

Mapping Networks via Parallel k -Hop Traceroute Queries

Ramtin Afshar ✉ 🏠 


University of California-Irvine, CA, USA

Michael T. Goodrich ✉ 🏠 

University of California-Irvine, CA, USA

Pedro Matias ✉ 

University of California-Irvine, CA, USA

Martha C. Osegueda ✉ 

University of California-Irvine, CA, USA

Abstract

For a source node, v , and target node, w , the `traceroute` command iteratively issues “ k -hop” queries, for $k = 1, 2, \dots, \delta(v, w)$, which return the name of the k th vertex on a shortest path from v to w , where $\delta(v, w)$ is the distance between v and w , that is, the number of edges in a shortest-path from v to w . The `traceroute` command is often used for network mapping applications, the study of the connectivity of networks, and it has been studied theoretically with respect to biases it introduces for network mapping when only a subset of nodes in the network can be the source of `traceroute` queries. In this paper, we provide efficient network mapping algorithms, that are based on k -hop `traceroute` queries. Our results include an algorithm that runs in a constant number of parallel rounds with a subquadratic number of queries under reasonable assumptions about the sampling coverage of the nodes that may issue k -hop `traceroute` queries. In addition, we introduce a number of new algorithmic techniques, including a high-probability parametric parallelization of a graph clustering technique of Thorup and Zwick, which may be of independent interest.

2012 ACM Subject Classification Theory of computation \rightarrow Parallel algorithms; Networks \rightarrow Network algorithms; Mathematics of computing \rightarrow Graph theory; Mathematics of computing \rightarrow Probabilistic algorithms

Keywords and phrases Network mapping, graph algorithms, parallel algorithms, distributed computing, query complexity, k -hop queries

Digital Object Identifier 10.4230/LIPIcs.STACS.2022.4

1 Introduction

Network mapping involves inferring the topology of a communication network, such as the Internet, from queries, e.g., see Figure 1 and [24, 49]. A prominent technique for network mapping is *active probing* using the Unix `traceroute` command to perform queries that reveal routing-path information, e.g., see [24, 32, 49]. We formulate the *network mapping* problem as follows. Suppose we are given access to a subset, $U \subseteq V$, of the vertices of a connected, undirected, unweighted graph, $G = (V, E)$, so that the *distance*, $\delta(u, v)$, between two vertices, u and v , in G is defined as the number of edges on a shortest path joining u and v in G . The n vertices in U are known, but the set of edges, E , is unknown. The subset U represents *vantage point* nodes from which we may issue the following type of queries:

- **k -hop(k, u, v)**: return the vertex, w , that is the k th vertex on a shortest path from u to v in G . If $k \geq \delta(u, v)$, then return v .

Note that for $u, v \in U$, **k -hop(k, u, v)** returns vertices in a single shortest path from u to v . Shortest paths in G are not necessarily unique, however. So, for example, if $\delta(u, v) = \delta(u, w) + \delta(w, v)$, it is not necessarily the case that **k -hop($\delta(u, w), u, v$)** = w . In



© Ramtin Afshar, Michael T. Goodrich, Pedro Matias, and Martha C. Osegueda; licensed under Creative Commons License CC-BY 4.0

39th International Symposium on Theoretical Aspects of Computer Science (STACS 2022).

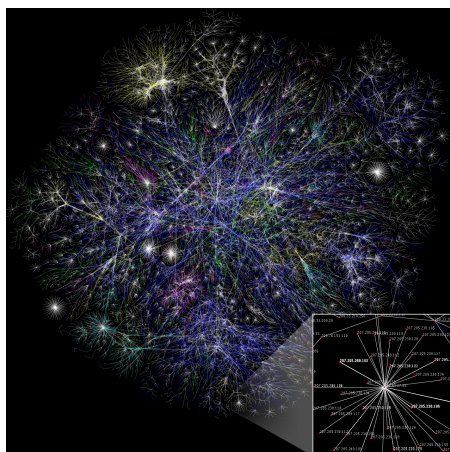
Editors: Petra Berenbrink and Benjamin Monmege; Article No. 4; pp. 4:1–4:21

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany





■ **Figure 1** Partial map of the Internet circa 2005. Image by The Opte Project, unchanged and licensed under the Creative Commons Attribution 2.5 Generic license.

the network mapping problem, we are interested in using k -hop queries to learn the edges of the *induced shortest-path graph*, $H = (U, \tilde{E})$, such that there is an edge $(u, v) \in \tilde{E}$, for $u, v \in U$, if and only if no k -hop(k, u, v) query would return a vertex $w \in U$ other than v , that is, k -hop(k, u, v) would return vertices of a shortest path from u to v that does not include any other vertex in U . Thus, H is a weighted, connected, undirected graph such that each edge (u, v) in H has weight $\delta(u, v)$.

Our motivation for focusing on k -hop queries is that they form the “inner loop” of how `traceroute` works by default. In particular, by default `traceroute` works by sending a series of packets in a network from a source, u , to a destination, v , with the packets having increasing time-to-live (TTL) values, up to an upper bound for the diameter, $\text{diam}(G)$, of G , which `traceroute` typically sets to 30 or 64 by default depending on the underlying operating system. The TTL field in a packet is decremented with each hop it traverses and when it reaches 1, then that node sends an ICMP message to the source address (with message including the node’s address), e.g., see [1, 2]. Thus, the `traceroute` tool can be viewed as first performing a k -hop($1, u, v$) query, then a k -hop($2, u, v$) query, and so on, until getting a response from the vertex v . In fact, one can use options with the `traceroute` command to issue a k -hop query directly, e.g., to find the 5th hop from a node to `example.com`, one could use the command, “`traceroute -m 5 -M 5 example.com`”.

Our formulation of the network mapping problem abstracts away certain system issues. In particular, we are implicitly assuming that messages in G are routed along shortest paths, which is a widely used setting assumed by the prior work [4, 20, 26]. An important system issue that we do not abstract away, however, is that only vertices in $U \subseteq V$ may issue queries. Indeed, there is some interesting prior work regarding the sampling biases introduced by only being able to issue queries from a subset, U , of the set of vertices, V , in G . For example, Achlioptas, Clauset, Kempe, and Moore [4] show that `traceroute` sampling¹ finds power-law degree distributions in both Δ -regular and Poisson-distributed random graphs, even though these underlying graphs do not themselves have power-law degree distributions, which is a statistical finding in experiments by Lakhina, Byers, Crovella, and Xie [37]. Maciej,

¹ *Traceroute sampling* samples the network graph as the union of paths that packets traverse in performing `traceroute` queries from a subset of the nodes in a network.

Markopoulou, and Patrick [36] study ways to correct for this bias when sampling large graphs. Further, Zhang, Kolaczyk, and Spencer [50] and Flaxman and Vera [26] study ways to correct for this bias for estimating degree distributions. Interestingly, Barrat, Alvarez-Hamelin, Dall’Asta, Vázquez, and Vespignani [13] provide an analysis that power laws still exist in the Internet graph in spite of the `traceroute` sampling bias, which these authors show is related to betweenness (see also [20]).

In spite of this interesting prior work concerning the sampling biases inherent in performing `traceroute` queries only from the nodes in the subset, U , we are not familiar with any prior work on efficient algorithms for solving the network mapping problem. We focus on two complexity measures for a network mapping algorithm, \mathcal{A} , in terms of $n = |U|$:

- $Q(n)$: the *query complexity* of \mathcal{A} . This is the total number of `kth-hop` queries issued. This complexity measure comes from learning theory (e.g., see [5, 18, 22, 43]) and complexity theory (where it is also known as “decision-tree complexity,” e.g., see [16, 48]).
- $R(n)$: the *round complexity* of \mathcal{A} . This is the number of rounds of querying performed by \mathcal{A} , where the queries issued in a round are given in a batch such that any query issued in a round may not depend on the response to any other query in that round (but each query may depend on results of queries from previous rounds).

Prior Related Work. As mentioned above, we are not aware of prior algorithmic work on network mapping. If we analyze the algorithm used in existing mapping systems that use active probing, this amounts to a brute-force quadratic algorithm implemented by cooperating nodes of the network, which perform a `traceroute` to every other known node in the network, e.g., see [23, 24, 33]. Viewed combinatorially, this algorithm has query complexity, $Q(n)$, that is $O(\text{diam}(G) \cdot n^2)$, and round complexity, $R(n)$, that is $O(\text{diam}(G))$, for `kth-hop` queries.

The network mapping problem is related to *graph reconstruction*, e.g., see [3, 6, 7, 9–12, 14, 15, 17, 18, 21, 27–30, 34, 35, 38, 40–42, 44, 46, 47]. In this problem, one is given a connected unweighted graph, $G = (V, E)$, for which V is known and goal is to discover E through queries, such as:

- `distance(u, v)`: return the distance, $\delta(u, v)$, between u to v in G .
- `shortest-path(u, v)`: return the vertices (in order) in a shortest path from u to v in G .

There is also work on other types of queries, including vertex-betweenness queries [3]; queries returning whether a given subset of vertices induce a given edge [9–12, 15, 17]; queries returning the number of edges induced by a given subset of vertices [18, 27–29]; queries returning *all* shortest paths from a given node to all other nodes [14, 42]; queries returning the distance between two leaves in a phylogenetic tree [6, 7, 30, 35, 40, 47]; and queries returning whether a given vertex is an ancestor of another given vertex in a rooted tree [6, 7, 46].

There are a number of important differences between the network mapping problem and graph reconstruction, however. Most significantly, the graph reconstruction problem assumes queries can be performed for any vertices in V , whereas in the network mapping problem we may only issue `kth-hop` queries for nodes in the subset $U \subseteq V$. In addition, even if we restrict the network mapping problem to the case where $U = V$, previous work on graph reconstruction has not considered `kth-hop` queries, which, as we mentioned above, form the “inner-loop” for how `traceroute` works and are distinct from `distance` and `shortest-path` queries. For example, it doesn’t seem possible to simulate a `kth-hop` query with fewer than $\Theta(n)$ `distance` queries, while a `distance` query can be simulated with $O(\log \text{diam}(G))$ `kth-hop` queries via binary search. Also, although it is trivial to simulate a `kth-hop` query with a single `shortest-path` query, it takes $\Theta(\text{diam}(G))$ `kth-hop` queries to simulate a single `shortest-path` query. Thus, `kth-hop` queries are strictly weaker than `shortest-path` queries while being better at capturing the true message complexity of the `traceroute` command.

Another difference between the network mapping problem and graph reconstruction is that previous work on graph reconstruction has mostly focused on how to sequentially reconstruct the graph, G , whereas the network mapping problem is inherently parallel, due to the motivation from mapping real-world networks, where each node is a computer. In terms of previous work on graph reconstruction in parallel, Mathieu and Zhou [38] recently provided a simple algorithm to reconstruct a connected, unweighted graph G , using an expected number of $\tilde{O}(N^{5/3})$ distance queries in 2 rounds.² They also show that their algorithm takes an expected number of $\tilde{O}(N)$ distance queries to reconstruct a random Δ -regular graphs.

The most relevant prior work on graph reconstruction, however, is by Kannan, Mathieu, and Zhou [34], who show how to reconstruct a connected, unweighted graph, G , using an expected number of $O(\Delta^3 N^{3/2} \log^2 N \log \log N)$ distance queries, or an expected number of $N^{1+O(\tau(N))}$ shortest-path queries, where $N = |V|$ and $\tau(N) = \sqrt{(\log \log N + \log \Delta) / \log N}$, which is $o(1)$ when Δ , the maximum degree of G , is $N^{o(1)}$. They also show that verifying a given set of edges can be done using $O(N^{1+O(\tau(N))})$ expected distance queries.

Our Results. A preliminary announcement of some of this paper’s results, using distance queries for graph reconstruction, where queries can be performed for any vertices in V , was presented in [8].

In Section 2, we introduce a new technique that may be of independent interest, where we provide a new parallel implementation of a well-known graph clustering technique of Thorup and Zwick [45] with round complexity of $O(1)$, while their original implementation implies an expected round complexity of $O(\log n)$. In doing so, we introduce a parameter that allows to trade off parallel time and cluster size. Moreover, we show that our complexity bounds hold with high probability,³ whereas Thorup and Zwick proved their complexity bounds only in expectation. In Section 3, we will use this new construction to compute a graph-theoretic Voronoi diagram in our network mapping algorithm. On the other hand, our graph clustering technique can be applied to other problems, such as that studied by Honiden, Houle, and Sommer [31] for balancing graph-theoretic Voronoi diagrams, to reduce the number of centers to $O(s)$ from $O(s \log n)$.

In Section 3, we provide the first non-trivial algorithmic results for the network mapping problem. Our query complexities and round complexities are characterized in terms of $n = |U|$ and some interesting parameters that capture the sampling coverage provided by the set U . For example, in addition to characterizing complexities in terms of Δ , the maximum degree of the graph, H , we introduce a distance coverage parameter, δ_{\max} , which is the maximum weight for an edge in H , and a nearby-vertices parameter, μ , which is an upper bound on the number of vertices within a distance of $2\delta_{\max}$ of any given vertex $v \in U$. As we show, these parameters are required for the sake of efficiency, for we show that without these parameters the network mapping problem has a quadratic query-complexity lower bound. For example, under reasonable assumptions regarding these parameters, we are the first to give a constant-round network-mapping algorithm with query complexity better than the trivial brute-force algorithm.

In Section 4, we introduce a greedy approach for network mapping that is based on parallel greedy approximate set cover, which allows us to achieve a near-quasilinear query complexity (when Δ is $n^{o(1)}$). As with a related sequential greedy graph reconstruction result of Kannan, Mathieu, and Zhou [34], our query and round complexity bounds are parameterized in terms

² The notation $\tilde{O}(f(N))$ is equivalent with $O(f(N) \cdot \text{polylog}(f(N)))$.

³ We say an event holds *with high probability* (w.h.p.) if it occurs with probability at least $1 - 1/n$.

of the best sequential query complexity for verifying the edges of a graph using distance queries (without knowing the exact value of this query complexity). Further, for small values of the parameters, δ_{\max} and Δ , our greedy approach uses a near-quasilinear number of *kth-hop* queries, which are strictly weaker than the *shortest-path* queries used by Kannan, Mathieu, and Zhou. We summarize our results in Table 1.

■ **Table 1** Our w.h.p. bounds for the network mapping problem, where ϵ denotes a fixed constant, $0 < \epsilon < 1/2$, $n = |U|$ and Δ , δ_{\max} , μ , and $\tau(\cdot)$ are as defined above.

$R(n)$	$Q(n)$
$O(1)$	$O(\delta_{\max} \mu n^{3/2+\epsilon})$
$O(\log n \cdot \log \text{diam}(G))$	$O(\mu n^{3/2} \log^{3/2} n \cdot \log \text{diam}(G))$
if $U \subset V$: $O(\Delta n)$	$\text{diam}(G) \cdot n^{1+O(\tau(n))}$
if $U = V$: $O(\Delta n \log n)$	$n^{1+O(\tau(n))}$

2 Parallel Graph Clustering

Thorup and Zwick [45] introduced a graph clustering technique in presenting a stretch⁴ 3 network routing scheme. We begin by describing our parallel graph clustering algorithm, which may be of independent interest, as it provides a parameterized parallel extension of the one by Thorup and Zwick [45]. Also, whereas Thorup and Zwick establish their bounds in expectation, we establish ours with high probability. In Section 3, we apply our parallel graph clustering algorithm in creating a graph-theoretic Voronoi diagram for our network mapping algorithm.

We begin with some review from Thorup and Zwick [45]. Let $G = (V, E)$ be a connected, undirected n -vertex graph, and let $\delta(u, v)$ denote the distance between vertices u and v in G . In this section, we allow G to be weighted, where $\delta(u, v)$ is the sum of weights on a shortest path (lowest weight path) from u to v , but in our algorithms for parallel network mapping, we assume G is unweighted, in which case $\delta(u, v)$ is the number of edges on a shortest path from u to v . For a subset $A \subseteq V$, let $\delta(A, v) = \min_{a \in A} \delta(a, v)$, and, for vertices $w, v \in V$, let $C_A(w)$ be the *cluster* of w and $B_A(v)$ be the *bunch* of v with respect to A , defined as follows:

$$C_A(w) = \{v \in V \mid \delta(w, v) < \delta(A, v)\} \quad \text{and} \quad B_A(v) = \{w \in V \mid \delta(w, v) < \delta(A, v)\}.$$

Note that if $w \in A$, then $C_A(w) = \emptyset$. Also, observe that bunches and clusters are “inverses” of each other, in that $v \in C_A(w)$ if and only if $w \in B_A(v)$. In addition, notice that clusters and bunches can only shrink as we add vertices to A ; that is, if $A' \subseteq A$, then $C_A(w) \subseteq C_{A'}(w)$ and $B_A(v) \subseteq B_{A'}(v)$, for all v and w in V .

Now, let $\beta \in [4, n)$, be a “parallelism” parameter and let $s \in [4 \ln n, n)$ be a “size” parameter. Define a subset, $A \subseteq V$, to be a set of (β, s) -balanced centers if $|C_A(w)| \leq \beta n/s$, for all $w \in V$. Thorup and Zwick [45] give a sequential algorithm for finding a set of $(4, s)$ -balanced centers of expected size $O(s \log n)$. In Algorithm 1, we give a parallel algorithm for finding a set of (β, s) -balanced centers of size $O(s \log_\beta n)$ in $O(\log_\beta n)$ rounds w.h.p. Thus, the parameter β allows one to trade off parallel time and cluster size.

⁴ *Routing Stretch* is the worst ratio between the length of a path on which a message is routed and the length of the shortest path in the network from the source to the destination.

4:6 Mapping Networks via Parallel kth-Hop Traceroute Queries

■ **Algorithm 1** parallel-centers(V, s, β).

```

1  $A \leftarrow \emptyset, W \leftarrow V$ 
2 while  $|W| > 0$  do
3    $A' \leftarrow \text{Sample}(W, s)$  // a random sample of expected size  $s$  (or  $W$  if  $s \geq |W|$ )
4    $A \leftarrow A \cup A'$ 
5   for  $w \in W$  do in parallel
6      $C_A(w) \leftarrow \{v \in V : \delta(w, v) < \delta(A, v)\}$ 
7    $W \leftarrow \{w \in W : |C_A(w)| > \beta n/s\}$ 
8 return  $A$ 

```

Our algorithm (Algorithm 1) takes a graph $G = (V, E)$ as input and initializes A , the eventual output of the algorithm, to be empty, and W , the set of nodes $v \in V$ where $|C_A(v)| > \beta n/s$, to be V . Then, we iteratively add $\text{Sample}(W, s)$ to A , and replace W with vertices $w \in W$ such that $|C_A(w)| > \beta n/s$, in parallel, where the function, $\text{Sample}(W, s)$, returns W if $|W| \leq s$ and, otherwise, returns a set of elements from W such that each element in W is selected independently at random with probability $s/|W|$. We continue in this way until $W = \emptyset$.

Since the size of a cluster, $|C_A(w)|$, does not increase as we add more vertices to A , the set A returned by our algorithm is a set of (β, s) -balanced centers. Also, the Sample function returns a sample of size at most $2s$ with probability at least $1 - e^{-s/3}$, which holds with high probability across all iterations when $s \geq 4 \ln n$, by a standard Chernoff bound, e.g., see [39, p. 69]. Incidentally, Thorup and Zwick use the same Sample function, but don't bound its maximum size as we do. This high-probability upper bound for the sample size is not sufficient to achieve a high-probability bound, however, for the entire parallel graph clustering algorithm.

To that end, we define a parameter, α , as follows:

$$\alpha = \begin{cases} 2 & \text{if } \beta \leq ((4/3)e)^4 \\ (4/3)e\beta^{1/2} & \text{otherwise} \end{cases}$$

where $e \approx 2.71828$ is Euler's number. This definition of α is made so that we may achieve high probability bounds for a range of β values.

Let W_i denote the set W at the beginning of iteration i , let A'_i denote the set A' that was added in iteration i , and let A_i denote the set A in this iteration, including the set, A'_i , i.e., $A_i = A_{i-1} \cup A'_i$, for $i = 1, 2, \dots$, where $A_0 = \emptyset$. Say that iteration i is "bad" if the following inequality holds:

$$\sum_{w \in W_i} |C_{A'_i}(w)| > \frac{\alpha n |W_i|}{s},$$

and that otherwise it is "good". Note that, since W_i is a given for iteration i , whether iteration i is good or bad depends only on A'_i .

► **Lemma 1** (Thorup-Zwick [45], Lemma 3.2). *Let $W \subseteq V$, let $1 \leq s \leq n$, and let $A' \leftarrow \text{Sample}(W, s)$. Then, for every $v \in V$, $E[|B_{A'}(v) \cap W|] \leq |W|/s$.*

This implies the following:

$$E \left[\sum_{w \in W_i} |C_{A'_i}(w)| \right] = E \left[\sum_{v \in V} |B_{A'_i}(v) \cap W_i| \right] \leq \frac{n |W_i|}{s}.$$

Therefore, by Markov's inequality, an iteration is bad with probability at most $1/\alpha$.

Let W_{i+1} denote the set of vertices, W , whose clusters have size at least $\beta n/s$ at the end of a good iteration i . As $W_{i+1} \subseteq W_i$, and $C_{A_i}(w) \subseteq C_{A'_i}(w)$, for all $w \in V$, in a good iteration we have:

$$\frac{\beta n |W_{i+1}|}{s} \leq \sum_{w \in W_i} |C_{A_i}(w)| \leq \sum_{w \in W_i} |C_{A'_i}(w)| \leq \frac{\alpha n |W_i|}{s};$$

hence, $|W_{i+1}| \leq (\alpha/\beta)|W_i|$ in a good iteration. Thus, the number of good iterations of our algorithm is $O(\log_{(\beta/\alpha)} n)$, which is $O(\log_\beta n)$ for either choice of α . Moreover, because an iteration is good independent of whether any other iteration is good or bad, we may use standard and non-standard Chernoff bounds to show that the number of bad iterations is also $O(\log_\beta n)$ w.h.p., for either value of α . (See Appendix A.1.) This gives us the following:

► **Theorem 2.** *Given an undirected, connected graph, $G = (V, E)$, we can find a set, A , of (β, s) -balanced centers of size $O(s \log_\beta n)$ in $O(\log_\beta n)$ parallel rounds w.h.p.*

For example, if $\beta = 4$, then A is constructed to have size $O(s \log n)$ in $O(\log n)$ rounds; if $\beta = n^\epsilon$, for constant $0 < \epsilon < 1/2$, then A is constructed to have size $O(s)$ in $O(1)$ rounds.

3 Our Fast Parallel Network Mapping Algorithms

In this section, we provide our fast parallel network mapping algorithms for a connected, undirected, unweighted network, $G = (V, E)$, given a subset $U \subseteq V$ from which we may perform k th-hop queries. We denote the size of U by n and the size of V by N . Let H be the graph induced by the shortest paths in G between pairs of vertices in U . That is, H has vertex set U and there is an edge (u, v) in H , for $u, v \in U$, if the shortest path between u and v in G determined by k th-hop queries contains no other vertex in U besides u and v . The weight of each edge (u, v) in H is the distance, $\delta(u, v)$, between u and v in G . The goal of network mapping is to determine the edges of H (which can then be used to easily determine the vertices in G in a shortest path corresponding to each edge (u, v) in H in a single round of $\delta(u, v)$ k th-hop queries). We assume we know the value of δ_{\max} , which is the weight of a maximum-weight edge in H . For example, if $U = V$, then $\delta_{\max} = 1$. In the worst case, δ_{\max} is equal to the diameter of G , but in real-world network mapping applications, δ_{\max} is likely to be a constant.

We perform all the queries needed in our parallel network mapping algorithms in a subroutine, $\text{Distances}(v, W)$, which determines the distance, $\delta(v, w)$, for a given $v \in U$ and every other $w \in W \subseteq U$. We describe two possible implementations for $\text{Distances}(v, W)$, which we choose between depending on our desired goals. In our first implementation, we perform a simple binary search using k th-hop(k, v, w) queries to determine $\delta(v, w)$, for each w . This requires $O(\log \text{diam}(G))$ rounds and a total of $O(|W| \log \text{diam}(G))$ k th-hop queries, and this implementation doesn't require any assumptions about W . Note that we assume that we know $\text{diam}(G)$, and if this is not the case, we can instead perform a doubling binary search with the same query complexity. In our second implementation, we perform δ_{\max} k th-hop(k, w, v) queries in parallel, for each w , for $k = 1$ to δ_{\max} . This implementation requires a single round of $O(\delta_{\max} |W|)$ k th-hop queries, and it requires that $v \in W$, and the nodes in W induce a connected subgraph of H that contains the shortest path in H from each w in W to v , and that we are only interested in finding the edges of this subgraph. This set of queries finds all the edges of a breadth-first search (BFS) tree, B_v , rooted at v , in the induced graph, H , since a shortest path from w to v is also a shortest path from v

■ **Algorithm 2** Our parallel querying algorithm, $\text{estimated-parallel-centers}(U, s, \beta)$, for finding a set of (β, s) -balanced centers A .

```

1  $A \leftarrow \emptyset, W \leftarrow U$ 
2  $T \leftarrow c_1(s/\beta) \log n$  //  $c_1$  is a constant set in the analysis
3 while  $|W| > 0$  do
4    $A' \leftarrow \text{Sample}(W, s)$ 
5    $A \leftarrow A \cup A'$ 
6    $R \leftarrow$  a random subset with  $v \in U$  chosen independently with probability  $T/n$ 
7   for each  $r \in R$  do in parallel
8      $\text{Distances}(r, U)$ 
9   for  $w \in W$  do in parallel
10     $S(w) \leftarrow \{v \in R : \delta(w, v) < \delta(A, v)\}$  //  $S(w) = C_A(w) \cap R$ 
11     $W \leftarrow \{w \in W : |S(w)| > 2\beta T/s\}$  // that is,  $|S(w)|(n/T) > 2\beta n/s$ 
12 return  $A$ 

```

to w , and a subpath of any shortest path is a shortest path for its endpoints. Thus, in this second implementation, we can determine $\delta(v, w)$, for each $w \in U$, from B_v , by summing the weights of the edges from v to w in B_v (which doesn't require any additional queries). This gives us the following lemma.

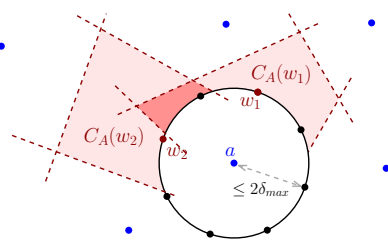
► **Lemma 3.** *Distances(v, W) can be implemented in $O(\log \text{diam}(G))$ rounds using a total of $O(|W| \log \text{diam}(G))$ k -hop queries. Alternatively, if $v \in W$, and the subgraph of H induced by W is connected and we are interested only in finding the edges of this subgraph, then Distances(v, W) can be implemented in 1 round with $O(\delta_{\max} |W|)$ k -hop queries.*

Let the cluster of vertex w with respect to centers A be $C_A(w) = \{v \in U \mid \delta(w, v) < \delta(A, v)\}$. The key idea of our parallel network mapping algorithm is to first find a set, A , of (β, s) -balanced centers, using our parallel algorithm from the previous section, and then use this set of centers to compute a graph-theoretic Voronoi diagram [25, 31] for G , from which we may efficiently then perform a brute-force querying step for each Voronoi region. This approach is similar in spirit to the one by Kannan, Mathieu, Zhou [34, Section 2], with some key important differences: i) the restriction of our queries to the vantage point $U \subseteq V$ and the parameters capturing sampling coverage of set U , ii) the usage of k -hop queries, and iii) our parallel graph clustering that allows us to trade off between round complexity and query complexity.

The initial center-finding step builds a set, A , of size $O(s \log_\beta n)$ such that each vertex in U has a cluster with respect to A of size at most $\beta n/s$. One of the challenges in implementing this algorithm efficiently in parallel using k -hop queries is that we need to determine cluster sizes for all vertices in U in each iteration, which would take too many queries to compute exactly. So, rather than compute such sizes exactly, we instead build a global random set, R , which we use to approximate the size of each cluster. We give the details in Algorithm 2.

► **Lemma 4.** *Our estimated-parallel-centers algorithm constructs a set, A , of $(3\beta, s)$ -balanced centers of size $O(s \log_\beta n)$. Suppose Distances(r, U) executes in $R(n)$ rounds and $Q(n)$ k -hop queries. Then estimated-parallel-centers algorithm executes in $O(R(n) \log_\beta n)$ rounds and $O(Q(n)(s/\beta) \log n \log_\beta n)$ k -hop queries, w.h.p.*

Proof. See Appendix A.2. ◀



■ **Figure 2** This figure represents a partial structure of our Voronoi Diagram. Blue vertices represent centers from A . The circle centered at $a \in A$ represents the vertices of distance at most $2\delta_{\max}$ from a . We use clusters of nearby vertices of a to discover boundary edges. For simplicity, we draw only two clusters for two arbitrary nodes $w_1, w_2 \in N_{2\delta_{\max}}(a)$.

Now that we have defined and analyzed the function $\text{estimated-parallel-centers}(U, s, \beta)$, let us next turn to our parallel algorithm for mapping a connected, undirected graph, $G = (V, E)$, given a subset, $U \subseteq V$, from which we can perform k th-hop queries. This algorithm takes as input the vertex set U , and outputs, \tilde{E} , the set of edges of the induced graph, H , defined by the vertex set U and the shortest paths in G returned by k th-hop queries.

Let $A \subseteq U$ be a set of *centers*, which in our network mapping algorithm will come from a call to our $\text{estimated-parallel-centers}(U, s, \beta)$ algorithm, but a graph-theoretic Voronoi diagram can be defined for any weighted graph and any set of centers. Given a center, $a \in A$, define the *Voronoi cell*, $\text{Vor}_A(a)$, for a in H as $\text{Vor}_A(a) = \{v \in U : \delta(a, v) \leq \delta(A \setminus \{a\}, v)\}$. The *graph-theoretic Voronoi diagram* for A in U consists of the union of Voronoi cells, $\text{Vor}_A(a)$, for each center, $a \in A$. We say that an edge $(v, w) \in \tilde{E}$ is an *interior* edge if $v, w \in \text{Vor}_A(a)$, for some center $a \in A$, and it is a *boundary* edge if $v \in \text{Vor}_A(a)$ and $w \in \text{Vor}_A(b)$, where $a \neq b$. If we were to perform a set of k th-hop queries for every pair of vertices in a Voronoi cell, then we are guaranteed to discover every internal edge in $\text{Vor}_A(a)$, but we will miss boundary edges going between two Voronoi cells. Thus, we need to “branch out” a little bit from the vertices of $\text{Vor}_A(a)$ in order to discover all the boundary edges. To facilitate this, for any center, $a \in A$, let $N_{2\delta_{\max}}(a)$ be the set of “nearby” vertices in H , that is, vertices that are within a distance of $2\delta_{\max}$ of a . Formally,

$$N_{2\delta_{\max}}(a) = \{v \in U : \delta(a, v) \leq 2\delta_{\max}\}.$$

We assume we know μ , the maximum size of $N_{2\delta_{\max}}(a)$, for any $a \in A$. Of course, $\mu < n$. The following lemma shows that it is sufficient to consider these nearby neighbors, for each center $a \in A$, in order to cover all the edges in H , including interior edges and boundary edges. (See also Figure 2.)

We give the details of our network mapping algorithm in Algorithm 3. Through a call to $\text{estimated-parallel-centers}(U, s, \beta)$, we find a set of $(O(\beta), s)$ -balanced centers, A . Next, we build a BFS tree from each vertex $a \in A$ to be able to identify nodes in $N_{2\delta_{\max}}(a)$. Then, our mapping algorithm, **map**, constructs graph-theoretic Voronoi diagram for the centers in A , and then “branches out” from each center $a \in A$ by considering the nodes in $N_{2\delta_{\max}}(a)$ and the clusters defined by nodes in $N_{2\delta_{\max}}(a)$. Finally, after having done this Voronoi decomposition, our algorithm performs exhaustive searches in each cluster in parallel. This part of the algorithm uses a method, $\text{Exhaustive-Query}(W)$, which finds all the edges of H between vertices in W by calling $\text{Distances}(v, W)$, for each $v \in W$.

The following lemmas establish the correctness and performance complexities for our network mapping algorithm.

■ **Algorithm 3** Parallel network mapping using kth-hop queries.

```

1 Function map( $U$ ):
2    $A \leftarrow$  estimated-parallel-centers( $U, s, \beta$ )
3   for each  $a \in A$  do in parallel
4     Distances( $a, U$ ) // gives us  $N_{2\delta_{\max}}(a)$  as well
5   for each  $a \in A$  do in parallel
6      $E_a \leftarrow$  Exhaustive-Query( $N_{2\delta_{\max}}(a)$ )
7     for  $b \in N_{2\delta_{\max}}(a)$  do in parallel
8       Distances( $b, U$ )
9        $C_A(b) \leftarrow \{v \in U : \delta(b, v) < \delta(A, v)\}$ 
10       $E_{a,b} \leftarrow$  Exhaustive-Query( $C_A(b)$ )
11  return  $\bigcup_{a \in A} (E_a \cup \bigcup_{b \in N_{2\delta_{\max}}(a)} E_{a,b})$ 

```

► **Lemma 5.** Let (u, v) be an edge in H . Then there exists a center, $a \in A$, such that u and v are both in $N_{2\delta_{\max}}(a)$ or both in $C_A(b)$, for some $b \in N_{2\delta_{\max}}(a)$.

Proof. Let (u, v) be an edge in H , and note that, by definition, $\delta(u, v) \leq \delta_{\max}$. Assume, without loss of generality, that $\delta(A, u) \leq \delta(A, v)$. Also, let a be a vertex in A such that $\delta(a, u) = \delta(A, u)$. If $\delta(a, u) \leq \delta_{\max}$, then both u and v are in $N_{2\delta_{\max}}(a)$, by the triangle inequality. So, suppose $\delta(a, u) > \delta_{\max}$. Let b be a vertex in U on a shortest path from a to u such that $\delta_{\max} < \delta(a, b) \leq 2\delta_{\max}$. Note that b must exist, since no edge in H has weight more than δ_{\max} (it is possible that $b = u$). Further, b is in $N_{2\delta_{\max}}(a)$ and $\delta(a, u) = \delta(a, b) + \delta(b, u)$. Also, $\delta(b, u) < \delta(a, u) = \delta(A, u)$; hence, u is in $C_A(b)$.

By the triangle inequality, and the above observations,

$$\begin{aligned}
\delta(b, v) &\leq \delta(b, u) + \delta(u, v) \\
&\leq \delta(b, u) + \delta_{\max} \\
&= \delta(a, u) - \delta(a, b) + \delta_{\max} \\
&< \delta(a, u) - \delta_{\max} + \delta_{\max} \\
&= \delta(a, u) \\
&= \delta(A, u).
\end{aligned}$$

Therefore, $\delta(b, v) < \delta(A, v)$; hence, v is also in $C_A(b)$. ◀

► **Lemma 6.** If Distances(v, W) executes in $R(|W|)$ rounds using $Q(|W|)$ kth-hop queries, then Algorithm 3 uses $O(Q(n)(s/\beta) \log n \log_{\beta} n + \mu(Q(n) + (\beta n/s)Q(\beta n/s))s \log_{\beta} n)$ queries in $O(R(n) \log_{\beta} n)$ rounds, w.h.p., where $\mu = \max_{a \in A} |N_{2\delta_{\max}}(a)|$.

Proof. By Lemma 4, estimated-parallel-centers(U, s, β) executes in $O(R(n) \log_{\beta} n)$ rounds and $O(Q(n)(s/\beta) \log n \log_{\beta} n)$ kth-hop queries, and returns a set of $(3\beta, s)$ -balanced centers of size $O(s \log_{\beta} n)$, w.h.p. The parallel Distances calls in line 4 thus executes in $O(R(n))$ rounds using $O(Q(n)s \log_{\beta} n)$ kth-hop queries, w.h.p., and the calls to Exhaustive-Query in line 6 execute in $O(R(\mu))$ rounds using a total of $O(\mu Q(\mu)s \log_{\beta} n)$ kth-hop queries, but these bounds are dominated by the Distances calls in line 8, all of which execute in $O(R(n))$ rounds using $O(\mu Q(n)s \log_{\beta} n)$ kth-hop queries, w.h.p. Finally, the calls to Exhaustive-Query in line 10 execute in $O(R(\beta n/s))$ rounds using $O(\mu(\beta n/s)Q(\beta n/s)s \log_{\beta} n)$ kth-hop queries, w.h.p. ◀

Plugging in our derived bounds for Distances, we get the following theorem.

► **Theorem 7.** *Given a connected graph, $G = (V, E)$, and subset, $U \subseteq V$, one can map the n -vertex induced shortest-path graph, H , with respect to G and U in $O(1)$ rounds using $O(\delta_{\max} \mu n^{3/2+\epsilon})$ k th-hop queries, for constant $0 < \epsilon < 1/2$, w.h.p. Alternatively, one can map H in $O(\log n \cdot \log \text{diam}(G))$ rounds using $O(\mu n^{3/2} \log^{3/2} n \cdot \log \text{diam}(G))$ queries, w.h.p.*

Proof. For the first result, set $\beta = n^\epsilon$ and $s = n^{1/2+\epsilon}$, and let us use the implementation of Distances that executes in 1 round using $O(\delta_{\max} n)$ k th-hop queries from Lemma 3. For the second result, set $\beta = 4$ and $s = (n/\log n)^{1/2}$ and let us use the implementation of Distances that executes in $O(\log \text{diam}(G))$ rounds using $O(n \log \text{diam}(G))$ k th-hop queries from Lemma 3. The bounds follow by Lemma 6. ◀

For example, depending on the values of δ_{\max} and μ , the above theorem establishes an improvement over the brute-force querying algorithm for solving the parallel network mapping problem in $O(1)$ rounds. The following theorem shows that bounding the parameters δ_{\max} and μ is needed in order to do better than a quadratic number of k th-hop queries.

► **Theorem 8.** *There is an infinite family of n -vertex graphs, G , such that mapping the induced shortest-path graph, H , for a set, U , of $O(n)$ vertices requires $\Omega(n^2)$ k th-hop queries, when μ is $\Theta(n)$ and δ_{\max} is $\Theta(\log n)$, even if G has maximum degree 3.*

Proof. Let G be the graph of a complete binary tree with n nodes and let U be the set of leaves of G . Thus, the distance in H between any node, v , in the left subtree of G , and a node, w , in the right subtree of G , is $2 \log n$. Thus, $\delta_{\max} = 2 \log n$ and μ is $\Theta(n)$. Now, let G' be G plus a single path in G of length $(2 \log n) - 1$ between two vertices, v and w , in U such that v is in the left subtree of G and w is in the right subtree of G . Thus, there is an edge with weight $(2 \log n) - 1$ joining v and w in the induced shortest-path graph, H' , for G' , and otherwise H' has the same edge set as H . But there are $\Omega(n^2)$ such possible pairs and the only way to discover the edge (v, w) in H is to perform a k th-hop(k, v, w) query. Any other type of k th-hop query cannot distinguish between H and H' . ◀

4 A Greedy Network Mapping Algorithm

Kannan, Methieu, and Zhou [34] introduce a proof technique that sequentially uses a verification algorithm for unweighted graphs as an oracle for issuing shortest-path queries in a greedy graph reconstruction algorithm. In this section, we show how to adapt this proof technique to a parallel setting and apply it to map the weighted graph, H . For example, our algorithm uses k th-hop queries and provides parallelism according to a parameter, $1 \leq p < n$.

Our greedy algorithm is based on performing steps of the classic greedy set cover algorithm in parallel batches. Recall that in this problem, one is given a collection of sets, S_1, S_2, \dots, S_m , whose union is the universe \mathcal{U} , and the goal is to find a smallest sub-collection of sets whose union is \mathcal{U} , that is, a sub-collection that covers \mathcal{U} . The greedy algorithm repeatedly chooses the set covering the maximum number of uncovered items in \mathcal{U} , and this results in a number of sets that is at most an $O(\log n)$ factor more than optimum [19].

Let $f(n, \Delta)$ be the query complexity of the best sequential algorithm for the problem of graph verification for any connected unweighted graph of n vertices and maximum degree Δ via distance queries, that is, for determining whether an unknown graph, $G = (V, E)$, is equal to a given graph, $\hat{G} = (V, \hat{E})$. For example, Kannan, Methieu, and Zhou [34] show that $f(n, \Delta)$ is $n^{1+O(\tau(n))}$, where $\tau(n) = \sqrt{(\log \log n + \log \Delta) / \log n}$. The function, $f(n, \Delta)$, is used only in our analysis, where we show that, given a parallelism parameter, $1 \leq p < n$, our parallel network mapping algorithm can be tuned to have the desired query complexity, $Q(n)$, and round complexity, $R(n)$.

4:12 Mapping Networks via Parallel kth-Hop Traceroute Queries

The distance queries in an unweighted graph verification algorithm perform two functions – confirming that edges in \hat{E} are actually in E and confirming that edges not in \hat{E} are also not in E , that is, confirming every $(u, v) \notin \hat{E}$ is not in E . To that latter end, let $\hat{\delta}_h(u, v)$ denote the *hop-count* (number of edges) distance between u and v based on the edges in \hat{E} , and let \hat{E}^c denote the set of non-edges in \hat{G} , that is, the set of pairs, (u, v) , such that $u \neq v$ and $(u, v) \notin \hat{E}$. Similarly, let E^c denote the set of non-edges in G . For any set of tentative edges, \hat{E} , define the following set for each pair of vertices, $(u, v) \in \hat{E}^c$: $S_{u,v}(\hat{E}) = \{(x, y) \in \hat{E}^c \mid \hat{\delta}_h(u, x) + \hat{\delta}_h(y, v) + 1 < \hat{\delta}_h(u, v)\}$.

Kannan, Methieu, and Zhou [34] prove the following two lemmas.

► **Lemma 9.** *Suppose $\hat{E} \subseteq E$. For any $(u, v) \in \hat{E}^c$, if $\delta_h(u, v) = \hat{\delta}_h(u, v)$, where $\delta_h(u, v)$ denotes the hop-count distance between u and v in G , then $S_{u,v}(\hat{E}) \subseteq E^c$, that is, each pair in $S_{u,v}(\hat{E})$ is a non-edge in G .*

► **Lemma 10.** *If a set of distance queries, T , verifies that every non-edge of \hat{G} is a non-edge of G , then $\cup_{(u,v) \in T} S_{u,v}(\hat{E}) = \hat{E}^c$.*

We present our parallel greedy algorithm for mapping $H = (U, \tilde{E})$ in $G = (V, E)$, for when $U \subset V$, that is, we incrementally build our tentative edge set $\hat{E} \subseteq \tilde{E}$:

1. We initialize a set of tentative edges, \hat{E} , to a spanning tree of H by calling `kth-hop`(k, v, u) from every vertex, $v \in U$, to an arbitrarily chosen vertex, $u \in U$, for $k = 1, \dots, \text{diam}(G)$, in parallel. We initialize a set of confirmed non-edges of H , $F \leftarrow \emptyset$. Note that we always maintain that $F \subseteq \hat{E}^c$. This requires 1 round of $O(\text{diam}(G)n)$ `kth-hop` queries.
2. We compute all the $S_{u,v}(\hat{E})$ sets, for pairs $(u, v) \in \hat{E}^c$, which requires no queries.
3. We perform p steps of the greedy set-cover algorithm applied to the sets, $S_{u,v}(\hat{E}) \setminus F$, with the goal of covering the remaining pairs, in $\hat{E}^c \setminus F$, in a greedy fashion, which also requires no queries. Let $\{(u_1, v_1), (u_2, v_2), \dots, (u_p, v_p)\}$ denote the vertex pairs for the $S_{u,v}(\hat{E})$ sets chosen by these greedy steps.
4. We perform `kth-hop`(k, u_i, v_i) queries, for $k = 1, \dots, \text{diam}(G)$, in parallel, to determine the actual hop-count distance, $\delta_h(u_i, v_i)$, between u_i and v_i in H , for each $i = 1, 2, \dots, p$ in parallel. This step requires $O(1)$ rounds of $O(p \cdot \text{diam}(G))$ `kth-hop` queries in total.
5. For each i such that $\delta_h(u_i, v_i) = \hat{\delta}_h(u_i, v_i)$, we add all the pairs in $S_{u_i, v_i}(\hat{E})$ to F . If $F = \hat{E}^c$, then we are done, by Lemma 10.
6. Otherwise, if $F \neq \hat{E}^c$ and $\delta_h(u_i, v_i) = \hat{\delta}_h(u_i, v_i)$, for all $i = 1, 2, \dots, p$, then we repeat the above process, performing another p steps of greedy set cover, looping back to Step 3.
7. If, on the other hand, $F \neq \hat{E}^c$ and $\delta_h(u_i, v_i) < \hat{\delta}_h(u_i, v_i)$, for some i , then there must be at least one edge on a shortest path from u_i to v_i that is in \tilde{E} and not yet in \hat{E} . In this case, we add all such edges (which were discovered when we performed the $\text{diam}(G)$ `kth-hop`(k, u_i, v_i) queries) to \hat{E} , and repeat the above greedy searching for this updated set, \hat{E} , of candidate edges, returning to Step 2.

This gives us the following result.

► **Theorem 11.** *Let $f(n, \Delta)$ be the query complexity of an optimal sequential algorithm for graph verification for any unweighted connected graph with n vertices and maximum degree Δ using distance queries. Then, for $1 \leq p < n$, our parallel network mapping algorithm has `kth-hop` query complexity, $Q(n) \in O((\Delta np + f(n, \Delta) \log n) \text{diam}(G))$ and round complexity, $R(n) \in O((\Delta n + (f(n, \Delta)/p) \log n))$, if $U \subset V$, or $Q(n) \in O((\Delta np + f(n, \Delta) \log n) \log n)$ and round complexity, $R(n) \in O((\Delta n + (f(n, \Delta)/p) \log n) \log n)$, if $U = V$.*

Proof. See Appendix A.3. ◀

Thus, setting p to be $n^{O(\tau(n))}$ gives us the following.

► **Corollary 12.** *One can solve the network mapping problem with query complexity, $Q(n)$, that is $\text{diam}(G) \cdot n^{1+O(\tau(n))}$ and round complexity, $R(n)$, that is $O(\Delta n)$, if $U \subset V$, or with $Q(n)$ that is $n^{1+O(\tau(n))}$ and round complexity, $R(n)$, that is $O(\Delta n \log n)$, if $U = V$.*

This query complexity is within an $n^{o(1)}$ factor of optimal when Δ is $n^{o(1)}$, by the following simple lower bound.

► **Theorem 13.** *Solving the network mapping problem for an n -vertex graph, G , with maximum degree, Δ , requires $\Omega(\Delta n)$ k th-hop queries, even if H has only n edges.*

Proof. Let H be a caterpillar (i.e., a tree where every leaf is at distance 1 from a vertex on a central path), such that every internal node has degree Δ . Choose any pair, u and v , of sibling leaves and connect them with an edge. The only way to discover the edge, (u, v) , is to perform a k th-hop(k, u, v) query, for $k \geq 1$. Thus, in expectation, any graph reconstruction algorithm must perform a query for over half of the pairs of siblings in H , that is, at least $\Omega((n/\Delta)\Delta^2) = \Omega(\Delta n)$ queries, in order to discover all the edges of H . ◀

5 Conclusion

We have given efficient algorithms for solving the network mapping problem in parallel. Such algorithms show the effectiveness of k th-hop queries, even though they are weaker than shortest-path queries. Our methods assume knowledge of δ_{\max} and μ , but this assumption can be relaxed at the expense of increasing the round complexity by an $O(\log n)$ factor, while keeping the query complexity unchanged, by using our algorithm as a blackbox to perform a doubling search for the values of these parameters. Our methods also assume k th-hop(k, u, v) remains same in the algorithm, which is a reasonable assumption in static routing. In our network mapping formulation, we abstracted away some system issues such that when the TTL field of a packet reaches 1, the node sends an ICMP message to the source address; however, in the real Internet, some nodes may have their ICMP responses switched off. Therefore, a direction to extend this work would be to design algorithms addressing such system issues.

We have also given new, parallel implementations for graph clustering, which provide tradeoffs between the number of center vertices and the sizes of clusters. Even for sequential algorithms, this result may prove useful for applications where minimizing the number of center points is a primary optimization goal. For instance, one can apply our construction to the problems studied by Honiden *et al.* [31] for balancing graph-theoretic Voronoi diagrams to shave a $O(\log n)$ factor of the number of centers. It seems likely, therefore, that this result will have other applications as well.

References

- 1 TRACEROUTE for Linux. <http://traceroute.sourceforge.net/>. Accessed: April 6, 2020.
- 2 *TRACEROUTE(8) Traceroute For Linux*, October 2006. Accessed: April 6, 2020. URL: <http://man7.org/linux/man-pages/man8/traceroute.8.html>.
- 3 Mikkel Abrahamsen, Greg Bodwin, Eva Rotenberg, and Morten Stöckel. Graph reconstruction with a betweenness oracle. In Nicolas Ollinger and Heribert Vollmer, editors, *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, volume 47 of *LIPICs*, pages 5:1–5:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICs.STACS.2016.5.

- 4 Dimitris Achlioptas, Aaron Clauset, David Kempe, and Cristopher Moore. On the bias of traceroute sampling: or, power-law degree distributions in regular graphs. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 694–703, 2005. doi:10.1145/1060590.1060693.
- 5 Peyman Afshari, Manindra Agrawal, Benjamin Doerr, Carola Doerr, Kasper Green Larsen, and Kurt Mehlhorn. The query complexity of finding a hidden permutation. In Andrej Brodnik, Alejandro López-Ortiz, Venkatesh Raman, and Alfredo Viola, editors, *Space-Efficient Data Structures, Streams, and Algorithms: Papers in Honor of J. Ian Munro on the Occasion of His 66th Birthday*, pages 1–11, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. doi:10.1007/978-3-642-40273-9_1.
- 6 Ramtin Afshar, Michael T. Goodrich, Pedro Matias, and Martha C. Osegueda. Reconstructing binary trees in parallel. In Christian Scheideler and Michael Spear, editors, *SPAA '20: 32nd ACM Symposium on Parallelism in Algorithms and Architectures, Virtual Event, USA, July 15-17, 2020*, pages 491–492. ACM, 2020. doi:10.1145/3350755.3400229.
- 7 Ramtin Afshar, Michael T. Goodrich, Pedro Matias, and Martha C. Osegueda. Reconstructing biological and digital phylogenetic trees in parallel. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 173 of *LIPICs*, pages 3:1–3:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.ESA.2020.3.
- 8 Ramtin Afshar, Michael T. Goodrich, Pedro Matias, and Martha C. Osegueda. Parallel network mapping algorithms. In Kunal Agrawal and Yossi Azar, editors, *SPAA '21: 33rd ACM Symposium on Parallelism in Algorithms and Architectures, Virtual Event, USA, 6-8 July, 2021*, pages 410–413. ACM, 2021. doi:10.1145/3409964.3461822.
- 9 Noga Alon and Vera Asodi. Learning a hidden subgraph. *SIAM J. Discrete Math.*, 18(4):697–712, 2005. doi:10.1137/S0895480103431071.
- 10 Noga Alon, Richard Beigel, Simon Kasif, Steven Rudich, and Benny Sudakov. Learning a hidden matching. *SIAM J. Comput.*, 33(2):487–501, 2004. doi:10.1137/S0097539702420139.
- 11 Dana Angluin and Jiang Chen. Learning a hidden hypergraph. *J. Mach. Learn. Res.*, 7:2215–2236, 2006. URL: <http://jmlr.org/papers/v7/angluin06a.html>.
- 12 Dana Angluin and Jiang Chen. Learning a hidden graph using $O(\log n)$ queries per edge. *J. Comput. Syst. Sci.*, 74(4):546–556, 2008. doi:10.1016/j.jcss.2007.06.006.
- 13 Alain Barrat, Ignacio Alvarez-Hamelin, Luca Dall’Asta, Alexei Vázquez, and Alessandro Vespignani. Sampling of networks with traceroute-like probes. *Complexus*, 3(1-3):83–96, 2006.
- 14 Zuzana Beerliova, Felix Eberhard, Thomas Erlebach, Alexander Hall, Michael Hoffmann, Matúš Mihalák, and L. Shankar Ram. Network discovery and verification. *IEEE Journal on Selected Areas in Communications*, 24(12):2168–2181, 2006. doi:10.1109/JSAC.2006.884015.
- 15 Richard Beigel, Noga Alon, Simon Kasif, Mehmet Serkan Apaydin, and Lance Fortnow. An optimal procedure for gap closing in whole genome shotgun sequencing. In Thomas Lengauer, editor, *Proceedings of the Fifth Annual International Conference on Computational Biology, RECOMB 2001, Montréal, Québec, Canada, April 22-25, 2001*, pages 22–30. ACM, 2001. doi:10.1145/369133.369152.
- 16 Anna Bernasconi, Carsten Damm, and Igor Shparlinski. Circuit and decision tree complexity of some number theoretic problems. *Information and Computation*, 168(2):113–124, 2001. doi:10.1006/inco.2000.3017.
- 17 Mathilde Bouvel, Vladimir Grebinski, and Gregory Kucherov. Combinatorial search on graphs motivated by bioinformatics applications: A brief survey. In Dieter Kratsch, editor, *Graph-Theoretic Concepts in Computer Science, 31st International Workshop, WG 2005, Metz, France, June 23-25, 2005, Revised Selected Papers*, volume 3787 of *Lecture Notes in Computer Science*, pages 16–27. Springer, 2005. doi:10.1007/11604686_2.
- 18 Sung-Soon Choi and Jeong Han Kim. Optimal query complexity bounds for finding graphs. *Artificial Intelligence*, 174(9):551–569, 2010. doi:10.1016/j.artint.2010.02.003.

- 19 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.
- 20 Luca Dall'Asta, J. Ignacio Alvarez-Hamelin, Alain Barrat, Alexei Vázquez, and Alessandro Vespignani. Exploring networks with traceroute-like probes: Theory and simulations. *Theor. Comput. Sci.*, 355(1):6–24, 2006. doi:10.1016/j.tcs.2005.12.009.
- 21 Luca Dall'Asta, Ignacio Alvarez-Hamelin, Alain Barrat, Alexei Vázquez, and Alessandro Vespignani. Exploring networks with traceroute-like probes: Theory and simulations. *Theoretical Computer Science*, 355(1):6–24, 2006. URL: <http://www.sciencedirect.com/science/article/pii/S0304397505009126>.
- 22 Shahar Dobzinski and Jan Vondrak. From query complexity to computational complexity. In *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing, STOC '12*, pages 1107–1116, New York, NY, USA, 2012. ACM. doi:10.1145/2213977.2214076.
- 23 Benoit Donnet. Internet topology discovery. In Ernst Biersack, Christian Callegari, and Maja Matijasevic, editors, *Data Traffic Monitoring and Analysis - From Measurement, Classification, and Anomaly Detection to Quality of Experience*, volume 7754 of *LNCS*, pages 44–81. Springer, 2013. doi:10.1007/978-3-642-36784-7_3.
- 24 Benoit Donnet and Timur Friedman. Internet topology discovery: A survey. *IEEE Communications Surveys and Tutorials*, 9(1-4):56–69, 2007. doi:10.1109/COMST.2007.4444750.
- 25 Martin Erwig. The graph Voronoi diagram with applications. *Networks*, 36(3):156–163, 2000. doi:10.1002/1097-0037(200010)36:3<156::AID-NET2>3.0.CO;2-L.
- 26 Abraham D. Flaxman and Juan Vera. Bias reduction in traceroute sampling - towards a more accurate map of the internet. In Anthony Bonato and Fan R. K. Chung, editors, *Algorithms and Models for the Web-Graph, 5th International Workshop, WAW 2007, San Diego, CA, USA, December 11-12, 2007, Proceedings*, volume 4863 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2007. doi:10.1007/978-3-540-77004-6_1.
- 27 Vladimir Grebinski. On the power of additive combinatorial search model. In Wen-Lian Hsu and Ming-Yang Kao, editors, *Computing and Combinatorics, 4th Annual International Conference, COCOON '98, Taipei, Taiwan, R.o.C., August 12-14, 1998, Proceedings*, volume 1449 of *Lecture Notes in Computer Science*, pages 194–203. Springer, 1998. doi:10.1007/3-540-68535-9_23.
- 28 Vladimir Grebinski and Gregory Kucherov. Reconstructing a hamiltonian cycle by querying the graph: Application to DNA physical mapping. *Discret. Appl. Math.*, 88(1-3):147–165, 1998. doi:10.1016/S0166-218X(98)00070-5.
- 29 Vladimir Grebinski and Gregory Kucherov. Optimal reconstruction of graphs under the additive model. *Algorithmica*, 28(1):104–124, 2000. doi:10.1007/s004530010033.
- 30 Jotun J Hein. An optimal algorithm to reconstruct trees from additive distance data. *Bulletin of mathematical biology*, 51(5):597–603, 1989.
- 31 S. Honiden, M. E. Houle, and C. Sommer. Balancing graph Voronoi diagrams. In *Sixth International Symposium on Voronoi Diagrams*, pages 183–191, 2009.
- 32 B. Huffaker, D. Plummer, D. Moore, and K. Claffy. Topology discovery by active probing. In *IEEE Symposium on Applications and the Internet (SAINT)*, pages 90–96, 2002.
- 33 Bradley Huffaker, Daniel Plummer, David Moore, and KC Claffy. Topology discovery by active probing. In *Proceedings 2002 Symposium on Applications and the Internet (SAINT) Workshops*, pages 90–96. IEEE, 2002.
- 34 Sampath Kannan, Claire Mathieu, and Hang Zhou. Graph reconstruction and verification. *ACM Trans. Algorithms*, 14(4):40:1–40:30, 2018. doi:10.1145/3199606.
- 35 Valerie King, Li Zhang, and Yunhong Zhou. On the complexity of distance-based evolutionary tree reconstruction. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 12-14, 2003, Baltimore, Maryland, USA.*, pages 444–453, 2003. URL: <http://dl.acm.org/citation.cfm?id=644108.644179>.
- 36 Maciej Kurant, Athina Markopoulou, and Patrick Thiran. Towards unbiased BFS sampling. *IEEE Journal on Selected Areas in Communications*, 29(9):1799–1809, 2011. doi:10.1109/JSAC.2011.111005.

- 37 A. Lakhina, J.W. Byers, M. Crovella, and P. Xie. Sampling biases in IP topology measurements. In *IEEE INFOCOM*, volume 1, pages 332–341, 2003. doi:10.1109/INFOCOM.2003.1208685.
- 38 Claire Mathieu and Hang Zhou. A simple algorithm for graph reconstruction. In Petra Mutzel, Rasmus Pagh, and Grzegorz Herman, editors, *29th Annual European Symposium on Algorithms, ESA 2021, September 6-8, 2021, Lisbon, Portugal (Virtual Conference)*, volume 204 of *LIPICs*, pages 68:1–68:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.ESA.2021.68.
- 39 Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2/e edition, 2017.
- 40 Lev Reyzin and Nikhil Srivastava. On the longest path algorithm for reconstructing trees from distance matrices. *Inf. Process. Lett.*, 101(3):98–100, 2007. doi:10.1016/j.ipl.2006.08.013.
- 41 Guozhen Rong, Wenjun Li, Yongjie Yang, and Jianxin Wang. Reconstruction and verification of chordal graphs with a distance oracle. *Theoretical Computer Science*, 859:48–56, 2021. doi:10.1016/j.tcs.2021.01.006.
- 42 Sandeep Sen and V. N. Muralidhara. The covert set-cover problem with application to network discovery. In Md. Saidur Rahman and Satoshi Fujita, editors, *WALCOM: Algorithms and Computation, 4th International Workshop, WALCOM 2010, Dhaka, Bangladesh, February 10-12, 2010. Proceedings*, volume 5942 of *Lecture Notes in Computer Science*, pages 228–239. Springer, 2010. doi:10.1007/978-3-642-11440-3_21.
- 43 G. Tardos. Query complexity, or why is it difficult to separate $NP^A \cap coNP^A$ from P^A by random oracles A ? *Combinatorica*, 9(4):385–392, December 1989. doi:10.1007/BF02125350.
- 44 Fabien Tarissan, Matthieu Latapy, and Christophe Prieur. Efficient measurement of complex networks using link queries. *CoRR*, abs/0904.3222, 2009. arXiv:0904.3222.
- 45 Mikkel Thorup and Uri Zwick. Compact routing schemes. In Arnold L. Rosenberg, editor, *13th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 1–10, 2001. doi:10.1145/378580.378581.
- 46 Zhaosen Wang and Jean Honorio. Reconstructing a bounded-degree directed tree using path queries. In *57th IEEE Allerton Conference on Communication, Control, and Computing*, 2019. See also arXiv:1606.05183.
- 47 Michael S Waterman, Temple F Smith, Mona Singh, and William A Beyer. Additive evolutionary trees. *Journal of theoretical Biology*, 64(2):199–213, 1977.
- 48 Andrew Chi-Chih Yao. Decision tree complexity and Betti numbers. In *Proceedings of the Twenty-sixth Annual ACM Symposium on Theory of Computing, STOC '94*, pages 615–624, New York, NY, USA, 1994. ACM. doi:10.1145/195058.195414.
- 49 X. Zhang and C. Phillips. A survey on selective routing topology inference through active probing. *IEEE Communications Surveys Tutorials*, 14(4):1129–1141, 2012.
- 50 Yaonan Zhang, Eric D. Kolaczyk, and Bruce D. Spencer. Estimating network degree distributions under sampling: An inverse problem, with applications to monitoring social media networks. *The Annals of Applied Statistics*, 9(1):166–199, 2015. doi:10.1214/14-AOAS800.

A Omitted Proofs

Here, we provide proofs that were omitted in the body of this paper.

A.1 Bounding the number of Bad Iterations for Parallel Graph Clustering

Recall that we chose α in our parallel graph clustering algorithm according to the formula

$$\alpha = \begin{cases} 2 & \text{if } \beta \leq ((4/3)e)^4 \\ (4/3)e\beta^{1/2} & \text{else,} \end{cases}$$

and we said an iteration is “bad” if $\sum_{w \in W_i} |C_{A'_i}(w)| > \frac{\alpha n |W_i|}{s}$ and otherwise it is “good.” We also noted, by Markov’s inequality, that an iteration is bad with probability at most $1/\alpha$. Further, note that $L = \log_{(\beta/\alpha)} n = O(\log_\beta n)$ is the maximum number of good iterations, for either choice for α . We wish to show that the number of bad iterations is $O(L)$ w.h.p.

Since W_i is a given for iteration i , whether iteration i is good or bad depends only on A'_i ; therefore, an iteration is good independent of whether any other iteration is good or bad, so, for the sake of analysis, consider a set of $c_0 L$ iterations (i.e., padding out with “dummy” iterations if necessary) where $c_0 \geq 4$ is a constant chosen below and each iteration is bad independently with probability $1/\alpha$. Let X denote the number of bad iterations in this set. So $E[X] = c_0 L/\alpha$; hence, the probability that over $3/4$ of our iterations are bad can be bounded as $p = \Pr(X > (3/4)c_0 L) = \Pr(X > (3/4)\alpha \cdot E[X])$. Thus, at least L of our iterations are good with probability at least $1 - p$.

Case 1: $\alpha = 2$. In this case, β is $O(1)$; hence, L is $\Theta(\log_\beta n) = \Theta(\log n)$, since $\beta \geq 4$.

Further, $\Pr(X > (3/4)\alpha \cdot E[X]) = \Pr(X > (3/2) \cdot E[X])$, and, by a standard Chernoff bound,⁵ e.g., see [39, p. 69],

$$\Pr(X > (3/2) \cdot E[X]) \leq e^{-E[X]/12} = e^{-c_0 L/24}.$$

Thus, choosing c_0 so that $c_0 L/24 \geq 2 \ln n$, we will have more than $(3/4)c_0 L$ bad iterations with probability at most $1/n^2$.

Case 2: $\alpha = (4/3)e\beta^{1/2}$. In this case, $(\alpha/\beta) \leq \beta^{-1/4}$; hence, L is $O(\log_\beta n)$. Further, we have that $\Pr(X > (3/4)c_0 L) = \Pr(X > (3/4)\alpha \cdot E[X]) = \Pr(X > e\beta^{1/2} \cdot E[X])$, and, by a non-standard Chernoff bound,⁶ e.g., see [39, p. 70],

$$\Pr(X > e\beta^{1/2} \cdot E[X]) \leq \left(\frac{e}{e\beta^{1/2}}\right)^{(3/4)c_0 L} = \beta^{-(3/8)c_0 L}.$$

Thus, by choosing c_0 so that $(3/8)c_0 L \geq 2 \log_\beta n$, we will have more than $(3/4)c_0 L$ bad iterations with probability at most $1/n^2$.

Therefore, we have the following.

► **Lemma 14.** *The number of good and bad iterations in Algorithm 1 is $O(\log_\beta n)$ w.h.p.*

A.2 The Complexity of the estimated-parallel-centers Algorithm

Recall that the estimated-parallel-centers algorithm uses a global random sample set, R , for estimating cluster sizes, where R is a random subset of U of size $T = c_1(s/\beta) \log n$. Recall that, for each vertex $w \in W$, we defined $S(w)$ such that $S(w) = R \cap C_A(w)$. Thus, $E[|S(w)|] = |C_A(w)|(T/n)$. We are interested in showing that w.h.p. this sample of $C_A(w)$ is giving neither an over-estimate nor an under-estimate for the size of $C_A(w)$, which we define respectively as follows:

- *Over-estimate event:* $|C_A(w)| \leq \beta n/s$, but $|S(w)| > 2\beta T/s$. In this case, we would be including w in W even though its cluster size is sufficiently small.
- *Under-estimate event:* $|C_A(w)| > 3\beta n/s$, but $|S(w)| \leq 2\beta T/s$. In this case, we would be excluding w from W even though its cluster size is big.

Let us consider each of these types of events in turn.

⁵ $\Pr(X \geq (1 + \delta) \cdot E[X]) \leq e^{-E[X] \cdot \delta^2/3}$, for $0 < \delta \leq 1$.

⁶ $\Pr(X \geq (1 + \delta) \cdot E[X]) \leq (e/(1 + \delta))^{(1 + \delta) \cdot E[X]}$.

4:18 Mapping Networks via Parallel k th-Hop Traceroute Queries

Over-estimate event. We wish to bound the probability that $|C_A(w)| \leq \beta n/s$ but $|S(w)| > 2\beta T/s$, where $T = c_1(s/\beta) \log n$. Let X denote the sum of $|C_A(w)|$ indicator random variables for counting the members of $C_A(w) \cap R$, i.e., where each variable is 1 independently with probability T/n . Thus, $E[X] = E[|S(w)|] = |C_A(w)|(T/n)$. So

$$\begin{aligned} \Pr(|S(w)| > 2\beta T/s) &= \Pr(X > 2\beta T/s) = \Pr\left(X > \frac{2\beta n}{s|C_A(w)|} \cdot E[X]\right) \\ &= \Pr(X > (1 + \delta) \cdot E[X]), \end{aligned}$$

where

$$\delta = \left(\frac{2\beta n}{s|C_A(w)|} - 1\right) > 1.$$

In addition,

$$\begin{aligned} \delta \cdot E[X] &= \left(\frac{2\beta n}{s|C_A(w)|} - 1\right) \cdot \frac{|C_A(w)|T}{n} = \frac{2\beta T}{s} - \frac{|C_A(w)|T}{n} \\ &\geq \frac{2\beta T}{s} - \frac{\beta T}{s} = \frac{\beta T}{s} = c_1 \log n. \end{aligned}$$

Thus, by a standard Chernoff bound,⁷ and the fact that $\delta > 1$,

$$\Pr(X \geq (1 + \delta) \cdot E[X]) \leq e^{-\delta^2 \cdot E[X]/(2+\delta)} \leq e^{-\delta \cdot E[X]/3} \leq e^{-(c_1 \log n)/3} \leq \frac{1}{n^3},$$

for $c_1 \geq 9 \ln 2 \approx 6.24$.

Under-estimate event. We wish to bound the probability that $|C_A(w)| > 3\beta n/s$ but $|S(w)| \leq 2\beta T/s$, where $T = c_1(s/\beta) \log n$. Let X denote the sum of $|C_A(w)|$ indicator random variables for counting the members of $C_A(w) \cap R$, i.e., where each variable is 1 independently with probability T/n . Thus, $E[X] = E[|S(w)|] = |C_A(w)|(T/n) > 3c_1 \log n$. So

$$\begin{aligned} \Pr(|S(w)| \leq 2\beta T/s) &= \Pr(X \leq 2\beta T/s) \\ &= \Pr\left(X \leq \frac{2\beta n}{s|C_A(w)|} \cdot E[X]\right) \leq \Pr(X \leq (2/3) \cdot E[X]). \end{aligned}$$

Thus, by a standard Chernoff bound,⁸ e.g., see [39, p. 71],

$$\Pr(|S(w)| \leq 2\beta T/s) \leq e^{-(3c_1 \log n)/18} \leq \frac{1}{n^3},$$

when $c_1 \geq 18 \ln 2 \approx 12.48$.

Of course, R is the same random sampling set for all our samples, $S(w)$, for $w \in W$. Nevertheless, by a union bound, the above analysis shows that R causes an over-estimate event or an under-estimate event, for some $S(w)$, with probability at most $1/n^2$.

By the bound on over-estimate events, we have shown that w.h.p. every cluster with size over $\beta n/s$ is included in W in any given iteration of our estimated-parallel-centers algorithm. In addition, by the bound on under-estimate events, we have shown that w.h.p. every vertex,

⁷ $\Pr(X \geq (1 + \delta) \cdot E[X]) \leq e^{-\delta^2 \cdot E[X]/(2+\delta)}$, for $\delta > 0$, e.g., see https://en.wikipedia.org/wiki/Chernoff_bound.

⁸ $\Pr(X \leq (1 - \delta) \cdot E[X]) \leq e^{-\delta^2 \cdot E[X]/2}$, for $0 < \delta < 1$.

w , that we exclude from W has a cluster size of at most $3\beta n/s$. Thus, using essentially the same analysis as we gave for the proofs of Theorem 2 and Lemma 14, and noting that each iteration of our estimated-parallel-centers algorithm has round complexity $O(R(n))$ and query complexity $O(Q(n)(s/\beta) \log n)$, where $R(n)$ and $Q(n)$ are the respective round and query complexities for the Distances algorithm, we have the following.

► **Lemma 15** (Lemma 4). *Our estimated-parallel-centers algorithm constructs a set, A , of $(3\beta, s)$ -balanced centers of size $O(s \log_\beta n)$. Suppose $\text{Distances}(r, U)$ executes in $R(n)$ rounds and $Q(n)$ k th-hop queries. Then estimated-parallel-centers algorithm executes in $O(R(n) \log_\beta n)$ rounds and $O(Q(n)(s/\beta) \log n \log_\beta n)$ k th-hop queries, w.h.p.*

A.3 Greedy Algorithm Proofs

Our description of our greedy algorithm given above in the body of our paper was for the case when $U \subset V$. For the case when $U = V$, we modify our algorithm to be the following (note that in this case, hop-count distance and graph distance are the same):

1. We initialize a set of tentative edges, \hat{E} , to a spanning tree of H by calling $\text{kth-hop}(1, v, u)$ from every vertex, $v \in U$, to an arbitrarily chosen vertex, $u \in U$. We initialize a set of confirmed non-edges, $F \leftarrow \emptyset$. This requires 1 round of $O(n)$ k th-hop queries.
2. We compute all the $S_{u,v}(\hat{E})$ sets, for pairs $(u, v) \in \hat{E}^c$, which requires no queries.
3. We perform p steps of the greedy set-cover algorithm applied to the sets, $S_{u,v}(\hat{E}) \setminus F$, with the goal of covering the remaining pairs, in $\hat{E}^c \setminus F$, in a greedy fashion, which also requires no queries. Let $\{(u_1, v_1), (u_2, v_2), \dots, (u_p, v_p)\}$ denote the vertex pairs for the $S_{u,v}(\hat{E})$ sets chosen by these greedy steps.
4. We perform a binary search using k th-hop queries to determine the actual distance, $\delta(u_i, v_i)$, between u_i and v_i in H , for each $i = 1, 2, \dots, p$ in parallel. This step requires $O(\log n)$ rounds of $O(p \log n)$ k th-hop queries in total.
5. For each i such that $\delta(u_i, v_i) = \hat{\delta}(u_i, v_i)$, we add all the pairs in $S_{u_i, v_i}(\hat{E})$ to F . If $F = \hat{E}^c$, then we are done, by Lemma 10.
6. Otherwise, if $F \neq \hat{E}^c$ and $\delta(u_i, v_i) = \hat{\delta}(u_i, v_i)$, for all $i = 1, 2, \dots, p$, then we repeat the above process, performing another p steps of greedy set cover, repeating a loop returning to Step 3.
7. If, on the other hand, $F \neq \hat{E}^c$ and $\delta(u_i, v_i) < \hat{\delta}(u_i, v_i)$, for some i , then there must be at least one edge on a shortest path from u_i to v_i that is in E and not yet in \hat{E} . In this case, we perform a binary search, described below, to find at least one such an edge, add all such edges to \hat{E} , and repeat the above greedy searching for this updated set, \hat{E} , of candidate edges, returning to Step 2. This step requires $O(\log n)$ rounds of at most $O(p \log n)$ k th-hop queries in total.

Before we give our analysis, let us describe the details for the binary search to find an undiscovered edge when $\delta(u_i, v_i) < \hat{\delta}(u_i, v_i)$, for some i . We begin with a query, $\text{kth-hop}(k, u_i, v_i)$, where $k = \lfloor \delta(u_i, v_i)/2 \rfloor$, and let w denote the returned vertex. So, $\delta(u_i, w) = k$ and $\delta(w, v_i) = \delta(u_i, v_i) - k$. Since $\delta(u_i, v_i) < \hat{\delta}(u_i, v_i)$, we know that $\delta(u_i, w) < \hat{\delta}(u_i, w)$ or $\delta(w, v_i) < \hat{\delta}(w, v_i)$. Thus, we recursively search for one of these until we discover a new edge not in \hat{E} , which must exist, since $\delta(u_i, v_i) < \hat{\delta}(u_i, v_i)$.

► **Theorem 16** (Theorem 11). *Let $f(n, \Delta)$ be the query complexity of an optimal sequential algorithm for graph verification for any unweighted connected graph with n vertices and maximum degree Δ using distance queries. Then, for $1 \leq p < n$, our parallel network mapping algorithm has k th-hop query complexity, $Q(n) \in O((\Delta np + f(n, \Delta) \log n) \text{diam}(G))$*

4:20 Mapping Networks via Parallel k -Hop Traceroute Queries

and round complexity, $R(n) \in O((\Delta n + (f(n, \Delta)/p) \log n))$, if $U \subset V$, or $Q(n) \in O((\Delta n p + f(n, \Delta) \log n) \log n)$ and round complexity, $R(n) \in O((\Delta n + (f(n, \Delta)/p) \log n) \log n)$, if $U = V$.

Proof. Building a spanning tree of H is a one-time expense taking $O(n \cdot \text{diam}(G))$ k -hop queries and a round complexity of $O(1)$ (step 1), for the case when $U \subset V$, or $O(n)$ queries with a round complexity of $O(1)$ (step 1), for the case when $U = V$. Each iteration of our greedy algorithm takes $O(p \cdot \text{diam}(G))$ k -hop queries with a round complexity of $O(1)$ (step 4), for the case when $U \subset V$, or $O(p \log n)$ k -hop queries with a round complexity of $O(\log n)$ (step 4 and step 7), for the case when $U = V$.

In the case when $\delta_h(u_i, v_i) < \hat{\delta}_h(u_i, v_i)$, for some $i \in [1, p]$, we discover at least one new edge – let us charge the queries for this iteration to this edge. Thus, the total number of k -hop queries due to this case is $O(\Delta n p \cdot \text{diam}(G))$, with $O(\Delta n)$ rounds, for the $U \subset V$ case, or $O(\Delta n p \log n)$, with $O(\Delta n \log n)$ rounds, for the $U = V$ case. So, let us consider the case when $\delta_h(u_i, v_i) = \hat{\delta}_h(u_i, v_i)$, for all $i \in [1, p]$, which we call a “completely-greedy” iteration. We will provide an upper bound for the number of such iterations. Recall that in step 3, for the case when $U \subset V$ (similarly in step 3, for the case when $U = V$) we performed p steps of greedy set-cover algorithm applied to the sets, $S_{u,v}(\hat{E}) \setminus F$, with the goal of covering the remaining pairs, in $\hat{E}^c \setminus F$ without additional queries. Let F_i denote the set of (x, y) pairs covered by the i -th step of this greedy set-cover algorithm, for $i = 1, 2, \dots, p$. Thus,

$$|F_1| \geq |F_2| \geq \dots \geq |F_p|,$$

and at the moment we chose the subset F_i it was the largest subset covering the uncovered pairs in $\mathcal{U}_i = \hat{E}^c \setminus (\bigcup_{j=1}^{i-1} F_j \cup F)$. The optimal sequential graph verification algorithm performs $f(n, \Delta)$ distance queries and confirms all the pairs in \hat{E}^c . Thus, in particular, this optimal algorithm must perform queries that cover \mathcal{U}_i as a part of its $f(n, \Delta)$ queries; hence, because F_i is the subset for a distance query that covers the largest number of pairs in \mathcal{U}_i , and the average number of pairs in \mathcal{U}_i covered by any distance query of the optimal algorithm is at least $|\mathcal{U}_i|/f(n, \Delta)$, we have that

$$|F_i| \geq \frac{|\mathcal{U}_i|}{f(n, \Delta)}.$$

Thus, in any iteration of our algorithm, since we perform p greedy steps, the size of the remaining pairs in $\hat{E}^c \setminus F$ is reduced by a multiplicative factor of

$$\left(1 - \frac{1}{f(n, \Delta)}\right)^p \leq e^{-p/f(n, \Delta)}.$$

Therefore, since $\hat{E}^c \leq n(n-1)$ and by the end of our algorithm we cover every pair in \hat{E}^c , the total number of completely-greedy iterations, g , can be bounded above by the smallest value of g such that

$$e^{-(p/f(n, \Delta))g} < n^{-2};$$

therefore, the total number of completely-greedy iterations, g , is at most $O((f(n, \Delta)/p) \log n)$. Note that the set \hat{E}^c is potentially growing during our algorithm, with completely-greedy iterations possibly interspersed with iterations that discover new edges in \hat{E} . Nevertheless, the above analysis still holds, because (1) the function, $f(n, \Delta)$ is a uniform bound for any connected graph with n nodes and maximum degree Δ , and (2) each time we (greedily)

confirm that $\hat{\delta}(u, v) = \delta(u, v)$ for a set, $S_{u,v}(\hat{E})$, all the pairs in $S_{u,v}(\hat{E})$ are, in fact, non-edges in \tilde{E}^c . The claimed complexity bounds follow then, since each completely-greedy iteration requires $O(p \cdot \text{diam}(G))$ k th-hop queries with round complexity $O(1)$, for the $U \subset V$ case, or $O(p \log n)$ k th-hop queries with round complexity $O(\log n)$, for the $U = V$ case. ◀