (respectively, C_v) be the first (resp., last) cluster (among those with centers in Q) that P intersects, let u (resp., v) be an intersection point of P and C_u (resp., C_v), and let c_u (resp., c_v) be the center of C_u (resp., C_v). Since Q is a shortest path in $\mathrm{dom}(\eta)$ that intersects at least 10 centers and as the distance between any two cluster centers is at least Δ, we have $\delta_{\mathrm{dom}(\eta)}(c_u, c_v) \ge 9\Delta$. By the triangle inequality, we have:

$$||P|| \;\ge\; \delta_{\mathrm{dom}(\eta)}(u,v) \;\ge\; \delta_{\mathrm{dom}(\eta)}(c_u,c_v) - \underbrace{(\delta_{\mathrm{dom}(\eta)}(c_u,u) + \delta_{\mathrm{dom}(\eta)}(c_v,v))}_{\le 4\Delta+4\Delta \quad \text{by Lemma 4.2}} \;\ge\; 9\Delta - 8\Delta \;\ge\; \Delta,$$

yielding a contradiction. □

We say that a path P is *k-constrained* if, for every supernode η that P intersects, there are at most k supernodes that P intersects that are contained in the bag B_η corresponding to η.

Lemma 4.5. *Let P be a k-constrained path with $\|P\| < \Delta/r$. Then* $\mathrm{cost}(P) \le (54r)^k$.

Proof: The proof is by induction on k. The basis is trivially true, as only the empty path is 0-constrained. We next prove the induction step. By Claim 4.3, there is some supernode η that P intersects, such that P is in $\mathrm{dom}(\eta)$. Choose an arbitrary vertex $v_\eta \in P \cap \eta$ and split P at v_η into two subpaths, P' and P''; v_η is an endpoint of both P' and P''. We will show $\mathrm{cost}(P') \le 27r \cdot (54r)^{k-1} = \frac{1}{2}(54r)^k$, so by symmetry, $\mathrm{cost}(P) = \mathrm{cost}(P') + \mathrm{cost}(P'') \le (54r)^k$.

We partition P' into a sequence of vertex-disjoint subpaths $P_1, P_{1:2}, P_2, P_{2:3}, \ldots$ as follows. Let $\mathcal{C}[\eta]$ denote the set of clusters in η that P' intersects. Define C_1 to be the cluster (in $\mathcal{C}[\eta]$) that contains v_η, define P_1 to be the maximal prefix of P' that ends in a vertex in C_1, and define $P[C_1 :] \coloneqq P' \setminus P_1$ to be the suffix of P' starting after P_1. For all $i \ge 1$ with $C_i \ne \emptyset$, we recursively define (see Figure 8):

- Define C_{i+1} to be the first cluster in $\mathcal{C}[\eta]$ that $P[C_i :]$ intersects. If $P[C_i :]$ intersects no clusters in $\mathcal{C}[\eta]$, then define $C_{i+1} \coloneqq \emptyset$.

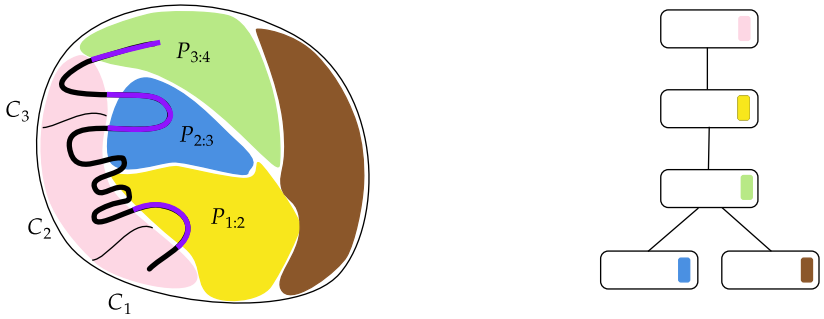

Figure 8. Path P' with subpaths $P_{1:2}$, $P_{2:3}$, and $P_{3:4}$ highlighted in purple, where η is the pink supernode.

- Define $P_{i:i+1}$ to be the maximal prefix of $P[C_i :]$ that contains no vertex in C_{i+1}. If $C_{i+1} = \emptyset$, then set $P_{i:i+1} := P[C_i :]$. Notice that $P_{i:i+1}$ may be the empty path if the first vertex on $P[C_i :]$ is in C_{i+1}.
- Define P_{i+1} to be the maximal subpath of $P[C_i :]$ with both endpoints in C_{i+1}. Notice that P_{i+1} starts immediately after $P_{i:i+1}$.
- Define $P[C_{i+1} :]$ to be the suffix of $P[C_i :]$ that starts after P_{i+1}. Notice that $P[C_{i+1} :]$ contains no vertices in C_{i+1}.

By Claim 4.4, $\mathcal{C}[\eta]$ contains at most $9r$ clusters[10]; thus, there are at most $9r$ subpaths P_i and $9r$ subpaths $P_{i:i+1}$ defined by the above procedure. There are at most $18r$ edges in P' that connect the subpaths. The cost of P' is bounded by the sum of costs of the subpaths as well as the edges between the subpaths.

- Each edge has cost at most 1.
- Each subpath P_i has cost 0, as either P_i is empty or its endpoints are in the same cluster.
- As we argue next, *each subpath $P_{i:i+1}$ has cost at most* $(54r)^{k-1}$.

  Observe that every supernode η' that P' intersects has η in its bag $B_{\eta'}$. Indeed, if $B_{\eta'}$ did not contain η, then η' and η would not be adjacent by definition; as η is above η' in $\mathcal{T}$ (by definition of η), the supernode buffer property implies that $\|P\| \geq \delta_{\mathrm{dom}(\eta)}(\eta', \eta) \geq \Delta/r$, a contradiction. Further, notice that $P_{i:i+1}$ does not intersect η. Thus, as P' is k-constrained, each subpath $P_{i:i+1}$ is $(k-1)$-constrained. The inductive hypothesis implies that $\mathrm{cost}(P_{i:i+1}) \leq (54r)^{k-1}$, as argued.

Since $k \geq 1$, we conclude that

$$\mathrm{cost}(P') \;\leq\; 18r \cdot 1 + 9r \cdot 0 + 9r \cdot (54r)^{k-1} \;\leq\; 27r \cdot (54r)^{k-1}.$$

This proves the lemma. □

Noting that every path is $(r-1)$-constrained (as every bag contains at most $r-1$ supernodes), Lemma 4.2 and Lemma 4.5 prove Theorem 1.2.

5 Other Applications

In this section, we describe several applications of our shortcut partition mentioned in the introduction. We will start with two direct applications (Section 5.1) and then proceed to the application on the embedding of apex-minor-free graphs (Section 5.2).

[10] Claim 4.4 is stronger than what we need here

5.1 Direct Applications

Tree cover. The authors of [CCL+23] provided a construction of $(1+\varepsilon)$-tree cover for minor-free graphs via shortcut partition. Their definition of (ε, h)-shortcut partition is slightly different than our Definition 1.1; it is strictly weaker. Recall that the low-hop property of Definition 1.1 guarantees:

> For any pair of vertices u and v in G, there exists some shortest path π in G between u and v, and a path $\check{\pi}$ in the cluster graph $\check{G}$ such that (in addition to other properties) $\check{\pi}$ only contains clusters that intersect π.

The definition of [CCL+23] (Definition 2.1) instead only guarantees:

> For any pair of vertices u and v in G, there exists some path π' in G between u and v with $\|\pi'\| \leq (1+\varepsilon)\delta_G(u,v)$, and a path $\check{\pi}$ in the cluster graph $\check{G}$ such that (in addition to other properties) $\check{\pi}$ only contains clusters that intersect π'.

This is a weaker guarantee. Also, as mentioned already, the definition of [CCL+23] states that the hop-length of $\check{\pi}$ is at most h, regardless of $\delta_G(u,v)$, while our definition takes $\delta_G(u,v)$ into account, allowing smaller hop-lengths for smaller distances. Consequently, the shortcut partition we construct in Theorem 1.2 can be used in their construction of tree cover.

The tree cover construction of [CCL+23] consists of two steps.[11] The first step is a reduction from a tree cover with multiplicative distortion $(1+\varepsilon)$ to a tree cover with additive distortion $+\varepsilon\Delta$, where Δ is the diameter, with a loss of a $O(\log(1/\varepsilon))$ factor to the cover size. In the second step, it is shown that an (ε, h)-shortcut partition for minor-free graphs implies a tree cover of size $2^{O(h)}$ with additive distortion $+\varepsilon\Delta$. Their result can be summarized as follows.

Lemma 5.1 (Lemma 1.7 and Theorem 1.8 in [CCL+23]). *Let G be a minor-free graph, and $\varepsilon \in (0,1)$. If every subgraph of G admits an (ε, h)-shortcut partition, then G has a $(1+\varepsilon)$-tree cover of size $2^{O(h)}$.*

By Theorem 1.2, any K_r-minor-free graph has an $(\varepsilon, r^{O(r)}/\varepsilon)$-shortcut partition. This proves Theorem 1.8.

Distance oracle. As discussed in Section 1.2, we construct our distance oracle from our tree cover (Theorem 1.8): given a query pair (u,v), query the distance $d_T(u,v)$ in T, for each tree T in the tree cover $\mathcal{T}$, and return $\min_{T\in\mathcal{T}} d_T(u,v)$. Distance query in a tree is reduced to a lowest common ancestor (LCA) query. In the RAM model, there are LCA data structures [HT84, BFC00] with $O(n)$ space and $O(1)$ query time. In the pointer machine model, this can be carried out with $O(n)$ space and $O(\log\log n)$ query time [vL76]. Theorem 1.7 now follows.

5.2 Embedding of apex-minor-free graphs

The authors of [CCL+23] show that any *planar* graph G with diameter Δ can be embedded into a graph of treewidth $O(\varepsilon^{-4})$ with distortion $+\varepsilon\Delta$. Their argument uses three properties of planar graphs, which carries over to any minor-free graph with these properties (Lemma 5.5). Loosely speaking, they show that if (P1) G has an (ε, h)-shortcut partition, (P2) G has an $+\varepsilon\Delta$ forest cover $\mathcal{F}$ for G that "interacts nicely" with the shortcut partition (Lemma 5.4), and (P3) G has the *local-treewidth* property, then G can be embedded into a graph of treewidth $O(h\cdot|\mathcal{F}|)$ with distortion $+\varepsilon\Delta$.

Theorem 1.2 gives us a shortcut partition for minor-free graphs and hence (P1). Apex-minor-free graphs have the local treewidth property [Epp00, DH04a, DH04b] (see Lemma 5.2 below) and hence

[11] The more efficient tree cover construction for *planar graphs* in [CCL+23] does not follow the two-step framework; instead, the authors exploited planarity to get a better (and indeed polynomial) dependency on ε in the size of the cover.

satisfy (P3). The main goal of this section is to show (P2) (by proving Lemma 5.4), and we do so by applying the framework of [CCL+23] to our shortcut partition to construct an appropriate forest cover.

Lemma 5.2 (Diameter-treewidth property [DH04b]). *Let G be a graph excluding a fixed apex graph as a minor. Let D be its (unweighted) diameter. Then* $\mathrm{tw}(G) = O(D)$.

We note that the big-O in Lemma 5.2 hides the dependency on the size of the minor; it is the Robertson-Seymour constant. As a result, the dependency on the minor of our Theorem 1.9 also has a Robertson-Seymour constant.

We recall the construction of the [CCL+23] tree cover for completeness. A forest F is *dominating* if $d_F(u,v) \geq d_G(u,v)$ for every two vertices $u, v \in V(G)$. In the definition of the tree cover for G, we allow *Steiner* vertices, i.e., which do not belong to G, in a tree. Here the forest we construct will contain no Steiner vertices, meaning that $V(F) \subseteq V(G)$. In this case, we say that F is *Steiner-free*. Let $\mathcal{C}$ be a clustering of G. Let $\check{G}$ be the *cluster graph* obtained from G by contracting clusters in $\mathcal{C}$; $\check{G}$ is a simple unweighted graph. Let $\check{F}$ be a forest, subgraph of $\check{G}$ such that every tree in $\check{F}$ is rooted at some node. We define the *star expansion* of $\check{F}$ to be a forest F of G obtained by applying the following to each tree $\check{T} \in \check{F}$ (see Figure 9):

> Let V_T denote the set of vertices (of G) that belong to clusters in $\check{T}$. Choose an arbitrary vertex r in the cluster that is the root of $\check{T}$. Let T be a star rooted at r connected to every vertex in V_T. We assign each edge (r,u) of T a weight $d_G(r,u)$. We then add T to F.

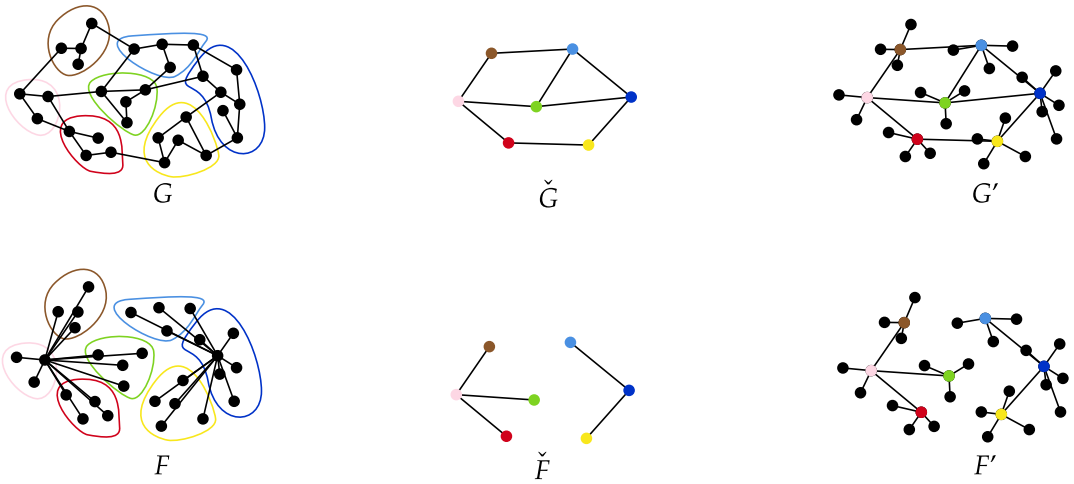

Figure 9. An illustration of G, $\check{G}$, G', F, $\check{F}$, and F'.

Let $\check{\mathcal{F}}$ be a collection of rooted forests of $\check{G}$, each of which is a subgraph of $\check{G}$, called a *spanning forest cover*. The *star expansion* of $\check{\mathcal{F}}$ is a collection of rooted forests, denoted by $\mathcal{F}$, obtained by taking star expansion of each rooted forest $\check{F}$ in $\check{\mathcal{F}}$. We note that a forest in $\check{\mathcal{F}}$ might not be a subgraph of G, but it is Steiner-free. (That is, each forest might contain Steiner edges—edges not in G—but not Steiner vertices.) The following lemma was proven in [CCL+23].

Lemma 5.3 (Adapted from Theorem 2.2 and Theorem 2.5 in [CCL+23]). *Let G be an edge-weighted minor-free graph with diameter Δ, and let $\varepsilon \in (0,1)$. Suppose that G has an $(\varepsilon, h(\varepsilon))$-shortcut partition $\mathcal{C}$. Let $\check{G}$ be the cluster graph obtained from G by contracting clusters in $\mathcal{C}$. Then there exists a spanning forest cover $\check{\mathcal{F}}$ of $\check{G}$ such that its star expansion $\mathcal{F}$ satisfies the following properties:*

- [Root preservation.] *For every pair of vertices u and v in G, there is a tree T in some forest of $\mathcal{F}$ such that (1) $\delta_T(u,v) \le \delta_G(u,v) + \varepsilon\Delta$, and (2) a shortest path from u to v in T passes through the root of T.*
- [Size.] *$\mathcal{F}$ contains $2^{O(h(\varepsilon))}$ forests.*

We remark that the second half of the root preservation property was not explicitly stated in [CCL+23]; however, it is immediate from the fact that $\mathcal{F}$ is a set of star forests.

We now show the following structural result for apex-minor-free graphs (see Figure 9 for illustration).

Lemma 5.4. *Let G be an edge-weighted graph with diameter Δ excluding a fixed apex graph as a minor, and let $\varepsilon \in (0,1)$. Then there is a partition $\mathcal{C}$ of the vertices of G into clusters with strong diameter $\varepsilon\Delta$, and a set $\mathcal{F}$ of $2^{O(1/\varepsilon)}$ forests with the same vertex set as G, that satisfy the following properties:*

- [Low-hop.] *For every pair of vertices u and v, there is a path between u and v that intersects at most h clusters, for some $h = O(1/\varepsilon)$.*
- [Root preservation.] *For every pair of vertices u and v in G, there is a tree T in some forest of $\mathcal{F}$ such that (1) $\delta_T(u,v) \le \delta_G(u,v) + \varepsilon\Delta$, and (2) a shortest path from u to v in T passes through the root of T.*

For each cluster C in $\mathcal{C}$, choose an arbitrary vertex v_C to be the center vertex, and define the star S_C to be a star connecting v_C to every other point in C. Define G' to be the graph obtained by replacing every supernode C in cluster graph $\breve{G}$ with the star S_C: every edge between two clusters in $\breve{G}$ is replaced with an edge in G' between the centers of the two clusters. Notice G' has the same vertex set as G.

- [Contracted treewidth.] *Graph G' has treewidth $O(h)$.*
- [Forest correspondence.] *For every forest F in $\mathcal{F}$, there is a corresponding spanning forest F' (a subgraph of G') such that: For every tree T in F, there is a tree T' in F' such that $V(T) = V(T')$ and $\mathrm{root}(T) = \mathrm{root}(T')$.*

Proof: By Theorem 1.2, there is a partition $\mathcal{C}$ of G that is an (ε, h)-shortcut partition, for $h = O(1/\varepsilon)$. By Lemma 5.3, we can construct a spanning forest cover $\breve{\mathcal{F}}$ of $\breve{G}$ and its star expansion $\mathcal{F}$ satisfying the root preservation property. Furthermore, $\mathcal{F}$ has $2^{O(1/\varepsilon)}$ forests. We now show the other three properties of the theorem.

[Low-hop.] The low-hop property follows directly from the statement of Theorem 1.2.

[Contracted treewidth.] Let $\breve{G}$ denote the graph obtained by contracting every cluster in $\mathcal{C}$ into a supernode. By the low-hop property, the (unweighted) diameter of $\breve{G}$ is at most h. Graph $\breve{G}$ excludes the same minors as G, so Lemma 5.2 implies that $\breve{G}$ has treewidth $O(h)$. Now notice that we can obtain G' from $\breve{G}$ by creating new (degree-1) vertices and adding an edge between each new vertex and a supernode in $\breve{G}$. We construct a tree decomposition for G', starting from the tree decomposition for $\breve{G}$: For each new vertex v attached to an existing supernode u, create a bag containing u and v, and add it as a child to an arbitrary bag containing u in the tree decomposition of $\breve{G}$. This procedure does not change the treewidth; it is still $O(h)$.

[Forest correspondence.] By definition, each forest $F \in \mathcal{F}$ is a star expansion of a forest $\breve{F} \in \breve{\mathcal{F}}$. To get the forest correspondence property, we simply transform $\breve{F}$ into a forest F' on G' in the natural way: For each tree $\breve{T}$ in $\breve{F}$, replace every vertex C in $\breve{T}$ with the corresponding star S_C in G', and replace every edge in $\breve{T}$ with the corresponding edge between star centers in G'. We claim that F' is a forest. Indeed,

this transformation maps each tree $\check{T}$ to a tree T' in G', because T' is connected and has one more vertex than it has edges. Recall that for each tree $\check{T} \in \check{F}$, there is a corresponding tree $T \in F$, which is obtained by star expansion. By construction, T and T' has the same vertex set. We then can set the root of T' to be the same as the root of T.

Further, the trees T' are vertex disjoint because the clusters $\mathcal{C}$ are vertex disjoint, and there are no edges between the trees T'. Thus, F' is a spanning forest of G'. □

The following reduction is implicit in [CCL+23].[12]

Lemma 5.5 ([CCL+23], Section 7). *Let G be an edge-weighted minor-free graph with diameter Δ, and let ε be a number in $(0,1)$. Suppose there is a partition $\mathcal{C}$ of G into clusters of strong diameter $\varepsilon\Delta$, together with a set of forests $\mathcal{F}$, such that $\mathcal{C}$ and $\mathcal{F}$ satisfy the low-hop property with parameter h, the root preservation property, the contracted treewidth property, and the forest correspondence property. Then G can be embedded deterministically into a graph with treewidth $O(h \cdot |\mathcal{F}|)$ and distortion $+O(\varepsilon\Delta)$.*

Combining Lemma 5.4 and Lemma 5.5 proves Theorem 1.9.

Acknowledgement. Hung Le and Cuong Than are supported by the NSF CAREER Award No. CCF-2237288 and an NSF Grant No. CCF-2121952. Shay Solomon is funded by the European Union (ERC, DynOpt, 101043159). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them. Shay Solomon is also supported by the Israel Science Foundation (ISF) grant No.1991/1. Shay Solomon and Lazar Milenković are supported by a grant from the United States-Israel Binational Science Foundation (BSF), Jerusalem, Israel, and the United States National Science Foundation (NSF).

References

[ADM+95] Sunil Arya, Gautam Das, David M. Mount, Jeffrey S. Salowe, and Michiel Smid. Euclidean spanners. In *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing, STOC '95*, 1995. doi:10.1145/225058.225191.

[AG06] Ittai Abraham and Cyril Gavoille. Object location using path separators. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Principles of Distributed Computing*, PODC '06, pages 188–197, 2006. doi:10.1145/1146381.1146411.

[AGG+14] Ittai Abraham, Cyril Gavoille, Anupam Gupta, Ofer Neiman, and Kunal Talwar. Cops, robbers, and threatening skeletons: Padded decomposition for minor-free graphs. In *Proceedings of the forty-sixth annual ACM symposium on Theory of Computing*, pages 79–88, 2014. doi:10.1145/2591796.2591849.

[AGGM06] Ittai Abraham, Cyril Gavoille, Andrew V. Goldberg, and Dahlia Malkhi. Routing in networks with low doubling dimension. In *Proc. of 26th ICDCS*, page 75, 2006.

[AKP94] Baruch Awerbuch, Shay Kutten, and David Peleg. On buffer-economical store-and-forward deadlock prevention. *IEEE transactions on communications*, 42(11):2934–2937, 1994.

[And86] Thomas Andreae. On a pursuit game played on graphs for which a minor is excluded. *Journal of Combinatorial Theory, Series B*, 41(1):37–47, 1986.

[12] There are small differences in the phrasing of [CCL+23]. (1) They do not explicitly state the *contracted treewidth* property; rather, they prove it as a lemma using properties of planar graphs. (2) They do not explicitly state the *forest correspondence* property; rather, they state a different condition (the "disjoint cluster" condition), that they then use to prove the forest correspondence property in a lemma. The "disjoint cluster" condition is used only to prove the forest correspondence property.

[AP90] Baruch Awerbuch and David Peleg. Sparse partitions. In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science*, FOCS '90, pages 503–513, 1990. doi:10.1109/fscs.1990.89571.

[AP92] Baruch Awerbuch and David Peleg. Routing with polynomial communication-space trade-off. *SIAM J. Discret. Math.*, 5(2):151–162, may 1992. doi:10.1137/0405013.

[Bar96] Yair Bartal. Probabilistic approximations of metric spaces and its algorithmic applications. In *37th Annual Symposium on Foundations of Computer Science, FOCS '96, Burlington, Vermont, USA, 14-16 October, 1996*, pages 184–193, 1996. URL: https://doi.org/10.1109/SFCS.1996.548477.

[BFC00] Michael A. Bender and Martin Farach-Colton. The LCA problem revisited. In *Latin American Symposium on Theoretical Informatics (LATIN '00)*, pages 88–94, 2000. doi:10.1007/10719839_9.

[BFN19] Yair Bartal, Nova Fandina, and Ofer Neiman. Covering metric spaces by few trees. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, pages 20:1–20:16, 2019. doi:10.4230/LIPIcs.ICALP.2019.20.

[BG08] Amitabh Basu and Anupam Gupta. Steiner point removal in graph metrics. Unpublished manuscript, available from https://www.ams.jhu.edu/~abasu9/papers/SPR.pdf, 2008.

[BLT14] Costas Busch, Ryan LaFortune, and Srikanta Tirthapura. Sparse covers for planar graphs and graphs that exclude a fixed minor. *Algorithmica*, 69(3):658–684, July 2014. doi:10.1007/s00453-013-9757-4.

[CCL+23] Hsien-Chih Chang, Jonathan Conroy, Hung Le, Lazar Milenković, Shay Solomon, and Cuong Than. Covering planar metrics (and beyond): O(1) trees suffice, 2023. Accepted to FOCS 2023.

[CFKL20] Vincent Cohen-Addad, Arnold Filtser, Philip N. Klein, and Hung Le. On light spanners, low-treewidth embeddings and efficient traversing in minor-free graphs. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science, FOCS '22*, 2020. doi:10.1109/focs46700.2020.00061.

[CGH16] Yun Kuen Cheung, Gramoz Goranci, and Monika Henzinger. Graph minors for preserving terminal distances approximately – lower and upper bounds. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*, Leibniz International Proceedings in Informatics (LIPIcs), pages 131:1–131:14, 2016. doi:10.4230/LIPIcs.ICALP.2016.131.

[Che18] Yun Kuen Cheung. Steiner point removal — distant terminals don't (really) bother. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms, (SODA '18)*, pages 1353–1360. 2018. doi:10.1137/1.9781611975031.89.

[CKT22] Hsien-Chih Chang, Robert Krauthgamer, and Zihan Tan. Almost-linear ε-emulators for planar graphs. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1311–1324, Rome Italy, June 2022. doi:10.1145/3519935.3519998.

[CS19] Timothy M. Chan and Dimitrios Skrepetos. Faster approximate diameter and distance oracles in planar graphs. *Algorithmica*, 81(8):3075–3098, 2019. doi:10.1007/s00453-019-00570-z.

[CXKR06] T.-H. Hubert Chan, Donglin Xia, Goran Konjevod, and Andrea Richa. A tight lower bound for the Steiner point removal problem on trees. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2006)*, pages 70–81. 2006. doi:10.1007/11830924_9.

[DH04a] Erik D. Demaine and Mohammad Taghi Hajiaghayi. Diameter and treewidth in minor-closed graph families, revisited. *Algorithmica*, 40(3):211–215, 2004. doi:10.1007/s00453-004-1106-1.

[DH04b] Erik D. Demaine and MohammadTaghi Hajiaghayi. Equivalence of local treewidth and linear local treewidth and its algorithmic applications. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '04, page 840–849, 2004.

[EGK+14] Matthias Englert, Anupam Gupta, Robert Krauthgamer, Harald Räcke, Inbal Talgam-Cohen, and Kunal Talwar. Vertex sparsifiers: New results from old techniques. *SIAM Journal on Computing*, 43(4):1239–1262, 2014. doi:10.1137/130908440.

[Epp00] David Eppstein. Diameter and treewidth in minor-closed graph families. *Algorithmica*, 27(3):275–291, 2000. doi:10.1007/s004530010020.

[Fil19] Arnold Filtser. Steiner point removal with distortion $O(\log k)$ using the relaxed-Voronoi algorithm. *SIAM Journal on Computing*, 48(2):249–278, 2019. doi:10.1137/18m1184400.

[Fil20a] Arnold Filtser. A face cover perspective to ℓ_1 embeddings of planar graphs. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1945–1954. 2020. doi:10.1137/1.9781611975994.120.

[Fil20b] Arnold Filtser. Scattering and sparse partitions, and their applications. In *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, volume 168 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 47:1–47:20, 2020. doi:10.4230/LIPIcs.ICALP.2020.47.

[FKS19] Eli Fox-Epstein, Philip N. Klein, and Aaron Schild. Embedding planar graphs into low-treewidth graphs with applications to efficient approximation schemes for metric problems. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1069–1088. 2019. doi:10.1137/1.9781611975482.66.

[FKT18] Arnold Filtser, Robert Krauthgamer, and Ohad Trabelsi. Relaxed Voronoi: A simple framework for terminal-clustering problems. In *2nd Symposium on Simplicity in Algorithms (SOSA 2019)*, pages 10:1–10:14, 2018. doi:10.4230/OASIcs.SOSA.2019.10.

[FL22] Arnold Filtser and Hung Le. Low treewidth embeddings of planar and minor-free metrics. In *IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS '22)*, 2022. doi:10.1109/focs54457.2022.00105.

[GKR01] Anupam Gupta, Amit Kumar, and Rajeev Rastogi. Traveling with a Pez dispenser (or, routing issues in MPLS). In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science, FOCS '01*, 2001. doi:10.1109/sfcs.2001.959889.

[Gup01] Anupam Gupta. Steiner points in tree metrics don't (really) help. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '01, page 220–227, 2001.

[GX19] Qian-Ping Gu and Gengchun Xu. Constant query time $(1+\varepsilon)$-approximate distance oracle for planar graphs. *Theoretical Computer Science*, 761:78–88, 2019. doi:10.1016/j.tcs.2018.08.024.

[HL22] D. Ellis Hershkowitz and Jason Li. $O(1)$ Steiner point removal in series-parallel graphs. In *30th Annual European Symposium on Algorithms (ESA 2022)*, pages 66:1–66:17, 2022. doi:10.4230/LIPIcs.ESA.2022.66.

[HT84] Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984. doi:10.1137/0213024.

[JLN+05] Lujun Jia, Guolong Lin, Guevara Noubir, Rajmohan Rajaraman, and Ravi Sundaram. Universal approximations for TSP, Steiner tree, and set cover. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 386–395, Baltimore MD USA, May 2005. ACM.

[KKN15] Lior Kamma, Robert Krauthgamer, and Huy L. Nguyen. Cutting corners cheaply, or how to remove Steiner points. *SIAM Journal on Computing*, 44(4):975–995, 2015. doi:10.1137/140951382.

[KKS11] Ken-ichi Kawarabayashi, Philip N. Klein, and Christian Sommer. Linear-space approximate distance oracles for planar, bounded-genus and minor-free graphs. In *The 38th International Colloquium on Automata, Languages and Programming*, ICALP '11, pages 135–146, 2011. doi:10.1007/978-3-642-22006-7_12.

[Kle02] Philip Klein. Preprocessing an undirected planar network to enable fast approximate distance queries. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '02, page 820–827, 2002.

[KLMS22] Omri Kahalon, Hung Le, Lazar Milenković, and Shay Solomon. Can't see the forest for the trees: Navigating metric spaces by bounded hop-diameter spanners. In *Proceedings of the 41st Symposium on Principles of Distributed Computing (to appear)*, PODC '22, 2022. doi:10.48550/ARXIV.2107.14221.

[KNZ14] Robert Krauthgamer, Huy L. Nguyen, and Tamar Zondiner. Preserving terminal distances using minors. *SIAM Journal on Discrete Mathematics*, 28(1):127–141, 2014. doi:10.1137/120888843.

[LWN22] Hung Le and Christian Wulff-Nilsen. Optimal approximate distance oracle for planar graphs. In *IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS '21)*, pages 363–374, 2022. doi:10.1109/FOCS52979.2021.00044.

[RS03] Neil Robertson and Paul D Seymour. Graph minors. XVI. excluding a non-planar graph. *Journal of Combinatorial Theory, Series B*, 89(1):43–76, 2003. doi:10.1016/s0095-8956(03)00042-x.

[Tho04] Mikkel Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *Journal of the ACM*, 51(6):993–1024, 2004. doi:10.1145/1039488.1039493.

[vL76] Jan van Leeuwen. Finding lowest common ancestors in less than logarithmic time, 1976.

[WN16] Christian Wulff-Nilsen. Approximate distance oracles for planar graphs with improved query time-space tradeoff. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithm*, SODA '16, page 351–362, 2016. doi:10.1137/1.9781611974331.ch26.