2012 IEEE 53rd Annual Symposium on Foundations of Computer Science

Split and Join: Strong Partitions and Universal Steiner Trees for Graphs

Costas Busch*, Chinmoy Dutta†, Jaikumar Radhakrishnan‡, Rajmohan Rajaraman† and Srivathsan Srinivasagopalan*
*Division of Computer Science and Eng., School of EECS
Louisiana State University, Baton Rouge, LA 70803, USA
Email: {busch,ssrini1}@csc.lsu.edu
†College of Computer and Information Science
Northeastern University, Boston, MA 02115, USA
Email: {chinmoy,rraj}@ccs.neu.edu
‡School of Technology and Computer Science
Tata Institute of Fundamental Research, Mumbai 400005, India
Email: jaikumar@tifr.res.in

***Abstract*—We study the problem of constructing universal Steiner trees for undirected graphs. Given a graph G and a root node r, we seek a single spanning tree T of minimum *stretch*, where the stretch of T is defined to be the maximum ratio, over all terminal sets X, of the cost of the minimal sub-tree T_X of T that connects X to r to the cost of an optimal Steiner tree connecting X to r in G. Universal Steiner trees (USTs) are important for data aggregation problems where computing the Steiner tree from scratch for every input instance of terminals is costly, as for example in low energy sensor network applications.**

We provide a polynomial time UST construction for general graphs with $2^{O(\sqrt{\log n})}$-stretch. We also give a polynomial time polylog(n)-stretch construction for minor-free graphs. One basic building block of our algorithms is a hierarchy of graph partitions, each of which guarantees small strong diameter for each cluster and bounded neighbourhood intersections for each node. We show close connections between the problems of constructing USTs and building such graph partitions. Our construction of partition hierarchies for general graphs is based on an iterative cluster merging procedure, while the one for minor-free graphs is based on a separator theorem for such graphs and the solution to a cluster aggregation problem that may be of independent interest even for general graphs. To our knowledge, this is the first subpolynomial-stretch ($o(n^\epsilon)$ for any $\epsilon > 0$) UST construction for general graphs, and the first polylogarithmic-stretch UST construction for minor-free graphs.

***Keywords*-universal Steiner tree; hierarchical graph partition; minor-free graphs; graph clustering.**

I. Introduction

In this paper, we study universal approximations for the Steiner Tree problem on undirected graphs. In the universal Steiner Tree (UST) problem for graphs, we are given an undirected graph G and a designated root vertex r in G, and the task is to find a *single spanning tree* T of G such that for any set X of terminal vertices, the minimal subtree T_X of T that connects X to r is a good approximation to the optimal Steiner tree connecting X to r in G. The quality of the solution T is given by its *stretch*, which is the maximum ratio of the cost of T_X to the cost of the optimal Steiner tree connecting X to r in G over all terminal sets X.

The universal Steiner tree problem has been studied extensively for the case of metrics where one is allowed to output an "overlay tree", whose edges correspond to paths in the given graph [1], [2], [3], [4]. Equivalently, the case of metrics can be viewed as a complete graph in which all edge weights satisfy the triangle inequality. In fact, for the case of metrics, there have been several important results on extensions of the UST problem and variants seeking sparse network structures that simultaneously approximate the optimal solutions for a range of input instances [5], [6], [7], [2].

The focus of this paper is on the UST problem on arbitrary graphs where we require that the solution being sought is a spanning tree of the given graph. The Minimum Steiner tree problem on a graph can be well-approximated by solving the same problem on the metric induced by the graph and then computing the minimum subtree connecting the terminals. Such an approach, however, does not apply to the UST problem owing to the requirement that the tree *simultaneously* approximate the optimal Steiner tree for *all* terminal sets. Note that this is a much stronger requirement than asking for a probability distribution over spanning trees that has small expected stretch for every terminal set. In the latter case, there might not be any single tree in the distribution that is good for all terminal sets, i.e., for every tree there is a terminal set such that the minimal subtree connecting the terminals to the root has a cost much larger than the optimal steiner tree.

Motivation. Our problem formulation is primarily motivated by information aggregation and data dissemination in sensor and ad-hoc wireless networks [8], [9], [10]. In a sensor network, data is often collected by a central agent that

0272-5428/12 $26.00 © 2012 IEEE
DOI 10.1109/FOCS.2012.45

periodically queries a subset of sensors for their sensed information. In many applications, the queries seek aggregate information which can be transmitted using a low cost tree that aggregates data at intermediate nodes. This reduces the number of transmissions which is crucial as sensors have limited battery life and wireless transmissions are power intensive. It is not realistic, however, to expect the sensors to compute and store a low cost tree for each potential subset of sensors being aggregated as the sensors have limited memory and computational power. In this setting, a universal tree provides a practical solution where the nodes just need to realize a single tree which approximates optimal aggregation trees for all subsets of sensors. Thus, one natural approach is to employ a universal *overlay* tree. This has several disadvantages, however. First, aggregation over the overlay tree requires a physical routing infrastructure that supports point-to-point communication among distant nodes in the network. Second, even if such an infrastructure exists, it may not route packets along minimum-cost paths as required by the overlay tree. Furthermore, aggregation over the overlay tree requires synchronization among distant nodes in the network and incurs overhead in terms of delays and storage. Thus, in some resource-constrained applications, we would ideally want to construct a universal spanning tree as opposed to an overlay tree.

Another motivation to study universal approximation algorithms comes from their relation with differential privacy which was recently established by Bhalgat, Chakrabarty and Khanna [3]. They showed that universal solutions such as USTs are differentially private, and argued that a kind of "strong" lower bounds for universal algorithms implies lower bounds for differentially private ones as well.

From a theoretical standpoint, our motivation is to find out whether the results known for UST and related problems in the metric case can, in fact, be achieved using spanning trees of the underlying graphs. The analogous question for approximating metrics by tree metrics has been answered affirmatively by [11], [12], [13] who showed that nearly logarithmic-stretch spanning trees exist for all graphs, almost matching the best bound achievable by tree metrics [14]. No comparable results are known for the UST problem.

A. Our results and techniques

Our main results are UST algorithms for general graphs and for the special class of minor-free graphs.

- **UST for general graphs:** We present a polynomial-time algorithm for computing a $2^{O(\sqrt{\log n})}$-stretch spanning tree for any undirected graph.
- **UST for minor-free graphs:** We present a polynomial-time algorithm for computing a $\text{polylog}(n)$-stretch spanning tree for any graph that is H-minor free for any finite graph H.

While the specific techniques used in the two algorithms are substantially different, both are grounded in a common general framework that draws close connections between USTs and certain graph partitions based on strong diameter. We define an (α, β, γ)-partition of a graph G as a partition of the vertices of G into clusters such that each cluster has strong diameter at most $\alpha\gamma$, and for every vertex, the ball of radius γ in G intersects at most β clusters. A primary motivation to study these partitions is the following result.

- **From USTs to partitions:** If every n-vertex graph has a $\sigma(n)$-stretch UST for some function σ, then for any real $\gamma > 0$, every n-vertex graph has an $(O(\sigma(n)^2), O(\sigma(n)), \gamma)$-partition. Moreover, such a partition can be efficiently constructed given black-box access to a $\sigma(n)$-stretch UST algorithm. (Section III-A)

While the above result says that one cannot construct USTs without (implicitly) constructing these graph partitions, the significance of our framework stems from our next result that one can also efficiently construct USTs from these strong partitions. We define an (α, β, γ)-partition hierarchy as a sequence of partitions starting from the trivial partition in which each vertex forms its own cluster, and the ith partition is an $(\alpha, \beta, \gamma^i)$-partition that coarsens the $(i-1)$th partition. (See Section II for formal definitions.) Given a partition hierarchy, a natural divide-and-conquer method to construct a UST (similar to one employed in [1] for metric UST) is to connect together subtrees recursively computed for lower levels of the hierarchy. This approach, however, does not work. In fact, we prove that *any* UST construction that *strictly obeys* the connectivity structure of the hierarchy, in the sense that the subgraph of the tree induced by every cluster of the hierarchy is connected, will have poor stretch in the worst case (see Section III-B1). We overcome this obstacle by introducing the novel notion of spanning trees that *approximately respect* a given partition hierarchy; such a tree may be disconnected within a cluster of the hierarchy, but is joined externally so as to approximately respect the distances within every cluster. We show how to construct such spanning trees from a given partition hierarchy and prove that they achieve desired stretch factors.

- **From partition hierarchies to USTs:** For any graph G, given an (α, β, γ)-partition hierarchy for G, an $O(\alpha^2\beta^2\gamma \log n)$-stretch UST for G can be constructed in polynomial time. (Section III-B)

A major consequence of the above result is that one can obtain a $\text{polylog}(n)$-stretch UST by constructing a $(\text{polylog}(n), \text{polylog}(n), \text{polylog}(n))$-partition hierarchy. Note that there is an $\Omega(\log n)$ lower bound on the best stretch achievable, even in the metric case [2], [3]. We next obtain our main results for general graphs and minor-free graphs by constructing suitable partition hierarchies.

- **Partition hierarchies for general graphs:** Every graph G has a polynomial-time computable $(2^{O(\sqrt{\log n})}, 2^{O(\sqrt{\log n})}, 2^{O(\sqrt{\log n})})$-partition hierarchy. (Section IV)

- **Partition hierarchies for minor-free graphs:** Every minor-free graph G has a polynomial-time computable $(O(\log^3 n), O(\log^4 n), O(\log^3 n))$-partition hierarchy. (Section VI)

The partition hierarchy for general graphs is obtained by an iterative procedure in which clusters are continually merged by identifying vertices for which the number of intersecting clusters within a specified distance exceeds the desired bound. The particular order in which the vertices are processed is carefully chosen; a natural greedy approach fails.

Our construction of the partition hierarchy for minor-free graphs is more complicated. It is based on a separator theorem due to [15], [16] which builds on [17] and shows that any minor-free graph can be decomposed into connected components, each of which contains at most half the number of nodes, by removing a sequence of a constant number of shortest paths. A key ingredient of our hierarchical construction for minor-free graphs is a result on cluster aggregation in general graphs, which is of independent interest.

- **Cluster aggregation:** We show that given any partition of G into disjoint clusters each with strong diameter at most D, and a set S of portal vertices, we can aggregate the clusters into disjoint connected components, each component with a distinguished portal from S, such that for any vertex v, the distance, within the component of v, from v to the distinguished portal in the component is at most $O(\log^2 n)D$ more than the distance of v to S in G. (Section V)

Due to space constraints, we have omitted several proofs and complete description of some algorithms in this extended abstract. We refer the reader to the full paper [18] for all details.

B. Related work

Research in network design over the past decade has revealed that it is often possible to derive sparse network structures (e.g., routes, multicast trees) that yield good approximations simultaneously for a range of input instances. One of the earliest examples of such a result is due to Goel and Estrin [5] who introduced the problem of *simultaneous single sink buy-at-bulk* and gave an $O(\log D)$ bound on the simulataneous ratio where D is the total demand. The guarantee is that their solution works simultaneously for all fusion cost function f which are concave and monotonically non-deceasing with $f(0) = 0$. In a related paper [7], Goel and Post constructed a distribution over trees such that the expected cost of a tree for any f is within an $O(1)$-factor of the optimum cost for that f. A recent improvement by Goel and Post [6] provides the first constant guarantee on the simultaneous ratio achievable by a tree. This result is incomparable to our results since the set of terminals that are being aggregated in the buy-at-bulk problem are fixed.

Jia et al. [1] introduced the notion of universal approximation algorithms for optimization problems, focusing on TSP, Steiner Tree and set cover problems. For the universal Steiner tree problem, they presented polynomial-time algorithms that construct overlay trees with a stretch of $O(\log^4 n/\log\log(n))$ for arbitrary metrics and logarithmic stretch for doubling, Euclidean, or growth-restricted metrics. At a high-level, our approach of using partition hierarchies to derive USTs is similar to that of [1]. There are several critical differences, however. First, as we discussed in Section I-A, the natural divide-and-conquer method employed in [1] of constructing the UST fails for graphs. Second, the construction of *strong partitions* for graphs (as opposed to the weak partitions of [1]) require entirely new techniques for both general graphs and minor-free graphs, and introduce new subproblems of independent interest, e.g., the cluster aggregation problem studied in Section V. The work of [1] also provided a lower bound of $\Omega(\log n/\log\log n)$ for UST that holds even when all the vertices are on a plane; for general metrics, this can be improved to $\Omega(\log n)$ [2], [3]. Note that these lower bounds extend to the UST problem on graphs. Lower bounds for universal TSP are given in [19], [20]. For earlier work on universal TSP, see [21], [22].

Gupta, Hajiaghayi and Räcke [2] developed an elegant framework to model *oblivious network design* problems and gave algorithms with poly-logarithmic approximation ratios. They give network structures that are simultaneously oblivious to both the link cost functions (subject to them being drawn from a suitable class) and traffic demand. Their algorithms are based on the celebrated tree metric embeddings of Fakcharoenphol et al. [14] and hierarchical cut-based decompostions of Räcke [23]. For the UST problem on metrics, the algorithm of [2] builds a UST as follows: First obtain $O(\log n)$ trees from the distribution of [14]; next assign each non-root vertex to a tree that well-approximates its distances to all other nodes; finally, take the union, over each of the $O(\log n)$ overlay trees, the subtree of the tree induced by the root and the vertices assigned to the tree. The resulting overlay tree is an $O(\log^2 n)$-stretch UST.

A potential approach to solving the UST problem on graphs is to adapt the techniques of [2] with $O(\log n)$ spanning trees drawn from the distributions of [11] instead of the overlay trees of [14]. A major challenge here is that the paths or subtrees chosen from the different $O(\log n)$ trees may share several vertices and hence create unavoidable cycles when combined. The only prior work on constructing universal Steiner trees for graphs is due to Busch et al. [4] who achieved a stretch of $O(\log^3 n)$ for the restricted class of graphs with bounded doubling dimension by showing how one can continually refine an $O(\log n)$-stretch overlay tree by removing cycles to obtain an $O(\log^3 n)$-stretch UST. Their techniques, however, are closely tied to the particular class of graphs and seem difficult to generalize. We also note that the spanning tree constructions aimed at minimizing

average stretch [24], [11], [13], [12] with respect to distance do not yield any useful bounds for our stretch measure with respect to optimal Steiner trees.

As mentioned in Section I-A, our universal Steiner trees are based on certain partitions of graphs where we would like to bound the strong diameter of the clusters while maintaining some sparsity constraints. Such partitions have been extensively studied [25], [26]. While nearly optimal partitions based on weak diameter bounds are known in many cases, strong-diameter based decompositions are less understood [25]. There have been recent results on strong-diameter decompositions [13], [27], [12], [11]; while our partitions share some of the ideas (e.g., of stitching together judiciously chosen shortest paths), there are significant differences in the details and the particular partitions being sought. In particular, none of the proposed partitions satisfy the requirement that the neighborhood around every node intersects a small number of clusters. Furthermore, while we seek partition hierarchies with deterministic guarantees, many previous results concerned hierarchies with either probabilistic or averaging guarantees or covers where clusters are allowed to overlap.

II. Definitions and notations

Let $G = (V, E, w)$ denote a weighted undirected graph, where V and E are the sets of vertices and edges, respectively, and $w : E \to \mathbb{R}$ is the length function on edges. We assume, without loss of generality, that the minimum edge length is 1, since otherwise we can scale all the edge lengths appropriately. The length of a path is simply the sum of the lengths of the edges in it. For any u and v in V, the distance between u and v, denoted by $d(u, v)$, is the length of a shortest path between u and v, according to w. For $v \in V$ and real number ρ, let $B(v, \rho)$ denote the ball of radius ρ centered at v, i.e., $B(v, \rho)$ is the set of all vertices that are at distance at most ρ from v, including v. The *diameter* of the graph, denoted by $\textsc{Diam}(G)$, is the maximum distance between any two vertices of G.

For any graph G and any subset X of vertices in G, let $G[X]$ denote the subgraph of G induced by X. For any subset X of vertices and u, v in X, let $d_X(u, v)$ denote the distance between u and v in $G[X]$.

Universal Steiner tree. Given a specified *root* vertex $r \in V$ and a set of *terminal* vertices $X \subseteq V$, a Steiner tree T for X is a minimal subgraph of G that connects the vertices of X to the root. The *cost* of a tree T, denoted by $\textsc{Cost}(T)$, is the sum of the lengths of edges in it. Assume G and r to be fixed. We let $\textsc{Opt}(X)$ denote the cost of the minimum cost steiner tree connecting X to r. Given a spanning tree T of G and terminal set X, we define its projection on the terminal set X, denoted by T_X, as the minimal subtree of T rooted at r that contains X.

Definition 1 (Universal Steiner tree (UST)): Let G be an weighted undirected graph, and r be a specified root vertex in V. We define the *stretch* of a spanning tree T of G to be $\max_{X \subseteq V} \textsc{Cost}(T_X)/\textsc{Opt}(X)$. The **universal Steiner tree** problem is to find a spanning tree with minimum stretch.

Partitions. A *partition* $\mathcal{P}$ of V is a collection of disjoint subsets of V whose union equals V. We refer to each element of $\mathcal{P}$ as a *cluster* of the graph G. There are two notions for the diameter of a cluster C. This paper focuses on the *strong* diameter, denoted by $\textsc{Diam}(C)$, which is the diameter of the subgraph induced by the cluster, i.e. $\textsc{Diam}(G[C])$. In contrast, the *weak* diameter of a cluster is simply the maximum distance between any two verices of the cluster in G.

Definition 2 ((α, β, γ)-partition): An (α, β, γ)-partition $\mathcal{P}$ of G is a partition of V satisfying:

1) **Strong diameter:** The strong diameter of every cluster C in $\mathcal{P}$ is at most $\alpha\gamma$; i.e., $\textsc{Diam}(C) \leq \alpha\gamma$.
2) **Cluster-valence:** For every vertex v in V, $B(v, \gamma)$ has a nonempty intersection with at most β clusters in $\mathcal{P}$. We refer to β as the cluster-valence of $\mathcal{P}$.

A notion of partition similar to our (α, β, γ)-partition appeared in Jia et al. [1], which required a bound on the weak diameter of clusters.

Definition 3 (Partition hierarchy): For $\gamma > 1$, an (α, β, γ)-partition hierarchy of a graph G is a sequence $\mathcal{H} = \langle \mathcal{P}_0, \mathcal{P}_1, \ldots, \mathcal{P}_d \rangle$ of partitions of V, where $d = \lceil \log_\gamma(\textsc{Diam}(G)/\alpha) \rceil$, satisfying:

1) **Partition:** For $0 \leq i \leq d$, $\mathcal{P}_i$ is an $(\alpha, \beta, \gamma^i)$-partition of G. Furthermore, $\mathcal{P}_d$ is the collection $\{V\}$. For convenience, we set $\mathcal{P}_{-1}$ to the collection $\{\{v\} \mid v \in V\}$.
2) **Hierarchy:** For $0 \leq i < d$, every cluster in $\mathcal{P}_i$ is contained in some cluster in $\mathcal{P}_{i+1}$.
3) **Root Padding:** For $0 \leq i \leq d$, the ball $B(r, \gamma^i)$ of radius γ^i around root r is contained in some cluster in $\mathcal{P}_i$.

For a partition $\mathcal{P}$ of a graph G, let $\widehat{G}[\mathcal{P}]$ denote a weighted graph in which the vertex set is $\mathcal{P}$, and there is an edge (C, C') between clusters C and C' if G has an edge between a vertex in C and a vertex in C'; the length of the edge (C, C') is the minimum length of an edge between C and C' in G.

For a partition $\mathcal{P}$, let $\mathcal{P}(v)$ denote the cluster of $\mathcal{P}$ that contains the vertex v and $\textsc{MaxDiam}(\mathcal{P})$ denote $\max_{C \in \mathcal{P}} \textsc{Diam}(C)$. For a subset X of vertices, let $\mathcal{P}[X]$ denote the partition restricted to X; i.e., $\mathcal{P}[X]$ is the collection $\{X \cap C \mid C \in \mathcal{P}\}$. For a partition hierarchy $\mathcal{H}$ and a cluster C that is an element of a partition $\mathcal{P}_i$ in $\mathcal{H}$, we let $\mathcal{H}[C]$ denote the partition hierarchy projected to C; that is, $\mathcal{H}[C] = \langle \mathcal{P}_0[C], \ldots, \mathcal{P}_i[C] \rangle$. Let T be a spanning tree and $\mathcal{H}$ be an (α, β, γ)-partition hierarchy of G. We say that T *strictly obeys* $\mathcal{H}$ if for each $\mathcal{P}_i \in \mathcal{H}$ and each cluster $C \in \mathcal{P}_i$, the subgraph of T induced by C is connected. We say that T *μ-respects* $\mathcal{H}$ if for each $\mathcal{P}_i \in \mathcal{H}$, each $C \in \mathcal{P}_i$ and every pair of vertices $u, v \in C$, $d_T(u, v)$ is at most $\mu\alpha\gamma^i$.

III. Strong partitions and universal Steiner trees

We now present close connections between the strong partitions of Definition 2 and universal Steiner trees. We first show in Section III-A that partitions with low strong diameter and low cluster-valence are necessary for deriving low-stretch trees. We next show in Section III-B how partition hierarchies yield USTs. Given an (α, β, γ)-partition hierarchy for any graph G, Section III-B1 shows how to get an $O((\alpha\beta)^{\log_\gamma n}\gamma\beta^2 \log_\gamma n)$-stretch UST for G that strictly obeys the partition hierarchy, and also presents a nearly matching lower bound on the stretch for such USTs. Section III-B2 then presents an improved $O(\alpha^2\beta^2\gamma \log_\gamma n)$-stretch UST construction that does not strictly obey but approximately respects the partition hierarchy.

A. From universal Steiner trees to strong partitions

Theorem 4: If every n-vertex graph has a $\sigma(n)$-stretch UST for some function σ, then for any real $\gamma > 0$, every n-vertex graph has an $(O(\sigma(n)^2), O(\sigma(n)), \gamma)$-partition. Moreover, such a partition can be efficiently constructed given black-box access to a $\sigma(n)$-stretch UST algorithm.

B. From partition hierarchies to a universal Steiner trees

We first prove the following important lemma, showing the significance of μ-respecting trees.

Lemma 5: A spanning tree T that μ-respects an (α, β, γ)-partition hierarchy has a stretch of $O(\mu\alpha\beta\gamma \log n)$.

Proof: Let $\langle \mathcal{P}_i \rangle$ denote the given (α, β, γ)-partition hierarchy. Fix a non-empty set X of vertices. Note that X is assumed to not contain the root r. For each cluster C in the partition hierarchy such that $C \cap (X \cup \{r\})$ is nonempty, let $v(C)$ denote an arbitrary vertex in $C \cap (X \cup \{r\})$.

We place an upper bound on the cost of T_X, the subgraph of T connecting the vertices in X to the root r, as follows. Let n_i denote the number of clusters in $\mathcal{P}_i$ that $X \cup \{r\}$ intersects. Since we have defined $\mathcal{P}_{-1}$ to be the trivial clustering consisting of a singleton set for each vertex, n_{-1} is simply $|X \cup \{r\}|$. Let j be the smallest integer such that X is a subset of the cluster in $\mathcal{P}_j$ that contains r. In other words, n_j equals 1 and $n_i > 1$ for all $-1 \le i < j$. Fix an i, $-1 \le i < j$. Let C be any cluster of $\mathcal{P}_i$ that intersects $X \cup \{r\}$, and let C' denote the cluster of $\mathcal{P}_{i+1}$ that contains C. Since T μ-respects the partition hierarchy, it follows that the length of the path from $v(C)$ to $v(C')$ in T is at most $\mu\alpha\gamma^{i+1}$. Therefore, the cost of T_X is at most $\sum_{-1 \le i < j} n_i \mu\alpha\gamma^{i+1}$. Let $I = \{i \mid (i = j) \vee (-1 \le i < j \wedge \exists p : n_i \ge 2^p \wedge n_{i+1} < 2^p)\}$. For $\ell \in I$, let $I_\ell = \{i \mid (-1 \le i \le \ell) \wedge \neg(\exists \ell' \in I : i \le \ell' < \ell)\}$. We have

$$\begin{aligned}\sum_{i \in I_\ell} n_i \mu\alpha\gamma^{i+1} &\le \sum_{i \in I_\ell} 2n_\ell \mu\alpha\gamma^{i+1} \le \sum_{-1 \le i \le \ell} 2n_\ell \mu\alpha\gamma^{i+1} \\ &= O(n_\ell \mu\alpha\gamma^{\ell+1}).\end{aligned}$$

We next place a lower bound on $\textsc{Opt}(X)$. Fix an i, $0 \le i < j$. By the cluster-valence property of the hierarchy, any ball of radius γ^i intersects at most β clusters in $\mathcal{P}_i$. Thus, there are at least $\lceil n_i/\beta \rceil$ vertices in X that are at pairwise distance at least γ^i from one another. This implies that $\textsc{Opt}(X)$ is at least $(\lceil n_i/\beta \rceil - 1)\gamma^i$. If $\lceil n_i/\beta \rceil = 1$, we invoke the padding property which says there is at least one vertex in X that is at distance at least γ_i from the root, implying a lower bound of γ^i on $\textsc{Opt}(X)$. Combining the two bounds, we obtain a lower bound of $\Omega(n_i\gamma^i/\beta)$. For $i = -1$, we also have a lower bound of n_{-1} since the minimum edge-weight is 1. Noting that $|I| = O(\log n)$, we get the stretch of $T(G)$ to be

$$\begin{aligned}O\left(\sum_{\ell \in I} \frac{\sum_{i \in I_\ell} n_i \mu\alpha\gamma^{i+1}}{\textsc{Opt}(X)}\right) &= O\left(\sum_{\ell \in I} \frac{n_\ell \mu\alpha\gamma^{\ell+1}}{n_\ell\gamma^\ell/\beta}\right) \\ &= O\left(\sum_{\ell \in I} \mu\alpha\gamma^{\ell+1}\beta/\gamma^\ell\right) = O(\mu\alpha\beta\gamma \log n).\end{aligned}$$

■

1) A basic bottom-up algorithm: We first present a bottom-up algorithm for constructing a spanning tree T from a partition hierarchy that strictly obeys it. Though the stretch achieved by the spanning tree is much weaker than what we obtain by a different algorithm, it helps develop our improved algorithm.

Algorithm UST:BASIC

Require: Undirected graph G, (α, β, γ)-partition hierarchy $\langle \mathcal{P}_i : -1 \le i \le d = \lceil \log_\gamma(\frac{\textsc{Diam}(G)}{\alpha}) \rceil \rangle$ for G.
Ensure: A spanning tree T of G.

1: For every cluster C in $\mathcal{P}_{-1}$, set $T(C)$ to $\emptyset$.
2: **for** level i from 0 to d **do**
3: **for** cluster C in $\mathcal{P}_i$ **do**
4: For an edge $e = (C_1, C_2)$ in $\widehat{G[C]}[\mathcal{P}_{i-1}[C]]$, let $m(e)$ denote the edge between C_1 to C_2 in $G[C]$ that has minimum weight. (Recall that $G[C]$ is the subgraph of G induced by C and $\mathcal{P}_{i-1}[C]$ is the partition $\mathcal{P}_{i-1}$ restricted to the set C.)
5: Compute a shortest path tree T' from an arbitrary source vertex in $\widehat{G[C]}[\mathcal{P}_{i-1}[C]]$.
6: Set $T(C)$ to be the union of $\cup_{C' \in \mathcal{P}_{i-1}[C]} T(C')$ and $\{m(e) : e \in T'\}$.
7: **end for**
8: **end for**
9: Set T to be $T(V)$. (Note that V is the lone cluster in $\mathcal{P}_d$.)

Theorem 6: For any graph G, given an (α, β, γ)-partition hierarchy, an $O((\alpha\beta)^{\log_\gamma n}\gamma\beta^2 \log n)$-stretch UST is computed by Algorithm UST:BASIC in polynomial time.

We complement the above construction by an almost matching lower bound for stretch achievable by any spanning tree that strictly obeys a partition hierarchy.

Theorem 7: Let α, $\beta < \gamma$. There exists a graph G and an (α, β, γ)-partition hierarchy $\mathcal{H}$ of G such that any spanning tree T of G that strictly obeys $\mathcal{H}$ has sretch $\Omega((\alpha\beta)^{\frac{\log_\gamma n}{4}}\gamma)$.

2) Split and join: An improved top-down algorithm: The tree returned by Algorithm UST:BASIC strictly obeys the given partition hierarchy. In doing so, however, it pays a huge cost in the distances within the cluster which is unavoidable.

We now present a much more careful construction of a universal Steiner tree which does not enforce the connectivity constraint within clusters; that is, we use a given partition hierarchy $\mathcal{H}$ to build a tree T in which $T[C]$ may be disconnected. By allowing this disconnectivity within clusters, however, we show that we can build a tree that μ-respects the given hierarchy for a much smaller μ, assuming γ is sufficiently large. The pseudocode is given below in Algorithm UST:SPLIT-JOIN.

Algorithm UST:SPLIT-JOIN

Require: Undirected graph $G = (V, E)$, a nonempty set $S_G \subseteq V$ of portals, a partition hierarchy $\mathcal{H} = \{\mathcal{P}_0, \mathcal{P}_1, \ldots, \mathcal{P}_\ell\}$.

Ensure: A forest F that connects every vertex in V to S_G.

1: If the graph consists of a single vertex, then simply return the vertex as the forest.
2: For an edge $e = (C_1, C_2)$ in $\widehat{G}[\mathcal{P}_\ell]$, let $m(e)$ denote the minimum-weight edge from C_1 to C_2 in G.
3: Let $\widehat{S}$ denote the set of clusters that have a nonempty intersection with S_G.
4: For every cluster C in $\widehat{S}$, set S_C to be $C \cap S$.
5: Compute a shortest path forest $\widehat{F}$ in $\widehat{G}[\mathcal{P}_\ell]$ rooted at $\widehat{S}$.
6: **for** cluster C in $\mathcal{P}_\ell$ in order of decreasing distance from $\widehat{S}$ in $\widehat{F}$ **do**
7: **if** C is a leaf node in $\widehat{F}$ **then**
8: Set RANK(C) to be 0, S_C to be {tail of $m(e)$} where e is the edge connecting C to its parent in $\widehat{F}$.
9: **else**
10: Let MAXC be $\max\{\text{RANK}(C') \mid C' \text{ is child of } C\}$. Set FAV$(C)$ to be a child of C with rank MAXC. Set HIGHWAY(C) to be a shortest path in C from the head of $m(e)$ to the tail of $m(e')$ where e and e' are the edges connecting FAV(C) to C and C to its parent, respectively, in $\widehat{F}$. Set S_C to be the set of nodes in HIGHWAY(C).
11: **if** there exist at least two children of C in $\widehat{F}$ whose rank equals MAXC **then**
12: Set RANK(C) to be MAXC $+ 1$
13: **else**
14: Set RANK(C) to be MAXC
15: **end if**
16: **end if**
17: **end for**
18: **for** each cluster C in $\mathcal{P}_\ell$ **do**
19: Compute $F(C) = \text{UST}(G[C], S_C, \mathcal{H}[C])$
20: **end for**
21: Return F to be the union of $\bigcup_{C \in \mathcal{P}_\ell}$ HIGHWAY(C), $\bigcup_{C \in \mathcal{P}_\ell} F(C)$, and $\{m(e) : e \in \widehat{F}\}$.

We have presented Algorithm UST:SPLIT-JOIN in a more general context where the goal is to compute a forest rooted at a given set of portals. To obtain a UST, we invoke the algorithm with the portal set being the singleton set consisting of the root.

Lemma 8: The output F of the algorithm is a spanning forest, each tree containing exactly one vertex in S_G.

Lemma 9: Let F be the final forest returned by the algorithm. For any cluster C, when UST is called on cluster C, either S_C is a subset of S_G or for any two vertices u and v in S_C, $d_F(u, v)$ is at most $d_C(u, v)$.

Lemma 10: The rank of any cluster C in partition $\mathcal{P}_\ell$ is at most $\log(|\mathcal{P}_\ell|)$.

Lemma 11: Let F be the final forest returned by the algorithm. If $\gamma \geq 3 \log n$, then for any cluster C in $\mathcal{P}_i$ and vertex u in C, $d_F(u, S_C)$ is at most $3\alpha^2\beta\gamma^i$.

Proof: We prove by induction on level i that $d_F(u, S_C)$ is at most $3\alpha^2\beta\gamma^i$, with the base case being $i = 0$. In this case, the cluster and its portal set are the same singleton vertex set, trivially yielding the desired claim. For the induction step, we consider $i > 0$. Let C be a cluster of $\mathcal{P}_i$. For any vertex u in C, let C_u denote $\mathcal{P}_{i-1}(u)$, that is, the cluster in partition $\mathcal{P}_{i-1}$ that contains u.

As in the algorithm, let $\widehat{S}$ denote the set of clusters in the partition of C that intersect S_C. Let $C_u = C_0, C_1, \ldots, C_k$, where $C_k \in \widehat{S}$, denote the sequence of clusters in the unique path from C_u to $\widehat{S}$ in $\widehat{G}[\mathcal{P}_\ell]$, which we refer to as the supergraph in the following argument. Note that C_i is the parent of C_{i-1} in the supergraph. By our argument in the proof of Theorem 6, we know that k is at most $\alpha\beta\gamma$. We now argue that there are at most $\log n$ elements C_i in the sequence such that C_i is not FAV(C_{i+1}). To see this, we note that if C_i is not FAV(C_{i+1}), then RANK(C_{i+1}) strictly exceeds RANK(C_i). Since the rank of any cluster is at most $\log n$ by Lemma 10, the desired claim holds.

This sequence of clusters induces a path from u to S_C, which consists of (a) the connecting edges in the supergraph, (b) the highway in each cluster C_i in the sequence, (c) for each cluster C_i such that C_{i-1} is not a favorite of C_i, the unique path in $F(C_i)$ (and, hence, in F) that connects the head of the edge connecting C_{i-1} and C_i to S_{C_i}. Since the number of clusters in the sequence is at most $\alpha\beta\gamma$, and the highway in each cluster is a shortest path of length at most $\alpha\gamma^{i-1}$, the total length of the paths in (a) and (b) is at most $2\alpha^2\beta\gamma^i$. The number of clusters in (c) is at most $\log n$, and by the induction hypothesis, the length of each path in (c) is at most $3\alpha^2\beta\gamma^{i-1}$. We thus have,

$$\begin{aligned} d_F(u, S_C) &\leq 2\alpha^2\beta\gamma^i + (3\log n)\alpha^2\beta\gamma^{i-1} \\ &\leq 3\alpha^2\beta\gamma^i \end{aligned}$$

for $\gamma \geq 3 \log n$, thus completing the proof of the lemma. ■

Lemma 12: The forest F returned by the algorithm $7\alpha\beta$-respects $\mathcal{H}$.

Proof: We show that for any cluster C in $\mathcal{P}_i$, and vertices u, v in C, $d_F(u,v)$ is at most $7\alpha^2\beta\gamma^i$; this will establish the desired claim. By Lemma 11, $d_F(u, S_C)$ and $d_F(v, S_C)$ are both at most $3\alpha^2\beta\gamma^i$. By Lemma 9, for any two nodes x and y in S_C, $d_F(x,y)$ is at most the strong diameter of C, which is at most $\alpha\gamma^i$. Putting these three distances together, we obtain that $d_F(u,v)$ is at most $7\alpha^2\beta\gamma^i$. ∎

Theorem 13: Given an undirected graph G, portal set $S_G = \{r\}$, where r is an arbitrary vertex of G, and (α, β, γ)-partition $\mathcal{H}$ of G as input, Algorithm UST:SPLIT-JOIN returns an $O(\alpha^2\beta^2\gamma \log n)$-stretch UST.

Proof: By Lemma 8, the output F is a spanning forest, each tree of which contains exactly one vertex of S_G. Since S_G has only one vertex, the forest F returned is a tree. By Lemma 12, F $(7\alpha\beta)$-respects $\mathcal{H}$. By Lemma 5, we obtain that F has stretch $O(\alpha^2\beta^2\gamma \log n)$. ∎

IV. Partition hierarchy for general graphs

In this section we present our algorithm for obtaining a partition hierarchy for general graphs. Our main result is the following.

Theorem 14: Fix integer $k \geq 1$ and $\epsilon > 0$. For any graph G, a hierarchical $((\frac{4}{3}+\epsilon)4^{k-1} - \frac{4}{3}, kn^{\frac{1}{k}}, \gamma)$-partition can be constructed in polynomial time for $\gamma \geq \frac{1}{\epsilon}((\frac{4}{3}+\epsilon)4^{k-1} - \frac{4}{3})$. In paricular, setting $k = \lceil\sqrt{\log n}\rceil$ and $\epsilon = 1$, a hierarchical $(2^{O(\sqrt{\log n})}, 2^{O(\sqrt{\log n})}, 2^{O(\sqrt{\log n})})$-partition for any graph can be constructed in polynomial time.

Algorithm. For notational convenience, we start building the hierarchy at level -1 by defining $\mathcal{P}_{-1}$ as the trivial partition where every vertex is in its own cluster. For $i = 0, 1, \ldots, d = \lceil \log_\gamma \frac{\text{DIAM}(G)}{\alpha} \rceil$, we build the ith level of the hierarchy, $\mathcal{P}_i$, after building the previous levels. Assuming that $\mathcal{P}_{i-1}$ has been constructed, we construct $\mathcal{P}_i$, as follows.

Construction of level i: Clusters of $\mathcal{P}_i$ are formed in successive stages starting from stage 0. We assign a *rank* to each cluster based on the stage in which it is created: a cluster formed in stage j gets the rank j. (All the clusters of level -1 are assigned the rank 0.) We will denote the set of clusters of rank j in $\mathcal{P}_i$ by S^i_j. At all times while building $\mathcal{P}_i$, we maintain a partitioning of the graph, i.e., we guarantee that each vertex of the graph is contained in exactly one cluster of $\mathcal{P}_i$. The partitioning, however, may change as clusters of a higher ranks are formed by merging clusters of lower ranks.

Stage 0: In stage 0, we simply add all the clusters of $\mathcal{P}_{i-1}$ to S^i_0.

Stage $j > 0$: For $j > 0$, stage j works in two phases, one after another.

- **First phase:** In the first phase, we repeatedly look for a vertex *contained in a cluster of rank at most $j-1$* such that the ball of radius γ^i around it, $B(v, \gamma^i)$, intersects more than $n^{\frac{1}{k}}$ clusters of rank precisely $j-1$. If we find such a vertex v, we merge the cluster containing v with all the clusters of rank $j-1$ that $B(v, \gamma^i)$ intersects. This newly created cluster is assigned the rank j and added to S^i_j while all the clusters that were merged to form it are deleted from their respective $S^i_{j'}$'s. The first phase ends when we can no longer find any such vertex v.
- **Second phase:** In the second phase, we repeat a simlar procedure for vertices *contained in clusters of rank j*. As long as we can find a vertex v in a cluster of rank j such that $B(v, \gamma^i)$ intersects more than $n^{\frac{1}{k}}$ clusters of rank $j-1$, we merge the cluster containing v with all the clusters of rank $j-1$ that $B(v, \gamma^i)$ intersects to form a new cluster of rank j. We include this new cluster in S^i_j and delete all the clusters that were merged to form it from their respective $S^i_{j'}$'s. The second phase, and also the stage j, ends when we cannot find any such vertex v, and the next stage begins.

If no new cluster gets formed in the first phase of a stage, the construction of level i of the hierarchy finishes and $\mathcal{P}_i$ is defined as simply the union of all the non empty S^i_j's.

Remark. Although the two phases of a stage are quite similar and one might be tempted to do away with this particular ordering of mergings, the naive approach without the ordering does not work. Having a careful order in which mergings are carried out enables us to control the growth of the strong diameter of the clusters. To see this, consider a cluster formed in the second phase of some stage j. It contains a unique cluster that was formed in the first phase of stage j. Call it the core. Our ordering ensures that only the vertices in the core can lead to mergings in the second phase of stage j. This is because for any vertex v outside the core, $B(v, \gamma^i)$ intersects at most $n^{\frac{1}{k}}$ clusters of rank $j-1$, otherwise the first phase would not have ended. Thus the mergings of the second phase cannot increase the diameter by too much as the new vertices are always "close" to the core.

Using Theorem 14 and Theorem 13, we get our USTconstruction for general graphs.

Corollary 15: A $2^{O(\sqrt{\log n})}$-stretch universal Steiner tree can be computed in polynomial time for any undirected graph.

V. The Cluster aggregation problem

In this section, we define the Cluster Aggregation problem which arises when building partition hierarchies for minor-free graphs (see Section VI). Our problem formulation and algorithm, however, apply to arbitrary graphs and may be of independent interest. Indeed, our cluster aggregation algorithm is useful for building other strong-diameter based hierarchical partitions with applications to distributed computing [28].

Definition 16 (Cluster Aggregation): Given a graph $G = (V, E)$, partition $\mathcal{P}$ of G, set $S \subseteq V$ of *portals*, a *cluster aggregation* is a function $\text{DEST} : \mathcal{P} \to S$. The function DEST naturally induces a new partition $\mathcal{Q} = \{\bigcup_{C:\text{DEST}(C)=s} C \mid s \in S\}$ that coarsens $\mathcal{P}$. For each vertex v in V, we define the *detour* $\text{DTR}_{\text{DEST}}(v)$ for v under DEST to be the difference between the distance from v to S in G and the distance from v to $\text{DEST}(\mathcal{P}(v))$ in subgraph of G induced by the cluster in $\mathcal{Q}$ that contains v; i.e., $\text{DTR}_{\text{DEST}}(v) = (d_{G[\mathcal{Q}v]}(v, \text{DEST}(C)) - d(v, S))$. We define the detour of DEST to be $\max_{v \in V} \text{DTR}_{\mathcal{Q}}(v)$. The goal of the cluster merging problem is to find a cluster aggregation with minimum detour.

Our algorithm for the Cluster Aggregation problem proceeds in $O(\log n)$ phases. Each phase has a number of iterations. Each iteration aggregates a subset of the clusters in $\mathcal{P}$ and assigns the same DEST value for each of them. The selection of clusters in a particular iteration is based on how shortest paths from these clusters to S proceed through the graph. The interaction of these shortest paths is captured by means of auxiliary directed graph. In the full paper, we give the complete algorithm and show that it solves the Cluster Aggregation problem for a given partition $\mathcal{P}$ with a detour of $O(\log^2(|\mathcal{P}|)\text{MAXDIAM}(\mathcal{P}))$.

Theorem 17: The detour for any vertex v in G in the cluster merger returned by the algorithm is at most $\log^2(|\mathcal{P}|)\text{MAXDIAM}(\mathcal{P})$.

VI. Partition hierarchy for minor-free graphs

A weighted graph G is H-minor free if zero or more edge contractions on G does not give a graph isomorphic H. Minor-free graphs are special cases of k-path separable graphs. A graph G is *k-path separable* [16] if there exists a subgraph S, called the *k-path separator*, such that: (i) $S = S_1 \cup S_2 \cup \cdots \cup S_l$, where for each $1 \leq i \leq l$, subgraph S_i is the union of k_i paths where each path is shortest in $G \setminus \bigcup_{1 \leq j < i} S_j$ with respect to its end points; (ii) $\sum_i k_i \leq k$; (iii) either $G \setminus S$ is empty, or each connected component of $G \setminus S$ is k-path separable and has at most $n/2$ nodes.

Thorup [15] shows that any planar graph G is 3-path separable, where all paths in the separator S belong in S_1, that is, they are shortest paths in G. Abraham and Gavoille [16] generalize the result to any H-minor free graph, where they show for fixed size H the graph is k-path separable, for some $k = k(H)$, and the k-path separator can be computed in polynomial time. Interesting classes of H-minor free graphs are: planar graphs, which exclude K_5 and $K_{3,3}$; outerplanar graphs, which exclude K_4 and $K_{2,3}$; series-parallel graphs, which exclude K_4; and trees, which exclude K_3. Busch *et al.* [29] introduced the technique of using path separators to build clusters for sparse covers in minor-free graphs. Here, we extend that technique to hierarchical partitions.

A. The minor-free clustering algorithm

Consider now an arbitrary weighted H-minor free graph G, for fixed size H. (You may also take G to be an arbitrary k-path separable graph.) We build the partition hierarchy bottom up by coarsening clusters. Suppose we are given a $(\alpha, \beta, \gamma^{i-1})$-partition $\mathcal{P}_{i-1}$, where $i > 0$. We describe how to build a $(\alpha, \beta, \gamma^i)$-partition $\mathcal{P}_i$, such that $\mathcal{P}_{i-1}$ is a refinement of $\mathcal{P}_i$.

Algorithm HIERARCHICAL-PARTITION:MINOR-FREE

Require: Connected component Φ of minor-free graph G, strong $(\alpha, \beta, \gamma^{i-1})$-partition $\mathcal{P}_{i-1}$ of G, set $\mathcal{N}$ with coarsen clusters of $\mathcal{P}_{i-1}$.

Ensure: Coarsening the $\mathcal{P}_{i-1}$ clusters in Φ; the resulting clusters are inserted into $\mathcal{N}$.

1: Let $S = S_1 \cup S_2 \cup \cdots \cup S_l$ be a k-path separator of Φ.
2: **for** χ from 1 to l **do**
3: **for** each path $p \in S_\chi$ **do**
4: Let Ψ be the connected component of $\Phi \setminus \bigcup_{1 \leq j < \chi} S_j$ in which p resides.
5: Invoke Subroutine PATH-CLUSTERING to update $\mathcal{N}$ by appropriately merging clusters of Ψ around p.
6: **end for**
7: **end for**
8: **for** each connected component $\Upsilon \in \Phi \setminus S$ **do**
9: Invoke (recursively) Algorithm HIERARCHICAL-PARTITION:MINOR-FREE with parameters Υ, $\mathcal{P}_{i-1}$, and $\mathcal{N}$.
10: Update $\mathcal{N}$ to be the result of the recursive invocation.
11: **end for**
12: Return $\mathcal{N}$.

High-level recursive structure. The first clusters of partition $\mathcal{P}_i$ are formed around a k-path separator of G by appropriately merging clusters of $\mathcal{P}_{i-1}$ close to the separator paths. We then remove the k-path separator. This may result in the formation of one or more disjoint connected components, each of which is still a H-minor free graph. We repeat the clustering process recursively on each residual connected component, until no nodes remain.

Clustering a connected component. Algorithm HIERARCHICAL-PARTITION:MINOR-FREE implements the recursive decomposition of G. The algorithm actually receives as input an arbitrary connected component Φ of G, which it then decomposes into possibly one or more connected components that are processed recursively. The initial invocation is with $\Phi = G$. The resulting clusters of the $\mathcal{P}_i$ partition of G will appear in a set $\mathcal{N}$ which is initially empty. New clusters formed around path separators are inserted and maintained into $\mathcal{N}$. The new clusters may merge with existing clusters in $\mathcal{N}$ created form previously processed paths. The partition $\mathcal{P}_i$ is the final $\mathcal{N}$ that we obtain after we recursively process all the path separators in each component in G. Let $S = S_1 \cup S_2 \cup \cdots \cup S_l$ be the path separator of Φ. We process the paths of S in sequence

starting from the paths in S_1, then the paths in S_2, and so on. The root node r can be treated as an artificial single node path of the first path separator of G.

Processing a path. Subroutine PATH-CLUSTERING is the central part of the minor-free clustering algorithm. Consider a path $p \in S_\chi$, where S_χ is path set of S in Φ. Let Ψ be the connected component of $\Phi \setminus \bigcup_{1 \le j < \chi} S_j$ in which p resides. Subroutine PATH-CLUSTERING merges clusters of $\mathcal{P}_{i-1}$ which are within distance $2\gamma^i$ from p using the cluster aggregation algorithm of Section V. The choice of this particular aggregation distance is to control the diameter of the new clusters and the amount of intersections in any ball of diameter γ^i.

The subroutine merges only integral clusters of $\mathcal{P}_{i-1}$ which are completely within Ψ, and which we denote $\mathcal{P}^{\Psi}_{i-1}$. Non-integral clusters of $\mathcal{P}_{i-1}$ have already been included in $\mathcal{N}$ from previously processed separator paths. Let $\mathcal{A} \subseteq \mathcal{P}^{\Psi}_{i-1}$ be the clusters within distance $2\gamma^i$ from p which are candidates for merging. We do not include in $\mathcal{A}$ any cluster of $\mathcal{P}^{\Psi}_{i-1}$ which has already been used in $\mathcal{N}$. Of particular interest are the clusters $\mathcal{B} \subseteq \mathcal{A}$ which are neighbours with $\mathcal{N}$ or next to non-integral clusters of $\mathcal{P}_{i-1}$, and these will be handled as special cases.

Let Ψ' be the sub-graph induced by $\mathcal{A}$ (note that Ψ' may not be connected). The clusters in $\mathcal{A}$ are merged by invoking the cluster aggregation algorithm of Section V. We define two sets of nodes L and U in Ψ' which serve as destination portals for the merged clusters. Set L contains the *leaders* of path p, which is a maximal set of nodes in $p \cap \Psi'$ such that for any pair $u, v \in L$, $d_p(u, v) \ge \gamma^i$, and u and v cannot belong to the same cluster of $\mathcal{A}$. Set U contains one arbitrary node from each cluster in $\mathcal{B}$ (from each cluster in $\mathcal{B}$ that does not already contain a node in L).

Let $\mathcal{R}$ contain all resulting merged (coarsen) clusters. We can write $\mathcal{R} = \mathcal{I}_p \cup \mathcal{K}_p$ where $\mathcal{I}_p$ consists of clusters that contain a portal node of L, and $\mathcal{K}_p$ consists of clusters that contain a portal node of U. Each cluster $X \in \mathcal{K}_p$ may further merge with at most one arbitrary adjacent cluster $Y \in \mathcal{N}$, for which there is an edge $(u, v) \in E(\Psi)$ such that $u \in X$, $v \in Y$, and $v \notin \Psi'$. We insert the merged cluster from X and Y back to $\mathcal{N}$. After processing all the clusters in $\mathcal{K}_p$, $\mathcal{N}$ is updated to $\mathcal{N}'$ which includes the new merged clusters from $\mathcal{K}_p$. It can be shown that all the clusters of $\mathcal{K}_p$ will merge with existing clusters of $\mathcal{N}$. Therefore, the new set of coarsen clusters from processing path p will be $\mathcal{N}' \cup \mathcal{I}_p$. Thus, Subroutine PATH-CLUSTERING will return the resulting set $\mathcal{N} = \mathcal{N}' \cup \mathcal{I}_p$.

B. The analysis of minor-free clustering

Consider a minor-free graph G with n nodes. The recursive process of removing path separators defines a decomposition tree T of G, such that each node $t \in T$ corresponds to a connected component of G, which we will denote $G(t)$. The root π of T corresponds to G, namely, $G(\pi) = G$. Denote $S(t)$ the path separator for the respective graph $G(t)$. If $G(t) \setminus S(t) = \emptyset$, then t is a leaf of T. Otherwise, for each connected component $\Phi \in G(t) \setminus S(t)$ there is a node $w \in T$ such that w is a child of t and $G(w) = \Phi$.

According to Algorithm HIERARCHICAL-PARTITION:MINOR-FREE, a newly created cluster which is formed after a path is processed, may grow larger when new clusters merge into it when subsequent paths are processed. Consider a path $p \in S(t)$, for some $t \in T$. We say that a *cluster belongs* to p if it contains a leader of p, where the leaders of p are chosen in Subroutine PATH-CLUSTERING. It can be shown that a cluster in $\mathcal{P}_i$ belongs to exactly one path. A key point of the analysis is that clusters of a path p are far from clusters in sibling nodes of T (any pair of nodes in two sibling clusters are at distance more than $2\gamma^i$ apart). Thus, when we bound intersections in balls of radius γ^i, we only need to consider clusters on the same branch from the root to a leaf of T. Hence, the amount of intersections can be bounded using the depth of T which is $O(\log n)$, and the number of paths k in a separator. Similarly, clusters can only grow along such a branch, which also helps to control the diameter. The following statements can be proven:

Lemma 18: $\mathcal{P}_i$ is a $(\alpha', c_2\alpha' k \log n, \gamma^i)$-partition, where $\alpha' = c_1 k \log^3 n$, for constants c_1 and c_2.

Theorem 19: We can obtain a hierarchical $(O(\log^3 n), O(\log^4 n), O(\log^3 n))$-partition for any minor-free graph G in polynomial time.

From Theorems 13 and 19 we obtain the following corollary.

Corollary 20: A polylog(n)-stretch universal Steiner tree can be computed in polynomial time for any minor-free graph with n nodes.

VII. Concluding remarks

In this paper, we have presented a polynomial-time $2^{O(\sqrt{\log n})}$-stretch UST construction for general graphs, which is the first known subpolynomial-stretch ($o(n^\epsilon)$ for any $\epsilon > 0$) solution for general graphs. We have also presented a polylog(n)-stretch UST algorithm for minor-free graphs, for which $\Omega(\log n)$ is a known lower bound.' Both UST algorithms are based on a framework that draws close connections between a certain class of strong graph partitions and low-stretch USTs. Our modular framework leads us to designing new strong-diameter partitions for both general and minor-free graphs, and solving a new cluster aggregation problem, all of which are of independent interest.

Our work leaves several important open problems. The most compelling one is that of deriving tight bounds on the best stretch achievable for general graphs (specifically, is polylog(n)-stretch achievable?). For minor-free graphs, the exponent in the polylog(n) factor we achieve for stretch is high. Our current analysis follows the modular algorithmic

framework; we believe that an improved bound can be achieved by a more careful "flatter" analysis. Furthermore, any improved approximation for the cluster aggregation problem will yield significant improvements in the UST stretch factors.

ACKNOWLEDGMENT

Costas Busch is supported in part by NSF grant CNS-1018273. Chinmoy Dutta is supported in part by NSF grant CCF-0845003 and a Microsoft grant to Ravi Sundaram. Rajmohan Rajaraman is supported in part by NSF grant CNS-0915985.

REFERENCES

[1] L. Jia, G. Lin, G. Noubir, R. Rajaraman, and R. Sundaram, "Universal approximations for tsp, steiner tree, and set cover," in *Proceedings of ACM STOC*, 2005, pp. 386–39.

[2] A. Gupta, M. T. Hajiaghayi, and H. Räcke, "Oblivious network design," in *Proceedings of ACM-SIAM SODA*, 2006, pp. 970–979.

[3] A. Bhalgat, D. Chakrabarty, and S. Khanna, "Optimal lower bounds for universal and differentially private steiner trees and tsps," in *Proceedings of APPROX*, 2011, pp. 75–86.

[4] S. Srinivasagopalan, C. Busch, and S. Iyengar, "An oblivious spanning tree for single-sink buy-at-bulk in low doubling-dimension graphs," *IEEE Transactions on Computers*, vol. 61, no. 5, pp. 700–712, May 2012.

[5] A. Goel and D. Estrin, "Simultaneous optimization for concave costs: single sink aggregation or single source buy-at-bulk," in *Proceedings of ACM-SIAM SODA*, 2003, pp. 499–505.

[6] A. Goel and I. Post, "One tree suffices: A simultaneous o(1)-approximation for single-sink buy-at-bulk," in *Proceedings of IEEE FOCS*, 2010, pp. 593–600.

[7] ——, "An oblivious o(1)-approximation for single source buy-at-bulk," in *Proceedings of IEEE FOCS*, 2009, pp. 442–450.

[8] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "Tag: A tiny aggregation service for ad hoc sensor networks," in *OSDI*, 2002.

[9] S. Madden, R. Szewczyk, M. J. Franklin, and D. Culler, "Supporting aggregate queries over ad-hoc wireless sensor networks," in *Proceedings of IEEE WMCSA*, 2002.

[10] B. Krishnamachari, D. Estrin, and S. Wicker, "Modelling data-centric routing in wireless sensor networks," in *Proceedings of IEEE INFOCOM*, 2002.

[11] M. Elkin, Y. Emek, D. Spielman, and S. Teng, "Lower-stretch spanning trees," *special issue of SIAM Journal on Computing for STOC'05*, vol. 38, no. 2, pp. 608–628, 2008.

[12] I. Abraham and O. Neiman, "Using petal-decompositions to build a low stretch spanning tree," in *Proceedings of ACM STOC*, 2012.

[13] I. Abraham, Y. Bartal, and O. Neiman, "Nearly tight low stretch spanning trees," in *Proceedings of IEEE FOCS*, 2006.

[14] J. Fakcharoenphol, S. Rao, and K. Talwar, "A tight bound on approximating arbitrary metrics by tree metrics," in *Proceedings of ACM STOC*, 2003, pp. 448–455.

[15] M. Thorup, "Compact oracles for reachability and approximate distances in planar digraphs," *Journal of ACM*, vol. 51, no. 6, pp. 993–1024, 2004.

[16] I. Abraham and C. Gavoille, "Object location using path separators," in *Proceedings of ACM PODC*, 2006, pp. 188–197.

[17] P. Klein, S. A. Plotkin, and S. Rao, "Excluded minors, network decomposition, and multicommodity flow," in *Proceedings of ACM STOC*, 1993.

[18] C. Busch, C. Dutta, J. Radhakrishnan, R. Rajaraman, and S. Srinavasagopalan, "Split and join: Strong partitions and universal steiner trees for graphs," *CoRR*, vol. abs/1111.4766, 2011.

[19] M. T. Hajiaghayi, R. D. Kleinberg, and F. T. Leighton, "Improved lower and upper bounds for universal tsp in planar metrics," in *Proceedings of ACM-SIAM SODA*, 2006, pp. 649–658.

[20] I. Gorodezky, R. D. Kleinberg, D. B. Shmoys, and G. Spencer, "Improved lower bounds for the universal and a priori tsp," in *Proceedings of APPROX-RANDOM*, 2010, pp. 178–191.

[21] L. K. Platzman and I. J. J. Bartholdi, "Spacefilling curves and the planar travelling salesman problem," *Journal of the ACM*, vol. 36, no. 4, pp. 719–737, 1989.

[22] D. Bertsimas and M. Grigni, "On the space-filling curve heuristic for the euclidean traveling salesman problem," *Operations Research Letters*, vol. 8, pp. 241–244, 1989.

[23] H. Räcke, "Minimizing congestion in general networks," in *Proceedings of IEEE FOCS*, 2002, p. 4352.

[24] N. Alon, R. M. Karp, D. Peleg, and D. West, "A graphtheoretic game and its application to the k-server problem," *SIAM J. Comput.*, vol. 24, no. 1, pp. 78–100, 1995.

[25] D. Peleg, *Distributed Computing: A Locality-Sensitive Approach*. SIAM, 2000.

[26] B. Awerbuch and D. Peleg, "Sparse partitions," in *Proceedings of IEEE FOCS*, 1990, pp. 503–513.

[27] I. Abraham, C. Gavoille, D. Malkhi, and U. Wieder, "Strong-diameter decompositions of minor free graphs," in *Proceedings of ACM SPAA*, 2007.

[28] T. Birk, I. Keidar, L. Liss, A. Schuster, and R. Wolff, "Veracity radius - capturing the locality of distributed computations," in *Proceedings of ACM SIGACT-SIGOPS PODC*, 2006.

[29] C. Busch, R. LaFortune, and S. Tirthapura, "Improved sparse covers for graphs excluding a fixed minor," in *Proceedings of ACM PODC*, 2007, pp. 61–70.