Graph Reconstruction and Verification

SAMPATH KANNAN, University of Pennsylvania

CLAIRE MATHIEU, École Normale Supérieure

HANG ZHOU, École Polytechnique

How efficiently can we find an unknown graph using distance or shortest path queries between its vertices? We assume that the unknown graph G is connected, unweighted, and has bounded degree. In the *reconstruction* problem, the goal is to find the graph G . In the *verification* problem, we are given a hypothetical graph $\hat{G}$ and want to check whether G is equal to $\hat{G}$.

We provide a randomized algorithm for reconstruction using $\tilde{O}(n^{3/2})$ distance queries, based on Voronoi cell decomposition. Next, we analyze natural greedy algorithms for reconstruction using a shortest path oracle and also for verification using either oracle, and show that their query complexity is $n^{1+o(1)}$. We further improve the query complexity when the graph is chordal or outerplanar. Finally, we show some lower bounds, and consider an approximate version of the reconstruction problem.

CCS Concepts: • **Networks** → *Network algorithms*; • **Theory of computation** → *Graph algorithms analysis*; *Data structures design and analysis*;

Additional Key Words and Phrases: Reconstruction, verification, network tomography

ACM Reference format:

Sampath Kannan, Claire Mathieu, and Hang Zhou. 2018. Graph Reconstruction and Verification. *ACM Trans. Algorithms* 14, 4, Article 40 (August 2018), 30 pages.

<https://doi.org/10.1145/3199606>

1 INTRODUCTION

How efficiently can we find an unknown graph using distance or shortest path queries between its vertices? This is a natural theoretical question from the standpoint of recovery of hidden information. This question is related to the *reconstruction* of internet networks. Discovering the topology of the internet is a crucial step for building accurate network models and designing efficient algorithms for internet applications. Yet, this topology can be extremely difficult to find, due to the dynamic structure of the network and to the lack of centralized control. Network reconstruction has been studied extensively [2, 3, 8, 19]. Sometimes we have some idea of what the network should be like, based perhaps on its state at some time in the past, and we want to check whether our

This is the full version of the extended abstracts that appear in Refs. [12, 14].

S. Kannan was partially supported by NSF Grant NRI 1317788. C. Mathieu was partially supported by the French *Agence Nationale de la Recherche* under Reference ANR-12-BS02-005 (RDAM project). H. Zhou was partially supported by the Lise Meitner Award Fellowship.

Authors' addresses: S. Kannan, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, USA; email: kannan@cis.upenn.edu; C. Mathieu, Département d'Informatique, École Normale Supérieure, Paris, France; email: cmathieu@di.ens.fr; H. Zhou, Computer Science department, École Polytechnique, Palaiseau, France, hzhou@lix.polytechnique.fr.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2018 ACM 1549-6325/2018/08-ART40 \$15.00

<https://doi.org/10.1145/3199606>

image of the network is correct. This is network *verification* and has received attention recently [3, 5]. This is an important task for routing, error detection, or ensuring service-level agreement compliance and the like. For example, internet service providers offer services that require quality of service guarantees, such as voice-over IP services, and thus need to check regularly whether the networks are correct.

The topology of internet networks can be investigated at the router level, where the set of routers and their physical connections are the vertices and edges of a graph, respectively. Traditionally, we use tools such as *traceroute* and *mtrace* to infer the network topology. These tools generate path information between a pair of vertices. It is a common and reasonably accurate assumption that the generated path is shortest, i.e., minimizes the hop distance between that pair. In our first theoretical model, we assume that we can query any pair of vertices and get in return a shortest path between them in the graph. Sometimes routers block *traceroute* and *mtrace* requests (e.g., due to privacy and security concerns). In this case, inference of topology can only rely on delay information. In our second theoretical model, we assume that we can query any pair of vertices and get in return the hop distance between them in the graph.

1.1 The Problem

Let $G = (V, E)$ be a hidden graph that is *connected*, *undirected*, and *unweighted*, where $|V| = n$. We consider two query oracles. A *shortest path oracle* receives a pair $(u, v) \in V^2$ and returns a shortest path between u and v .¹ A *distance oracle* receives a pair $(u, v) \in V^2$ and returns the number of edges on a shortest path between u and v .

In the *graph reconstruction* problem, we are given the vertex set V and have access to either a distance oracle or a shortest path oracle. The goal is to find every edge in E .

In the *graph verification* problem, again we are given V and have access to either oracle. In addition, we are given a connected, undirected, and unweighted graph $\hat{G} = (V, \hat{E})$. The goal is to check whether $\hat{G}$ is correct, that is, whether $\hat{G} = G$.

The efficiency of an algorithm is measured by its *query complexity*,² i.e., the number of queries to an oracle. All our algorithms run in polynomial time and space. We note that $O(n^2)$ queries are enough for reconstruction (and *ergo* for verification), using either a distance oracle or a shortest path oracle: we only need to query every pair of vertices, and return the pairs (u, v) whose u -to- v distance is 1. We call this reconstruction algorithm EXHAUSTIVE-QUERY.

Focusing on bounded degree graphs,³ it is not hard to see that $\Omega(n)$ distance or shortest path queries are required for both reconstruction and verification. The central question in this line of work is therefore: Is the query complexity linear, quadratic, or somewhere in between? We show that the query complexity for reconstruction using a distance oracle is subquadratic: $\tilde{O}(n^{3/2})$, and that the query complexity for reconstruction using a shortest path oracle or for verification using either oracle is near-linear: $n^{1+o(1)}$. We leave open the following question: Is there a reconstruction algorithm using a near-linear number of queries to a distance oracle?

1.2 Related Work

The inference of the topology of a network or a graph using queries that reveal partial information has been studied extensively in different contexts, independently stemming from a number of applications.

¹If there are several shortest paths between u and v , the oracle returns an arbitrary one.

²Expected query complexity in the case of randomized algorithms.

³Without the assumption that the graph has bounded degree, both reconstruction and verification require $\Omega(n^2)$ distance or shortest path queries, see Section 5.1.

All Shortest Paths and All Distances. Beerliova et al. [3] studied the network inference problem, where an oracle receives a node v and returns *all* shortest paths from v to *all* other nodes. For the verification problem, they showed that there is no approximation algorithm of factor $o(\log n)$ unless $P = NP$; and for the reconstruction problem, they gave a randomized on-line algorithm with competitive ratio $O(\sqrt{n \log n})$, which was later improved to $O(\log^2 n)$ by Sen and Muralidhara [18]. Beerliova et al. [3] suggested as a future direction: *It would also be interesting to study further query models. For example, a query could be given by nodes u and v and return all shortest paths between u and v (or just one shortest path).* They mentioned that *traceroute* experiments in real networks often reveal only a single shortest path (or at most a few different paths), but not all shortest paths. Hence our motivation of the shortest path oracle model: the oracle returns an *arbitrary* shortest path between a given pair of nodes.

The authors of Reference [3] also considered the distance oracle model: a distance oracle receives a node v and returns the distances from v to *all* other nodes in the graph. (This is in contrast to our distance oracle, which returns the distance between a given pair of nodes.) They noted that, in many networks, it is realistically possible to obtain the distance information, while it is difficult or impossible to obtain the path information. In the distance oracle model, they gave a randomized on-line algorithm for reconstruction with competitive ratio $O(\sqrt{n \log n})$. They proved that minimizing the number of queries for verification is NP-hard and they gave an approximation algorithm of factor $O(\log n)$, i.e., the number of queries is $O(\log n)$ times the optimum number of queries. This algorithm is based on a reduction to SET-COVER. To achieve this reduction, they showed an easy property for non-edge verification and more delicate properties for edge verification. Our verification algorithm in Section 3.1 bears similarity: we also use a reduction to SET-COVER. In our model, edge verification is straightforward since the graph has bounded maximum degree. The focus is thus on non-edge verification. We develop interesting properties for non-edge verification.

Betweenness Oracle. Recently, Abrahamsen et al. [1] studied a weaker oracle, called the *betweenness oracle*. This oracle receives three nodes u, v, w and returns whether w lies on a shortest path between u and v . In particular, the authors achieved a graph reconstruction algorithm with $\tilde{O}(n^{3/2})$ betweenness queries by extending our result on graph reconstruction (Theorem 1.1) to their setting.

Evolutionary Biology. Waterman et al. [21] introduced the problem of reconstructing evolutionary (phylogenetic) trees using a distance oracle. An oracle receives two species and returns their distance in the tree. When the tree has maximum degree Δ , Hein [9] gave a reconstruction algorithm using $O(\Delta n \log_\Delta n)$ queries. King, Zhang, and Zhou [13] showed that this bound is tight by providing a matching lower bound. On the other hand, when the maximum degree is unbounded, there is an $\Omega(n^2)$ lower bound [9].

Other Related Work. Dall'Asta et al. [8] studied network inference using a shortest path oracle which, upon receiving a *random* pair of vertices, returns a shortest path between them. Chung et al. [7] studied *network realization*, where the goal is to determine the sparsest graph given the distances between certain pairs of vertices. Tarissan, Latapy, and Prieur [19] studied network inference using *link queries*, where a link query returns whether there is a link between a pair of vertices.

1.3 Our Results

See Table 1 for a summary. The results based on the Voronoi cell decomposition and the greedy approaches are the main contributions.

Let Δ denote the maximum degree of any vertex in the graph G . The notation $\tilde{O}(f(n))$ stands for $O(f(n) \text{ polylog } n)$ where $f(n)$ is some function of n .

Table 1. Our Results (for Bounded Degree Graphs)

Objective	Query complexity	
verification (either oracle)	$n^{1+o(1)}$	Theorems 1.2 to 1.4
reconstruction (shortest path oracle)		
reconstruction(distance oracle)	$\tilde{O}(n^{3/2})$	Theorem 1.1
	$\Omega(n \log n / \log \log n)$	Theorem 5.2
	chordal: $\tilde{O}(n)$	Theorem 4.10
	outerplanar: $\tilde{O}(n)$	Theorem 4.19

Main results are in bold.

Algorithms Based on Voronoi Cell Decomposition. To design reconstruction and verification algorithms, we apply some algorithmic ideas previously developed for compact routing [20] and for Voronoi cells [10].

THEOREM 1.1. *For graph reconstruction using a distance oracle, there is a randomized algorithm with query complexity $O(\Delta^3 \cdot n^{3/2} \cdot \log^2 n \cdot \log \log n)$, which is $\tilde{O}(n^{3/2})$ when the maximum degree Δ is $O(\text{polylog } n)$.*

The algorithm of Theorem 1.1 (Section 2.2) selects a set of $\tilde{O}(\sqrt{n})$ nodes (called *centers*) partitioning V into *Voronoi cells* of roughly the same size, and expands them slightly so as to cover every edge of G (Figure 1). It is then sufficient to reconstruct each cell, which is done using exhaustive search inside that cell.

THEOREM 1.2. *For graph verification using a distance oracle, there is a randomized algorithm with query complexity $n^{1+O(\sqrt{(\log \log n + \log \Delta) / \log n})}$, which is $n^{1+o(1)}$ when the maximum degree Δ is $n^{o(1)}$.*

The algorithm of Theorem 1.2 (Section 2.3) is a more sophisticated *recursive* version of the algorithm in Theorem 1.1. Again, it selects a set of centers partitioning V into *Voronoi cells*. To verify each cell, instead of using exhaustive search, the algorithm applies recursion. This is a challenge because, when we query a pair (u, v) in a cell, the oracle returns the distance between u and v in the entire graph, not in the cell. Our approach is to allow selection of centers *outside* the cell, while still limiting the subcells to being contained *inside* the cell (Figure 2).

Greedy Algorithms. We provide simple greedy algorithms for verification and reconstruction.

The main task for verification is to confirm the *non-edges* of the graph. We develop a necessary and sufficient condition for a set of queries to confirm all the non-edges. This condition enables us to reduce the non-edge verification problem to SET-COVER. As a counterpart of the greedy algorithm for SET-COVER, greedy non-edge verification (Section 3.1) repeatedly makes queries that confirm the largest number of non-edges that are not yet confirmed. We have:

THEOREM 1.3. *If there is an algorithm for graph verification using $f(n, \Delta)$ distance queries, then there is a greedy algorithm for verification that uses $O(\Delta n + \log n \cdot f(n, \Delta))$ distance or shortest path queries.*

Next, we extend the idea of greedy verification and obtain a greedy algorithm for reconstruction (Section 3.2) as in the following theorem.

THEOREM 1.4. *If there is an algorithm for graph verification using $f(n, \Delta)$ distance queries, then there is a greedy algorithm for reconstruction that uses $O(\Delta n + \log n \cdot f(n, \Delta))$ shortest path queries.*

To prove Theorem 1.4, we show that each query to a shortest path oracle makes as much progress for reconstruction as the corresponding query to a distance oracle would have made for verifying

a given graph. The main observation here is that reconstruction using a shortest path oracle can be viewed as the verification of a dynamically changing graph using a distance oracle.

Combining Theorems 1.2 to 1.4, we know that greedy algorithms for graph verification using either oracle and for graph reconstruction using a shortest path oracle have query complexity $n^{1+o(1)}$.

We note that the greedy algorithm for verification is much simpler than the algorithm using Voronoi cell decomposition (Theorem 1.2), although both algorithms achieve the query complexity $n^{1+o(1)}$.

Algorithms Based on Decomposition by Separators. For *chordal graphs* and *outerplanar graphs*, there exist *balanced separators* of small size. This property enables us to design reconstruction algorithms of $\tilde{O}(n)$ query complexity in the distance oracle model. To reconstruct *chordal graphs* (Section 4.2) and *outerplanar graphs* (Section 4.3), our algorithms find a node x that is on many shortest paths and grow a small separator including x . Next, the algorithms partition the graph into subgraphs with respect to this separator and recurse on each subgraph. The difficulty is to obtain the node x . Computing all shortest paths to obtain x would require too many queries. However, we show that x can be found using random sampling and statistical estimation.

We note that both chordal graphs and outerplanar graphs (of bounded degree) have bounded treewidth. A natural open problem is the following: Is there a reconstruction algorithm for treewidth-bounded graphs using a near-linear number of queries to a distance oracle?

Lower Bounds. For graphs of unbounded degree, we give a simple $\Omega(n^2)$ query lower bound⁴ for both reconstruction and verification and under both oracle models (Section 5.1). For graphs of bounded degree, it is easy to see that both reconstruction and verification require $\Omega(n)$ distance or shortest path queries. In addition, there is a slightly better lower bound for reconstructing graphs of bounded degree using a distance oracle (Section 5.2). We thank Cyril Gavoille and Uri Zwick for this lower bound.

Approximate Reconstruction. We also study an approximate version of the reconstruction problem for graphs of unbounded degree (Section 6). We give a simple algorithm that achieves the optimal query complexity.

1.4 Notations and Definitions

Let $G = (V, E)$ be a connected, undirected, and unweighted graph, where V is the vertex set and E is the edge set. Let δ be the distance metric of G . For a subset of vertices $S \subseteq V$ and a vertex $v \in V$, define $\delta(S, v)$ to be $\min_{s \in S} \delta(s, v)$. For $v \in V$, define the neighborhood of v as $N(v) = \{u \in V : \delta(u, v) \leq 1\}$, and define the neighborhood of v within distance 2 as $N_2(v) = \{u \in V : \delta(u, v) \leq 2\}$. For $S \subseteq V$, define the neighborhood of S as $N(S) = \bigcup_{s \in S} N(s)$. We define $\hat{\delta}$, $\hat{N}$, and $\hat{N}_2$ similarly, but with respect to the graph $\hat{G}$.

For a pair of distinct vertices $(u, v) \in V^2$, we say that uv is an *edge* of G if $uv \in E$, and is a *non-edge* of G if $uv \notin E$.

For a subset of vertices $S \subseteq V$, let $G[S]$ be the subgraph induced by S . For a subset of edges $H \subseteq E$, we identify H with the subgraph induced by the edges of H . Let δ_H denote the distance metric of the subgraph H .

For vertices $s, t \in V$, $\text{QUERY}(s, t)$ returns either the distance $\delta(s, t)$ or a shortest s -to- t path, depending on the oracle model. For a vertex $s \in V$ and a subset $T \subseteq V$, define $\text{QUERY}(s, T)$ (or

⁴We note that this lower bound holds even when the graph is restricted to be outerplanar or chordal.

equivalently $\text{QUERY}(T, s)$ as $\text{QUERY}(s, t)$ for every $t \in T$. For subsets $S, T \subseteq V$, define $\text{QUERY}(S, T)$ as $\text{QUERY}(s, t)$ for every $(s, t) \in S \times T$.

In the verification problem, an algorithm, after performing a set of queries, outputs *no* if some query gives the wrong distance (or shortest path), and outputs *yes* if all queries give the right distances (or shortest paths).

1.5 Organization

In Section 2, we give reconstruction and verification algorithms based on Voronoi cell decomposition. In Section 2.1, we review the *center-selecting* technique from Reference [20], which is a main subroutine in our algorithms. In Sections 2.2 and 2.3, we prove Theorems 1.1 and 1.2, respectively. In Section 3, we give greedy algorithms for verification and for reconstruction. In Sections 3.1 and 3.2, we prove Theorems 1.3 and 1.4, respectively. In Section 4.1, we give some preliminaries on special graph classes. In Sections 4.2 and 4.3, we focus on chordal graphs and outerplanar graphs respectively. In Section 5, we study lower bounds. In Section 6, we provide results on the approximate reconstruction.

2 VORONOI CELL DECOMPOSITION

In this section, we design a reconstruction algorithm (Section 2.2) and a verification algorithm (Section 2.3) using a distance oracle. Both algorithms are based on the *center-selecting* technique from Reference [20] for general graphs, which we review in Section 2.1.

2.1 Center-Selecting Technique

Thorup and Zwick [20] gave a *compact routing scheme* that uses $\tilde{O}(n^{1/2})$ bits of memory at each node. The key to this scheme was to select a set of *centers*. First we recall some notations from [20]. For every node $w \in V$, the *cluster* of w with respect to a subset of nodes $A \subseteq V$ is defined as

$$C_A(w) := \{v \in V : \delta(w, v) < \delta(A, v)\}.$$

Note that if $w \in A$, then $C_A(w) = \emptyset$, since $\delta(w, v) \geq \delta(A, v)$ for every $v \in V$. The subscript A is omitted when clear from the context. The goal of the center-selecting problem is to select a set of centers $A \subseteq V$ such that every $|C_A(w)|$ is small, i.e., roughly $n/|A|$.

In Algorithm 1, we review the center-selecting algorithm from Reference [20]. The algorithm takes as input a graph $G = (V, E)$ and an integer parameter $s \in [1, n]$ and outputs a set of centers A . It uses a subroutine $\text{SAMPLE}(W, s)$, which receives a set $W \subseteq V$ and an integer s and returns a random subset of W obtained by selecting each element, independently, with probability $s/|W|$. If $|W| \leq s$, then $\text{SAMPLE}(W, s)$ returns the set W itself. Thorup and Zwick [20] proved the following lemma.

LEMMA 2.1 (REPHRASING OF THEOREM 3.1 IN [20]). *With probability at least $1/2$, the algorithm CENTERS (Algorithm 1) outputs a set $A \subseteq V$ such that $|A| \leq 4s \log n$ and $|C(w)| \leq 4n/s$ for every $w \in V$.*

We will use modified versions of the center-selecting algorithm (Algorithm 1) in the design of the reconstruction and the verification algorithms in Sections 2.2 and 2.3, respectively.

2.2 Reconstruction via a Distance Oracle

In this section, we prove the following theorem.

THEOREM 2.2. *For graph reconstruction using a distance oracle, there is a randomized algorithm that terminates after $O(\Delta^3 \cdot n^{3/2} \cdot \log^2 n \cdot \log \log n)$ queries with probability at least $1/4$.*

ALGORITHM 1: Selecting Centers [20] (see Theorem 2.1)

```

1: function CENTERS( $G, s$ )
2:    $A \leftarrow \emptyset, W \leftarrow V$ 
3:   while  $W \neq \emptyset$  do
4:      $A' \leftarrow \text{SAMPLE}(W, s)$ 
5:      $A \leftarrow A \cup A'$ 
6:     for  $w \in W$  do
7:        $C(w) \leftarrow \{v \in V : \delta(w, v) < \delta(A, v)\}$ 
8:      $W \leftarrow \{w \in W : |C(w)| > 4n/s\}$ 
9:   return  $A$ 

```

ALGORITHM 2: Selecting Centers by Estimation (see Theorem 2.3)

```

1: function ESTIMATED-CENTERS( $V, s$ )
2:    $A \leftarrow \emptyset, W \leftarrow V$ 
3:    $T \leftarrow K \cdot s \cdot \log n \cdot \log \log n$   $\triangleright K = O(1)$ 
4:   while  $W \neq \emptyset$  do
5:      $A' \leftarrow \text{SAMPLE}(W, s)$ 
6:      $\text{QUERY}(A', V)$ 
7:      $A \leftarrow A \cup A'$ 
8:     for  $w \in W$  do
9:        $X \leftarrow$  random multi-subset of  $V$  with  $T$  elements
10:       $\text{QUERY}(X, w)$ 
11:       $\tilde{C}(w) \leftarrow |\{v \in X : \delta(w, v) < \delta(A, v)\}| \cdot n/T$ 
12:       $W \leftarrow \{w \in W : \tilde{C}(w) \geq 5n/s\}$ 
13:   return  $A$ 

```

PROOF OF THEOREM 1.1 USING THEOREM 2.2. If the reconstruction algorithm in Theorem 2.2 does not terminate after $O(\Delta^3 \cdot n^{3/2} \cdot \log^2 n \cdot \log \log n)$ queries, we stop it and execute it again. The expected number of executions of the algorithm is a constant. Therefore, we obtain a reconstruction algorithm with expected query complexity $O(\Delta^3 \cdot n^{3/2} \cdot \log^2 n \cdot \log \log n)$, as stated in Theorem 1.1. $\square$

It only remains to prove Theorem 2.2.

2.2.1 Subroutine: Selecting Centers. The reconstruction algorithm uses a subroutine ESTIMATED-CENTERS (Algorithm 2) to select *centers* $A \subseteq V$ such that the vertices of V are roughly equipartitioned into the Voronoi cells centered at vertices in A . This algorithm is a modified version of the algorithm CENTERS (Algorithm 1). It takes as input the vertex set V of a graph and a parameter $s \in [1, n]$ and outputs a set of centers A . Unlike the algorithm CENTERS, it uses sampling to estimate each $|C(w)|$ so as to reduce the query complexity. Recall that the algorithm CENTERS eliminates a node w from the set W when $|C(w)| \leq 4n/s$. However, in our query model, computing $|C(w)|$ would require $\Omega(n)$ queries for each w . Instead, the algorithm ESTIMATED-CENTERS computes an estimate $\tilde{C}(w)$ of $|C(w)|$ using fewer queries, and then eliminates a node w from W when $\tilde{C}(w)$ is small. The following lemma analyzes the algorithm ESTIMATED-CENTERS. It is a counterpart of Theorem 2.1. Its proof combines arguments from [20] and Chernoff bounds.

LEMMA 2.3. *Let K (in Algorithm 2) be some well-chosen constant. With probability at least $1/4$, the algorithm ESTIMATED-CENTERS (Algorithm 2) terminates after $O(s \cdot n \cdot \log^2 n \cdot \log \log n)$ queries, and outputs a set $A \subseteq V$ such that $|A| \leq 12s \log n$ and $|C(w)| \leq 6n/s$ for every $w \in V$.⁵*

PROOF. Let W_i be the set W at the beginning of the i -th iteration of the **while** loop. Let $A' = \text{SAMPLE}(W, s)$ be the centers selected in this iteration, and let A be the union of A' with all previously selected centers. We say that the i -th iteration is *successful* if

$$\sum_{w \in W_i} |C_A(w)| \leq 2n|W_i|/s. \quad (1)$$

Thorup and Zwick [20] showed that each iteration is successful with probability at least $1/2$.

Consider a node $w \in W_i$. Let X be a random multi-subset of V with $T = K \cdot s \cdot \log n \cdot \log \log n$ elements for some parameter K . Define $Y = |\{v \in X : \delta(w, v) < \delta(A, v)\}|$, which is $|X \cap C_A(w)|$. Let $E[Y]$ be the expected value of Y . By standard Chernoff bounds, there is an absolute constant K such that

$$\begin{cases} \mathbb{P}[Y \geq 5T/s] \geq 1 - 1/(96n \log n), & \text{if } E[Y] > 6T/s, \\ \mathbb{P}[Y < 5T/s] > 1 - 1/(96n \log n), & \text{if } E[Y] < 4T/s. \end{cases}$$

Note that $|C_A(w)| = E[Y] \cdot n/T$. Define $\tilde{C}(w) = Y \cdot n/T$. Thus, with probability at least $1 - 1/(48n \log n)$, we have

$$\begin{cases} \tilde{C}(w) \geq 5n/s, & \text{if } |C_A(w)| > 6n/s, \\ \tilde{C}(w) < 5n/s, & \text{if } |C_A(w)| < 4n/s. \end{cases} \quad (2)$$

Since $|W_i| \leq n$, with probability at least $1 - 1/(48 \log n)$, Property Equation (2) holds for all $w \in W_i$ in the i -th iteration. We call iterations in which this happens *correct* iterations. As a consequence, if the i -th iteration is correct, then for every $w \in W_{i+1}$, we have $|C_A(w)| \geq 4n/s$.

If the i -th iteration is both successful and correct, which happens with probability at least $1/2 - 1/(48 \log n) > 1/3$, then we have

$$4n|W_{i+1}|/s \leq \sum_{w \in W_i} |C_A(w)| \leq 2n|W_i|/s,$$

and thus $|W_{i+1}| \leq |W_i|/2$. Since $|W|$ is non-increasing during the algorithm, the expected number of iterations is at most $3 \log n$.

By Markov's inequality, with probability at least $3/4$, the number of iterations is at most $12 \log n$. The probability that the first (at most) $12 \log n$ iterations are correct is at least $1 - 12 \log n / (48 \log n) = 3/4$. Therefore, with probability at least $1/2$, there are at most $12 \log n$ iterations and all iterations are correct. In that case, every $w \in V$ that is eliminated from W has $|C_A(w)| \leq 6n/s$. Observe that $|C_A(w)|$ cannot increase when elements are added to A . Therefore, $|C_A(w)| \leq 6n/s$ for every $w \in V$ when the **while** loop terminates.

The expected size of A is at most $3s \log n$, since the expected number of iterations is at most $3 \log n$, and in every iteration, a set A' of expected size s is added to A . By Markov's inequality, $|A| \leq 12s \log n$ with probability at least $3/4$.

All together, with probability at least $1/4$, there are at most $12 \log n$ iterations, $|A| \leq 12s \log n$, and $|C_A(w)| \leq 6n/s$ for every $w \in V$. In that case, the number of queries is at most

$$|A| \cdot n + (12 \log n) \cdot nT = O(s \cdot n \cdot \log^2 n \cdot \log \log n).$$

This completes the proof. $\square$

⁵We choose K so that the success probability of the algorithm is at least $1/4$ (instead of some large value, say, $1 - 1/\text{poly}(n)$), since this is sufficient to obtain the expected query complexity in Theorem 1.1.

Fig. 1. Partition by centers. Here vertices $a_1, \dots, a_5$ are centers in A . The dotted lines indicate the partition of V into Voronoi cells by those centers. Extending the Voronoi cells slightly, we obtain the cells $D_{a_1}, \dots, D_{a_5}$.

ALGORITHM 3: Reconstruction (see Theorem 2.2)

```

1: procedure RECONSTRUCT( $V$ )
2:    $A \leftarrow \text{ESTIMATED-CENTERS}(V, s)$  ▷ every pair in  $A \times V$  is queried
3:   for  $a \in A$  do
4:     QUERY( $N_2(a), V$ )
5:     for  $b \in N_2(a)$  do
6:        $C(b) \leftarrow \{v \in V : \delta(b, v) < \delta(A, v)\}$ 
7:        $D_a \leftarrow \bigcup \{C(b) : b \in N_2(a)\} \cup N_2(a)$ 
8:        $E_a \leftarrow \text{EXHAUSTIVE-QUERY}(D_a)$ 
9:   return  $\bigcup_a E_a$ 

```

2.2.2 Algorithm and Analysis.

Algorithm. The reconstruction algorithm (Algorithm 3) takes as input the vertex set V and computes the edge set of G . It first selects a set of centers A using ESTIMATED-CENTERS, and then partitions the graph into slightly overlapping subgraphs with respect to the centers in A , see Figure 1. The parameter s is defined later to achieve the desired query complexity. For each $a \in A$, we define its *extended Voronoi cell* $D_a \subseteq V$ as

$$D_a := \bigcup \{C(b) : b \in N_2(a)\} \cup N_2(a). \quad (3)$$

The algorithm then proceeds by exhaustive search within each subgraph $G[D_a]$, and returns all the edges found in these subgraphs.

Correctness Analysis. We show in Theorem 2.4 that these subgraphs together cover every edge of the graph. Thus, the output of the algorithm is indeed the edge set of G .

LEMMA 2.4. $\bigcup_{a \in A} G[D_a]$ covers every edge of G .

PROOF. Let uv be any edge of G . We prove that there is some $a \in A$ such that both u and v are in D_a . Without loss of generality, we assume $\delta(A, u) \leq \delta(A, v)$. We choose $a \in A$ such that $\delta(a, u) = \delta(A, u)$. If $\delta(a, u) \leq 1$, then both u and v are in $N_2(a) \subseteq D_a$. If $\delta(a, u) \geq 2$, let b be the

ALGORITHM 4: Selecting Centers for a Subset (see Theorem 2.5)

```

1: function SUBSET-CENTERS( $\hat{G}, U, s$ )
2:    $A \leftarrow \emptyset, W \leftarrow V$ 
3:   while  $W \neq \emptyset$  do
4:      $A' \leftarrow \text{SAMPLE}(W, s)$ 
5:      $A \leftarrow A \cup A'$ 
6:     for  $w \in W$  do
7:        $\hat{C}(w) \leftarrow \{v \in V : \hat{\delta}(w, v) < \hat{\delta}(A, v)\}$ 
8:        $W \leftarrow \{w \in W : |\hat{C}(w) \cap U| > 4|U|/s\}$ 
9:   return  $A$ 

```

vertex at distance 2 from a on a shortest a -to- u path in G . By the triangle inequality, we have

$$\delta(b, v) \leq \delta(b, u) + \delta(u, v) = \delta(b, u) + 1.$$

Since $\delta(b, u) = \delta(a, u) - 2$ and $\delta(a, u) = \delta(A, u) \leq \delta(A, v)$, we have $\delta(b, u) < \delta(A, u)$ and $\delta(b, v) < \delta(A, v)$. Therefore, both u and v are in $C(b)$, and thus in D_a , since $b \in N_2(a)$. $\square$

Query Complexity Analysis. By Theorem 2.3, with probability at least $1/4$, ESTIMATED-CENTERS uses $O(s \cdot n \cdot \log^2 n \cdot \log \log n)$ queries, and outputs a set $A \subseteq V$ such that $|A| \leq 12s \log n$, and $|C(w)| \leq 6n/s$, for every $w \in V$. In this case, the subsequent steps of the algorithm RECONSTRUCT have query complexity:

$$\sum_{a \in A} (|N_2(a)| \cdot n + |D_a|^2) = O(s \log n \cdot \Delta^2 (n + \Delta^2 n^2 / s^2)),$$

using the bounds on $|A|$ and $|C(w)|$, and the fact that $|N_2(a)| \leq \Delta^2 + 1$.

Let $s = \Delta \cdot \sqrt{n}$. Then with probability at least $1/4$, the algorithm RECONSTRUCT terminates after $O(\Delta^3 \cdot n^{3/2} \cdot \log^2 n \cdot \log \log n)$ queries, as stated in Theorem 2.2.

2.3 Verification via a Distance Oracle

In this section, we prove Theorem 1.2, which we now recall.

THEOREM 1.2. *For graph verification using a distance oracle, there is a randomized algorithm with query complexity $n^{1+O(\sqrt{(\log \log n + \log \Delta)/\log n})}$, which is $n^{1+o(1)}$ when the maximum degree Δ is $n^{o(1)}$.*

2.3.1 Subroutine: Selecting Centers. The verification algorithm uses the subroutine SUBSET-CENTERS (Algorithm 4), which takes as input a graph $\hat{G} = (V, \hat{E})$, a subset of vertices $U \subseteq V$, and an integer $s \in [1, n]$, and outputs a set of centers $A \subseteq V$ such that in the graph $\hat{G}$, the vertices of the subset U are roughly equipartitioned into the Voronoi cells centered at vertices in A . This algorithm is a generalization of the algorithm CENTERS (Algorithm 1). When the subset U equals V , this algorithm becomes the same as the algorithm CENTERS. We recall the definition of w 's cluster $C_A(w) := \{v \in V : \delta(w, v) < \delta(A, v)\}$ for every $w \in V$. Similarly, we define w 's cluster with respect to the graph $\hat{G}$ as $\hat{C}_A(w) := \{v \in V : \hat{\delta}(w, v) < \hat{\delta}(A, v)\}$. The subscript A is omitted when clear from the context. The following lemma is a straightforward extension of Theorem 2.1.

LEMMA 2.5. *With probability at least $1/2$, the algorithm SUBSET-CENTERS (Algorithm 4) outputs a set $A \subseteq V$, such that $|A| \leq 4s \log n$, and that $|\hat{C}(w) \cap U| \leq 4|U|/s$, for every $w \in V$. It uses no queries since $\hat{G}$ is given.*

2.3.2 Algorithm and Analysis.

Algorithm and Correctness Analysis. The task of verification comprises verifying that every edge of $\hat{G}$ is an edge of G , and verifying that every non-edge of $\hat{G}$ is a non-edge of G . The first part requires only $O(\Delta n)$ queries for graphs of bounded degree. Thus, the focus is on the second part, which is called *non-edge verification*. In the second part, we assume that the first part is already done, so we have $\hat{E} \subseteq E$.

We design a recursive algorithm for non-edge verification. Let $U \subseteq V$ represent the set of vertices in a recursive call. The goal is to verify that every non-edge of $\hat{G}[U]$ is a non-edge of $G[U]$. This is equivalent to verifying that every edge of $G[U]$ is an edge of $\hat{G}[U]$.

Let A be a set of centers computed by SUBSET-CENTERS. We define, for each $a \in A$, its *extended Voronoi cell restricted on U* as

$$D_a := \left(\bigcup \{C(b) : b \in N_2(a)\} \cup N_2(a) \right) \cap U. \quad (4)$$

Similarly, with respect to the graph $\hat{G}$, we define

$$\hat{D}_a := \left(\bigcup \{\hat{C}(b) : b \in \hat{N}_2(a)\} \cup \hat{N}_2(a) \right) \cap U. \quad (5)$$

The following lemma is a simple extension of Theorem 2.4.

LEMMA 2.6. $\bigcup_{a \in A} G[D_a]$ covers every edge of $G[U]$.

PROOF. Let uv be any edge of $G[U]$. We prove that there is some $a \in A$ such that both u and v are in D_a . Without loss of generality, we assume $\delta(A, u) \leq \delta(A, v)$. We choose $a \in A$ such that $\delta(a, u) = \delta(A, u)$. If $\delta(a, u) \leq 1$, then both u and v are in $N_2(a) \cap U \subseteq D_a$. If $\delta(a, u) \geq 2$, let b be the vertex at distance 2 from a on a shortest a -to- u path in G . By the triangle inequality, we have

$$\delta(b, v) \leq \delta(b, u) + \delta(u, v) = \delta(b, u) + 1.$$

Since $\delta(b, u) = \delta(a, u) - 2$ and $\delta(a, u) = \delta(A, u) \leq \delta(A, v)$, we have $\delta(b, u) < \delta(A, u)$ and $\delta(b, v) < \delta(A, v)$. Therefore, both u and v are in $C(b)$, and thus in D_a , since $b \in N_2(a)$. $\square$

From Theorem 2.6, to verify that every edge of $G[U]$ is an edge of $\hat{G}[U]$, we only need to verify that every edge of $G[D_a]$ is an edge of $\hat{G}[D_a]$, for every $a \in A$. This enables us to apply recursion on each D_a .

The main difficulty is how to obtain D_a efficiently. If we compute D_a from its definition, we would need to compute $N_2(a)$, which requires $\Omega(n)$ queries since $N_2(a)$ may contain nodes outside U . Instead, a careful analysis shows that we can check whether $D_a = \hat{D}_a$ without even knowing $N_2(a)$, whereas $\hat{D}_a$ can be inferred from the graph $\hat{G}$ with no queries. See Theorem 2.7 below, which is the key to the design of the verification algorithm.

LEMMA 2.7. Assume that $\hat{E} \subseteq E$. If $\delta(b, u) = \hat{\delta}(b, u)$ for every pair (b, u) from $\bigcup_{a \in A} \hat{N}_2(a) \times U$, then $D_a = \hat{D}_a$ for all $a \in A$.⁶

PROOF. The proof is delicate but elementary. For every $b \in \bigcup_{a \in A} \hat{N}_2(a)$, we have $\hat{C}(b) \cap U = C(b) \cap U$, because $\hat{\delta}(b, u) = \delta(b, u)$ and $\hat{\delta}(A, u) = \delta(A, u)$ for every $u \in U$. Therefore, $\hat{D}_a$ can be rewritten as

$$\hat{D}_a = \left(\bigcup \{C(b) : b \in \hat{N}_2(a)\} \cup \hat{N}_2(a) \right) \cap U.$$

Since $\hat{E} \subseteq E$, we have $\hat{N}_2(a) \subseteq N_2(a)$. Therefore, $\hat{D}_a \subseteq D_a$.

⁶The statement is non-trivial, since $N_2(a)$ may differ from $\hat{N}_2(a)$.

Fig. 2. Two levels of recursive calls of $\text{VERIFY-SUBGRAPH}(\hat{G}, V, s)$. The solid points are top-level centers returned by $\text{SUBSET-CENTERS}(\hat{G}, V, s)$. The dotted lines indicate the partition of V into Voronoi cells by those centers. For a center a , its extended Voronoi cell is $\hat{D}_a$. On the second level of the recursive call for $\hat{D}_a$, the hollow points are the centers returned by $\text{SUBSET-CENTERS}(\hat{G}, \hat{D}_a, s)$. Observe that some of those centers lie outside $\hat{D}_a$. The dashed lines indicate the partition of $\hat{D}_a$ into Voronoi cells by those centers. For a center a' on the second level, its extended Voronoi cell is $\hat{D}'_{a'}$.

On the other hand, we have $N_2(a) \cap U \subseteq \hat{N}_2(a) \cap U$, because $\hat{\delta}(a, u) = \delta(a, u)$ for every $u \in N_2(a) \cap U$. To prove $D_a \subseteq \hat{D}_a$, it only remains to show that, for any vertex $u \notin N_2(a)$ such that $u \in C(b) \cap U$ for some $b \in N_2(a)$, we have $u \in C(x) \cap U$ for some $x \in \hat{N}_2(a)$. We choose x to be the vertex at distance 2 from a on a shortest a -to- u path in $\hat{G}$. By the assumption and the definition of x , we have

$$\delta(x, u) = \hat{\delta}(x, u) = \hat{\delta}(a, u) - 2 = \delta(a, u) - 2.$$

By the triangle inequality, and using $b \in N_2(a)$ and $u \in C(b)$, we have

$$\delta(a, u) \leq \delta(a, b) + \delta(b, u) \leq 2 + \delta(b, u) < 2 + \delta(A, u).$$

Therefore, $\delta(x, u) < \delta(A, u)$. Thus, $u \in C(x) \cap U$. $\square$

The recursive algorithm VERIFY-SUBGRAPH for non-edge verification is given in Algorithm 5. It receives a graph $\hat{G} = (V, \hat{E})$ and a subset $U \subseteq V$, and verifies all the non-edges of $\hat{G}[U]$. It first queries every $(u, v) \in \bigcup_{a \in A} \hat{N}_2(a) \times U$, and then recurses on each extended Voronoi cell $\hat{D}_a$, see Figure 2. The parameters s and n_0 are defined later to achieve the desired query complexity. Correctness of the algorithm follows from Theorems 2.6 and 2.7.

Query Complexity Analysis. To provide intuition, we first analyze an algorithm of 4 recursive levels, and show that its query complexity is $\tilde{O}(n^{4/3})$. To simplify the presentation, we assume $\Delta = O(1)$. Let $s = n^{1/3}$ and let n_0 be some well-chosen constant. Consider any recursive call $\text{VERIFY-SUBGRAPH}(\hat{G}, U)$ where $|U| > n_0$. From Theorem 2.5, the expected number of iterations of the **repeat** loop is constant. Let $A \subseteq V$ be the centers at the end of the **repeat** loop. Since $\Delta = O(1)$, every $\hat{N}_2(a)$ has constant size for $a \in A$. Every $\hat{C}(w) \cap U$ has size $O(|U|/n^{1/3})$, so every $\hat{D}_a$ has size $O(|U|/n^{1/3})$. Since $|A| = \tilde{O}(n^{1/3})$, the number of recursive calls on the next level is $\tilde{O}(n^{1/3})$. Therefore, during the recursion, on the second level, there are $\tilde{O}(n^{1/3})$ recursive calls, where every

ALGORITHM 5: Verification (see Theorem 1.2)

```

1: procedure VERIFY-SUBGRAPH( $\hat{G}, U$ )
2:   if  $|U| > n_0$  then
3:     repeat
4:        $A \leftarrow \text{SUBSET-CENTERS}(\hat{G}, U, s)$ 
5:       until  $|A| \leq 4s \log n$  and  $|\hat{C}(w) \cap U| \leq 4|U|/s$  for every  $w \in V$ 
6:       for  $a \in A$  do
7:         QUERY( $\hat{N}_2(a), U$ )
8:         for  $b \in \hat{N}_2(a)$  do
9:            $\hat{C}(b) \leftarrow \{v \in V : \hat{\delta}(b, v) < \hat{\delta}(A, v)\}$ 
10:           $\hat{D}_a \leftarrow \left( \bigcup \{ \hat{C}(b) : b \in \hat{N}_2(a) \} \cup \hat{N}_2(a) \right) \cap U$ 
11:          VERIFY-SUBGRAPH( $\hat{G}, \hat{D}_a, s$ )
12:     else
13:       QUERY( $U, U$ )

```

subset has size $O(n^{2/3})$; on the third level, there are $\tilde{O}(n^{2/3})$ recursive calls, where every subset has size $O(n^{1/3})$; and on the fourth level, there are $\tilde{O}(n)$ recursive calls, where every subset has size $O(1)$. Every recursive call with subset U uses $\tilde{O}(n^{1/3} \cdot |U|)$ queries. Therefore, the overall query complexity is $\tilde{O}(n^{4/3})$.

Next, we give the full proof of the complexity as stated in Theorem 1.2. Define

$$k_0 = \left\lceil \sqrt{\frac{\log n}{\log(\log n \cdot 64(\Delta^2 + 1)^2)}} \right\rceil.$$

Let $s = n^{1/k_0}$ and $n_0 = (4(\Delta^2 + 1))^{k_0}$. Consider any recursive call when $|U| > n_0$. From Theorem 2.5, the expected number of iterations of the **repeat** loop is constant. Let $A \subseteq V$ be the centers at the end of the **repeat** loop. Then $|A| \leq 4s \log n$ and every $|\hat{C}(w) \cap U|$ is at most $4|U|/s$. Since the graph has maximum degree Δ , the size of every $\hat{D}_a$ is at most $(\Delta^2 + 1) \cdot \max(4|U|/s, 1)$. Therefore, by induction, for any $1 \leq k \leq k_0 + 1$, any subset U on the k^{th} level of the recursion has size at most $t_k := n(4(\Delta^2 + 1)/s)^{k-1}$, where $t_{k_0+1} = n_0$. Hence, the maximum level of the recursion is at most $k_0 + 1$.⁷

Consider the recursive calls with $|U| \leq n_0$. There are at most $(4s \log n)^{k_0}$ such calls and each call takes $|U|^2 \leq (4(\Delta^2 + 1))^{2k_0}$ queries. Thus, their overall query complexity is at most $n \cdot (\log n \cdot 64(\Delta^2 + 1)^2)^{k_0} \leq n^{1+1/k_0}$.

Consider the recursive calls with $|U| > n_0$ on the k^{th} level of the recursion for some fixed $k \in [1, k_0]$.⁸ There are at most $(4s \log n)^{k-1}$ such calls and each takes at most $(\Delta^2 + 1)|A| \cdot |U|$ queries, where $|U| \leq t_k$. Thus, their overall query complexity is at most $n^{1+1/k_0} (\log n \cdot 16(\Delta^2 + 1))^k$. Summing over k from 1 to k_0 , the query complexity of all recursive calls with $|U| > n_0$ is at most $2 \cdot n^{1+1/k_0} (\log n \cdot 16(\Delta^2 + 1))^{k_0} \leq 2 \cdot n^{1+2/k_0}$.

Therefore, the number of queries for non-edge verification is at most $3 \cdot n^{1+2/k_0}$, which is $n^{1+O(\sqrt{\log \log n + \log \Delta} / \log n)}$. Since it takes at most $\Delta n = n^{1+\log \Delta / \log n}$ queries to verify that $\hat{E} \subseteq E$, we obtain the overall query complexity as stated in Theorem 1.2.

⁷The randomness does not have an influence on the maximum level of the recursion, but only on the running time of the **repeat** loops.

⁸We note that there are no recursive calls on the $(k_0 + 1)^{\text{th}}$ level (i.e., the last level) of the recursion with $|U| > n_0$.

3 GREEDY ALGORITHMS

In this section, we first give a greedy verification algorithm using a distance oracle and with roughly the same query complexity, and then extend it to a reconstruction algorithm using a shortest path oracle.

3.1 Verification via a Distance Oracle

THEOREM 3.1. *If there is an algorithm for graph verification using $f(n, \Delta)$ distance queries, then there is a greedy algorithm for verification that uses $O(\Delta n + \log n \cdot f(n, \Delta))$ distance queries.*

PROOF OF THEOREM 1.3 USING THEOREM 3.1. Each query to a distance oracle can be simulated by the same query to a shortest path oracle. $\square$

It only remains to prove Theorem 3.1. Let $\widehat{NE}$ be the set of the non-edges of $\hat{G}$. For each pair of vertices $(u, v) \in V^2$, we define a set $S_{u,v} \subseteq \widehat{NE}$ of non-edges. This is the key to the reduction from graph verification to the SET-COVER problem.

$$S_{u,v} = \{ab \in \widehat{NE} : \hat{\delta}(u, a) + \hat{\delta}(b, v) + 1 < \hat{\delta}(u, v)\}.^9 \quad (6)$$

The following lemmas relate the sets $\{S_{u,v} : (u, v) \in V^2\}$ with non-edge verification.

LEMMA 3.2. *Assume that $\hat{E} \subseteq E$. For every $(u, v) \in V^2$, if $\delta(u, v) = \hat{\delta}(u, v)$, then every pair $ab \in S_{u,v}$ is a non-edge of G .*

PROOF. Consider any pair $ab \in S_{u,v}$. By the triangle inequality, $\delta(u, a) + \delta(a, b) + \delta(b, v) \geq \delta(u, v) = \hat{\delta}(u, v)$. By the definition of $S_{u,v}$ and using $\hat{E} \subseteq E$, we have $\hat{\delta}(u, v) > \hat{\delta}(u, a) + \hat{\delta}(b, v) + 1 \geq \delta(u, a) + \delta(b, v) + 1$. Thus, $\delta(a, b) > 1$, i.e., ab is a non-edge of G . $\square$

LEMMA 3.3. *If a set of queries T verifies that every non-edge of $\hat{G}$ is a non-edge of G , then $\bigcup_{(u,v) \in T} S_{u,v} = \widehat{NE}$.*

PROOF. Assume, for a contradiction, that some $ab \in \widehat{NE}$ does not belong to any $S_{u,v}$ for $(u, v) \in T$. Consider adding ab to the set of edges $\hat{E}$: this will not create a shorter path between u and v , for any $(u, v) \in T$. Thus, including ab in $\hat{E}$ is consistent with the answers of all queries in T . This contradicts the assumption that T verifies that ab is a non-edge of G .¹⁰ $\square$

Algorithm and Analysis. From Theorems 3.2 and 3.3, the non-edge verification is equivalent to the SET-COVER problem with the universe $\widehat{NE}$ and the sets $\{S_{u,v} : (u, v) \in V^2\}$. The SET-COVER problem can be solved using the well-known greedy algorithm [11], which gives an H_n -approximation. Hence our greedy algorithm for verification (Algorithm 6). For the query complexity, first, verifying that $\hat{E} \subseteq E$ takes at most Δn queries, since the graph has maximum degree Δ . The part of non-edge verification uses a number of queries that is at most $(\ln n + 1)$ times the optimum number of queries. Thus, the overall query complexity is $O(\Delta n + \log n \cdot f(n, \Delta))$. We complete the proof of Theorem 3.1.

3.2 Reconstruction via a Shortest Path Oracle

In this section, we prove Theorem 1.4 that we recall.

THEOREM 1.4. *If there is an algorithm for graph verification using $f(n, \Delta)$ distance queries, then there is a greedy algorithm for reconstruction that uses $O(\Delta n + \log n \cdot f(n, \Delta))$ shortest path queries.*

⁹In this definition, a pair $ab \in \widehat{NE}$ is not ordered, i.e., ab is identical to ba .

¹⁰We assume that Δ is an unknown quantity and cannot be used to rule out the existence of the edge ab .

ALGORITHM 6: Greedy Verification

```

1: procedure VERIFY( $\hat{G}$ )
2:   for  $uv \in \hat{E}$  do
3:     QUERY( $u, v$ )
4:    $Y \leftarrow \emptyset$ 
5:   while  $\hat{E} \cup Y$  does not cover all vertex pairs do
6:     choose  $(u, v)$  that maximizes  $|S_{u,v} \setminus Y|$   $\triangleright S_{u,v}$  defined in Equation (6)
7:     QUERY( $u, v$ )
8:      $Y \leftarrow Y \cup S_{u,v}$ 

```

ALGORITHM 7: Greedy Reconstruction

```

1: procedure RECONSTRUCT( $V$ )
2:    $u_0 \leftarrow$  an arbitrary vertex
3:   for  $u \in V \setminus \{u_0\}$  do
4:     QUERY( $u, u_0$ ) to get a shortest  $u$ -to- $u_0$  path
5:    $X \leftarrow$  the union of the above paths
6:    $Y \leftarrow \emptyset$ 
7:   while  $X \cup Y$  does not cover all vertex pairs do
8:     choose  $(u, v)$  that maximizes  $|S_{u,v}^X \setminus Y|$   $\triangleright S_{u,v}^X$  defined in Equation (7)
9:     QUERY( $u, v$ ) to get a shortest  $u$ -to- $v$  path
10:    if  $\delta_G(u, v) = \delta_X(u, v)$  then
11:       $Y \leftarrow Y \cup S_{u,v}^X$ 
12:    else
13:       $e \leftarrow$  some edge of the above  $u$ -to- $v$  path that is not in  $X$ 
14:       $X \leftarrow X \cup \{e\}$ 
15:  return  $X$ 

```

Algorithm. Our greedy algorithm (Algorithm 7) constructs an increasing set X of edges so that in the end $X = E$. At any time, the candidate graph is X .¹¹ Initially, X is the union of the shortest paths given as answers by $n - 1$ queries, so that X is a connected subgraph of G spanning all vertices. At each subsequent step, the algorithm makes a query that leads either to the confirmation of many non-edges of G , or to the discovery of an edge of G .

Formally, we define, for every pair $(u, v) \in V^2$,

$$S_{u,v}^X = \{ab \in \text{non-edges of } X : \delta_X(u, a) + \delta_X(b, v) + 1 < \delta_X(u, v)\}. \quad (7)$$

This is similar to $S_{u,v}$ defined in Equation 6. From Theorem 3.2, the pairs in $S_{u,v}^X$ can be confirmed as non-edges of G if $\delta_G(u, v) = \delta_X(u, v)$. At each step, the algorithm queries a pair (u, v) that maximizes the size of the set $S_{u,v}^X \setminus Y$. After the query, either all pairs in $S_{u,v}^X \setminus Y$ are confirmed as non-edges of G , or $\delta_G(u, v) \neq \delta_X(u, v)$, and in that case, some edge along a shortest u -to- v path in G that is not in X is revealed; we then add this edge to X .

¹¹We identify X with the subgraph induced by the edges of X .

Correctness Analysis. We note that the algorithm maintains the invariant that all pairs in X are confirmed as edges of G , and that all pairs in Y are confirmed as non-edges of G . Thus, when $X \cup Y$ covers all vertex pairs, we have $X = E$.

Query Complexity Analysis. First, consider the queries that lead to $\delta_G(u, v) \neq \delta_X(u, v)$. After each such query, an edge is added to X . This can happen at most $|E| \leq \Delta n$ times, since the graph has maximum degree Δ .

Next, consider the queries that lead to $\delta_G(u, v) = \delta_X(u, v)$. Define R to be the set of vertex pairs that are not in $X \cup Y$. We analyze the size of R during the algorithm. After each query with $\delta_G(u, v) = \delta_X(u, v)$, the size of R decreases by $|S_{u,v}^X \setminus Y|$. To lower bound $|S_{u,v}^X \setminus Y|$, we consider the problem of non-edge verification using a distance oracle on the input graph X , and let T be an (unknown) optimal set of queries. Since there is a verification algorithm using $f(n, \Delta)$ distance queries, we have $|T| \leq f(n, \Delta)$. By Theorem 3.3, the sets $S_{u,v}^X$ for all pairs $(u, v) \in T$ together cover $R \cup Y$, hence R . Therefore, for some pair $(u, v) \in T$ we have

$$|S_{u,v}^X \setminus Y| \geq |R|/|T| \geq |R|/f(n, \Delta).$$

Initially, $|R| \leq n(n-1)/2$, and right before the last query, $|R| \geq 1$. Thus, the number of queries with $\delta_G(u, v) = \delta_X(u, v)$ is $O(\log n) \cdot f(n, \Delta)$.

Therefore, the overall query complexity is $O(\Delta n + \log n \cdot f(n, \Delta))$.

Remark. The above analysis depends crucially on the fact that $f(n, \Delta)$ is a uniform bound on the number of distance queries for verifying *any* n -vertex graph of maximum degree Δ . Thus, even though the graph X changes during the course of the algorithm because of the queries (u, v) such that $\delta_G(u, v) \neq \delta_X(u, v)$, each query (u, v) with $\delta_G(u, v) = \delta_X(u, v)$ confirms at least $1/f(n, \Delta)$ fraction of the non-edges.

4 DECOMPOSITION BY SEPARATORS

In this section, we consider special classes of graphs (chordal graphs and outerplanar graphs), and we give $\tilde{O}(n)$ reconstruction algorithms for these graphs via a distance oracle. Our algorithms use separators to decompose the graph into subgraphs, and then apply recursion on each subgraph.

4.1 Preliminaries

We review the definition and properties of *separators*, *tree decomposition*, *chordal graphs*, and *outerplanar graphs*.

Definition 4.1. A subset $S \subseteq V$ is a β -balanced separator of the graph $G = (V, E)$ (for $\beta < 1$) if the size of every connected component of $G[V \setminus S]$ is at most $\beta|V|$. In this case, the partition of $G[V \setminus S]$ into connected components is called a β -balanced partition.

Definition 4.2. A *tree decomposition* of a graph $G = (V, E)$ is a tree T with nodes $n_1, n_2, \dots, n_\ell$ where every node n_i is identified with a *bag* $S_i \subseteq V$ such that the following conditions hold:

- (1) For every vertex v in G , the nodes whose bags contain v form a connected subtree of T .
- (2) For every edge uv in G , some bag contains both u and v .

The *width* of a tree decomposition T is the size of the largest bag minus 1, and the *treewidth* of G is the minimum width over all possible tree decompositions T of G .

LEMMA 4.3 ([16], LEMMA 4.3.1). *Every tree decomposition T of G contains a bag $S \subseteq V$ that is a $(1/2)$ -balanced separator of G .*

Definition 4.4. A graph is *chordal* if every cycle of length greater than three has a chord, an edge connecting two non-consecutive vertices on the cycle.

An introduction to chordal graphs can be found in, e.g., Reference [4].

LEMMA 4.5 ([4]). *Let G be a chordal graph. Then G has a tree decomposition T such that every bag is a unique maximal clique.*¹²

From Theorems 4.3 and 4.5, we have:

COROLLARY 4.6. *Let G be a chordal graph of maximum degree Δ . Then G has a tree decomposition T such that every bag is a unique maximal clique of size at most $\Delta + 1$ and that some bag $S \subseteq V$ in T is a $(1/2)$ -balanced separator of G .*

Definition 4.7. A graph is *outerplanar* if it can be embedded in the plane with all vertices on the exterior face.

Chartrand and Harary [6] first introduced outerplanar graphs and showed that an outerplanar graph contains no subgraph homeomorphic to K_4 or $K_{2,3}$.

Definition 4.8. We say that the k -tuple $(x_1, \dots, x_k) \in V^k$ (where $k \geq 3$) is a *polygon* if $G[\{x_1, \dots, x_k\}]$ has exactly k edges: $x_1x_2, x_2x_3, \dots, x_kx_1$.

We introduce the notion of a *convex* subset of vertices.

Definition 4.9. A subset of vertices $U \subseteq V$ is said to be *convex* if, for every pair of vertices $(x, y) \in U^2$, all shortest paths in G between x and y go through nodes only in U .

4.2 Reconstruction of Chordal Graphs

In this section, we prove the following theorem.

THEOREM 4.10. *For reconstruction of chordal graphs using a distance oracle, there is a randomized algorithm (Algorithm 10) with query complexity $O(\Delta^3 2^\Delta \cdot n(2^\Delta + \log^2 n) \log n)$, which is $\tilde{O}(n)$ when the maximum degree Δ is $O(\log \log n)$.*

The reconstruction algorithm first computes a vertex that is on many shortest paths, and then grows a clique separator including this vertex. Next, it partitions the graph into subgraphs with respect to this separator, and recursively reconstructs each subgraph. Before providing the reconstruction algorithm, we first describe two subroutines: computing a shortest path between a pair of given vertices (Section 4.2.1) and partitioning the graph with respect to a set of vertices (Section 4.2.2).

In what follows, the set U represents the set of vertices for which we are currently reconstructing the induced subgraph $G[U]$.

4.2.1 Subroutine: Computing a Shortest Path.

LEMMA 4.11. *Let U be a convex subset of V . Let a and b be vertices in U . The function $\text{SHORTEST-PATH}(U, a, b)$ (Algorithm 8) outputs a shortest path between a and b in G . Its query complexity is $O(|U| \log |U|)$.*¹³

The algorithm is based on dichotomy. First, it makes $2|U|$ queries to get $\delta(u, a)$ and $\delta(u, b)$ for every $u \in U$. Then it finds the middle node c on some shortest a -to- b path. Next, it recursively computes a shortest a -to- c path and a shortest c -to- b path. The concatenation of these two paths is a shortest a -to- b path.

¹²A maximal clique is a clique which is not contained in any other clique.

¹³The $\log |U|$ factor comes since there might be several shortest paths between a and b .

ALGORITHM 8: Finding a Shortest Path (see Theorem 4.11)

```

1: function SHORTEST-PATH( $U, a, b$ )
2:   if  $\delta(a, b) > 1$  then
3:     QUERY( $U, a$ ); QUERY( $U, b$ )
4:      $T \leftarrow \{v \in U \mid \delta(v, a) + \delta(v, b) = \delta(a, b)\}$ 
5:      $\ell \leftarrow \lfloor \delta(a, b)/2 \rfloor$ 
6:      $c \leftarrow$  an arbitrary node in  $T$  such that  $\delta(c, a) = \ell$ 
7:      $U_1 \leftarrow \{v \in T \mid \delta(v, a) < \ell\}$ 
8:      $U_2 \leftarrow \{v \in T \mid \delta(v, a) > \ell\}$ 
9:      $P_1 \leftarrow$  SHORTEST-PATH( $U_1, a, c$ )
10:     $P_2 \leftarrow$  SHORTEST-PATH( $U_2, c, b$ )
11:    return the concatenation of  $P_1$  and  $P_2$ 
12:   else
13:     return the path of a single edge  $ab$ 

```

During the recursion, the distance $\delta(a, b)$ between the endpoints is reduced by half at every level. Thus, there are $O(\log |U|)$ levels of the recursion. Since the U sets on each level of the recursion are disjoint, the total number of queries at each level is $O(|U|)$. Therefore, the overall query complexity of SHORTEST-PATH is $O(|U| \log |U|)$.

4.2.2 Subroutine: Partitioning by a Set.

LEMMA 4.12. *Let U be a convex subset of V . Let S be a subset of U . The function PARTITION(U, S) (Algorithm 9) outputs the partition of $G[U \setminus S]$ into connected components. Its query complexity is $O(\Delta|S| \cdot |U|)$.*

The algorithm roughly partitions the vertices of $U \setminus S$ according to the second vertex on a shortest path from S to the vertex, expanding slightly so as to cover every edge. Formally, for every vertex $a \in W$ where $W := (N(S) \cap U) \setminus S$, we define the *cluster* of a as

$$B(a) = \{x \in U \setminus S \mid \delta(a, x) \leq \delta(S, x)\}. \quad (8)$$

Since U is convex, every $x \in U \setminus S$ belongs to some cluster $B(a)$. The PARTITION algorithm successively merges two clusters that have overlaps. See Figure 3.

ALGORITHM 9: Computing the Partition (See Theorem 4.12)

```

1: function PARTITION( $U, S$ )
2:   QUERY( $S, U$ ) and obtain  $N(S) \cap U$ 
3:   QUERY( $N(S) \cap U, U$ )
4:    $W \leftarrow (N(S) \cap U) \setminus S$ 
5:    $\mathcal{B} \leftarrow \{B(a) \mid a \in W\}$   $\triangleright B(a)$  defined in Equation (8)
6:   while  $\exists B_1, B_2 \in \mathcal{B}$  s.t.  $B_1 \cap B_2 \neq \emptyset$  do
7:     merge  $B_1$  and  $B_2$  in  $\mathcal{B}$ 
8:   return  $\mathcal{B}$ 

```

The query complexity of the algorithm is $O(|N(S)| \cdot |U|) = O(\Delta|S| \cdot |U|)$. Theorem 4.12 follows directly from Theorems 4.13 and 4.14.

LEMMA 4.13. *Let C be a connected component in $G[U \setminus S]$. Then $C \subseteq B$ for some set $B \in \mathcal{B}$ in the output of the algorithm.*

Fig. 3. Here $S = \{s_1, s_2\}$ and $W = \{a, b, c, d, e\}$. The clusters $B(a)$, $B(b)$, $B(c)$, $B(d)$, $B(e)$ are indicated by the circles. Using their overlaps, the algorithm produces the partition $\mathcal{B} = \{B(a) \cup B(b), B(c) \cup B(d) \cup B(e)\}$.

PROOF. First we argue that every edge of $G[U \setminus S]$ is contained in some cluster $B(a)$ where $a \in W$. Consider any edge uv of $G[U \setminus S]$. Without loss of generality, we assume $\delta(S, u) \leq \delta(S, v)$. We choose $s \in S$ such that $\delta(s, u) = \delta(S, u)$. Let a be the second vertex on a shortest s -to- u path in G . Then $a \in N(S) \setminus S$. Since U is convex, we have $a \in U$. Therefore, $a \in W$. By the triangle inequality, we have

$$\delta(a, v) \leq \delta(a, u) + \delta(u, v) = \delta(a, u) + 1.$$

Since $\delta(a, u) = \delta(s, u) - 1$ and $\delta(s, u) = \delta(S, u) \leq \delta(S, v)$, we have $\delta(a, u) < \delta(S, u)$ and $\delta(a, v) \leq \delta(S, v)$. Therefore, both u and v are in $B(a)$.

Next we consider any pair of vertices that are connected in $G[U \setminus S]$ and any path connecting them. After the merging step (Lines 6–7), the entire path belongs to the same set in $\mathcal{B}$. $\square$

LEMMA 4.14. *Let $B \subseteq \mathcal{B}$ be a set in the output of the algorithm. Then $B \subseteq C$ for some connected component C in $G[U \setminus S]$.*

PROOF. First we show that for every $a \in W$ and every $x \in B(a)$, a and x belong to the same component in $G[U \setminus S]$. Suppose there exists some $x \in B(a)$, such that x and a belong to different components in $G[U \setminus S]$. Then any shortest path from a to x must pass through the separator S , so we have $\delta(a, x) \geq \delta(a, S) + \delta(S, x) = 1 + \delta(S, x)$. Contradiction with $x \in B(a)$.

Next we prove an invariant on $\mathcal{B}$ during the **while** loop (Lines 6–7): *Every set $B \in \mathcal{B}$ is a subset of some connected component of $G[U \setminus S]$.* This invariant holds before the **while** loop starts. Suppose the invariant holds before the i^{th} iteration of the **while** loop, and in this iteration two sets $B_1, B_2 \in \mathcal{B}$ get merged. Since $B_1 \cap B_2 \neq \emptyset$, there exists some node $z \in B_1 \cap B_2$. All nodes in B_1 (resp. in B_2) are in the same component of $G[U \setminus S]$ as z . Thus, all nodes in $B_1 \cup B_2$ are in the same component of $G[U \setminus S]$. By induction, the invariant holds when the **while** loop terminates. This completes the proof. $\square$

4.2.3 Algorithm and Analysis. The RECONSTRUCT-CHORDAL algorithm (Algorithm 10) takes as input a convex subset $U \subseteq V$ of a chordal graph and returns the edge set of $G[U]$. The key function BALANCED-PARTITION-CHORDAL finds a β -balanced partition of U , for some β . This function first

ALGORITHM 10: Reconstruction of Chordal Graphs (see Theorem 4.10)

```

1: procedure RECONSTRUCT-CHORDAL( $U$ )
2:   if  $|U| > n_0$  then
3:      $(\{U_i\}_i, K) \leftarrow \text{BALANCED-PARTITION-CHORDAL}(U)$ 
4:     return  $\bigcup_i \text{RECONSTRUCT-CHORDAL}(U_i \cup K)$ 
5:   else
6:     return EXHAUSTIVE-QUERY( $U$ )

7: function BALANCED-PARTITION-CHORDAL( $U$ )
8:   repeat
9:      $\{(a_i, b_i)\}_{1 \leq i \leq C_1} \leftarrow$  uniform and independent random pairs from  $U$ 
10:    for  $i \leftarrow 1$  to  $C_1$  do
11:       $P_i \leftarrow \text{SHORTEST-PATH}(U, a_i, b_i)$  ▷ see Algorithm 8
12:       $x \leftarrow$  the node in  $U$  with the most occurrences among  $\{P_i\}_i$ 
13:      QUERY( $x, U$ ) and obtain  $N(x) \cap U$ 
14:      QUERY( $N(x) \cap U, N(x) \cap U$ ) and obtain all cliques in  $U$  containing  $x$ 
15:      for every clique  $K$  in  $U$  containing  $x$  do
16:         $\mathcal{P}_K \leftarrow \text{PARTITION}(U, K)$  ▷ see Algorithm 9
17:    until a  $\beta$ -balanced partition  $(\mathcal{P}_K, K)$  is found

```

computes a vertex that is on many shortest paths in the sampling, and then looks for a β -balanced clique separator including this vertex. It repeats until a β -balanced partition is found. We set $n_0 = 2^{\Delta+2}(\Delta + 1)^2$; $C_1 = 36(\Delta + 1)^2 \log |U|$; and

$$\beta = \max \left(1 - 1/(\Delta \cdot 2^{\Delta+1}), \sqrt{1 - 1/(4(\Delta + 1))} \right).$$

LEMMA 4.15. *RECONSTRUCT-CHORDAL(U) returns the edge set of $G[U]$.*

PROOF. By Theorem 4.12, $\{U_i\}_i$ is the partition of $G[U \setminus K]$ into connected components. Every edge of $G[U]$ belongs to some $G[U_i \cup K]$, since there is no edge between any U_i and U_j for $i \neq j$. Thus, the edge set of $G[U]$ is the union of the edge sets of $G[U_i \cup K]$ for all i . The statement follows by induction. $\square$

To bound the query complexity, the key is the following lemma.

LEMMA 4.16. *In every **repeat** loop of the function BALANCED-PARTITION-CHORDAL, a β -balanced partition is found with probability at least $2/3$.*

To prove Theorem 4.16, we need Theorems 4.17 and 4.18.

LEMMA 4.17. *For every $v \in U$, let p_v denote the fraction of pairs $(a, b) \in U^2$ such that v is on some shortest path between a and b . Then $\max_v p_v \geq 1/(2(\Delta + 1))$.*

PROOF. By Theorem 4.6, there is some clique $S \subseteq U$ of size at most $\Delta + 1$ such that every connected component in $G[U \setminus S]$ has size at most $|U|/2$. Notice that for any pair of vertices a, b from different components of $G[U \setminus S]$, any shortest a -to- b path must go by some node in S . The number of such pairs is at least $|U|^2/2$. By the Pigeonhole Principle, there exists some $z \in S$ such that for at least $1/|S| \geq 1/(\Delta + 1)$ fraction of these pairs, their shortest paths go by z . Thus, $p_z \geq 1/(2(\Delta + 1))$. $\square$

LEMMA 4.18. For every vertex $v \in U$, let $\tilde{p}_v$ denote the fraction of pairs (a_i, b_i) among C_1 uniform and independent random pairs from U such that v is on some shortest path between a_i and b_i . Let $x = \arg \max_v \tilde{p}_v$. If $C_1 \geq 9 \log |U| / (\max_v p_v)^2$, then with probability at least $2/3$, we have $p_x > (\max_v p_v)/2$.

PROOF. Let z be the node in U which maximizes p_z . We will show that $\mathbb{P}[\tilde{p}_y \geq \tilde{p}_z] < 1/(3|U|)$ for any node $y \in U$ with $p_y \leq p_z/2$. This implies the lemma statement because we then have $\mathbb{P}[\exists y \in U, \text{ s.t. } p_y \leq p_z/2 \text{ and } \tilde{p}_y \geq \tilde{p}_z] < 1/3$. Thus, with probability at least $2/3$, the node x with the largest $\tilde{p}_x$ satisfies $p_x > p_z/2$.

Let y be any node with $p_y \leq p_z/2$. For every integer $i \in [1, C_1]$, define a variable $Y_i \in \{0, 1\}$ such that $Y_i = 1$ if the node y is on a shortest a_i -to- b_i path and $Y_i = 0$ otherwise. Since $\{(a_i, b_i)\}_i$ are uniform and independent random pairs from U , $\{Y_i\}_i$ are independent random variables, and each Y_i equals 1 with probability p_y . We then have $\mathbb{E}[Y_i] = p_y \leq p_z/2$. Similarly, for every $i \in [1, C_1]$, define a variable $Z_i \in \{0, 1\}$ such that $Z_i = 1$ if the node z is on a shortest a_i -to- b_i path and $Z_i = 0$ otherwise. Then $\mathbb{E}[Z_i] = p_z$.

For every i , define $T_i = Y_i - Z_i$. Let $\bar{T}$ be the average of all T_i 's. Then $\mathbb{E}[\bar{T}] = \mathbb{E}[T_i] = \mathbb{E}[Y_i] - \mathbb{E}[Z_i] \leq -p_z/2$. We have

$$\mathbb{P}[\tilde{p}_y \geq \tilde{p}_z] = \mathbb{P}\left[\sum_i T_i \geq 0\right] \leq \mathbb{P}\left[|\bar{T} - \mathbb{E}[\bar{T}]| \geq p_z/2\right].$$

By Hoeffding's inequality, the last term is at most $2 \cdot \exp(-p_z^2 \cdot C_1/8)$, which is at most $1/(3|U|)$ by the definition of p_z and C_1 . Therefore, we have $\mathbb{P}[\tilde{p}_y \geq \tilde{p}_z] < 1/(3|U|)$. $\square$

PROOF OF THEOREM 4.16. By Theorem 4.6, there is a *tree decomposition* T of $G[U]$ such that every bag of T is a unique maximal clique of $G[U]$. Consider any iteration of the **repeat** loop. Let x be the node computed in Line 12 of Algorithm 10. Let T_x be the subtree of T induced by the bags containing x . Define F to be the forest after removing T_x from T . For any subgraph H of T , define $V(H) \subseteq U$ to be the set of vertices in U that appear in at least one bag of H .

Case 1: There exists some connected component T' in F with $(1 - \beta)|U| \leq |V(T')| \leq \beta|U|$. Consider the (unique) edge K_1K_2 in T such that $K_1 \in T_x$ and $K_2 \in T'$. $K_1 \cap K_2$ is a β -balanced separator of $G[U]$, since $V(T')$ is a component in $G[U \setminus (K_1 \cap K_2)]$. Thus, $K_1 \supseteq K_1 \cap K_2$ is also a β -balanced separator. Since K_1 contains x , K_1 is one of the cliques checked on Line 15. The algorithm succeeds in finding a β -balanced separator.

Case 2: There exists some connected component T' in F with $|V(T')| > \beta|U|$. The algorithm then fails to find a β -balanced separator. We bound the probability of this case by at most $1/3$. Again let K_1K_2 be the edge in T such that $K_1 \in T_x$ and $K_2 \in T'$. For any vertices u, v of U that are in $V(T')$, any shortest u -to- v path cannot go by x . Since there are at least β^2 fraction of such pairs in U^2 , we have $p_x \leq 1 - \beta^2$, which is at most $1/(4(\Delta + 1))$ by the definition of β . This happens with probability at most $1/3$ by Theorems 4.17 and 4.18.

We argue that the two cases above are exhaustive. Suppose, for the sake of contradiction, that every component T' in F is such that $|V(T')| < (1 - \beta)|U|$. The number of components in F is at most $\Delta \cdot 2^\Delta$, because every component has a bag that contains a neighbor of x , and all bags are unique. Thus, $|V(F)| < \Delta \cdot 2^\Delta \cdot (1 - \beta)|U|$, which is at most $|U|/2$ by the definition of β . On the other hand, every node $v \in U \setminus N(x)$ is contained in some bag from F , so $|V(F)| \geq |U| - (\Delta + 1)$, which is greater than $|U|/2$ since $|U| > n_0$. Contradiction. $\square$

Query Complexity Analysis. First, we analyze the complexity of BALANCED-PARTITION-CHORDAL. Computing C_1 shortest paths takes $O(\Delta^2|U| \log^2 |U|)$ queries, since a shortest path between two

given nodes can be computed using $O(|U| \log |U|)$ queries (Theorem 4.11). We note that the neighborhood $N(x)$ of x has size at most $\Delta + 1$, and there are at most 2^Δ cliques containing x . For every clique K containing x , $\text{PARTITION}(U, K)$ takes $O(\Delta|K| \cdot |U|)$ queries by Lemma 4.12, where $|K| \leq \Delta + 1$. Therefore, every **repeat** loop takes $O(\Delta^2|U|(2^\Delta + \log^2 |U|))$ queries. By Theorem 4.16, the expected number of **repeat** loops is constant. Thus, the query complexity of $\text{BALANCED-PARTITION-CHORDAL}$ is $O(\Delta^2|U|(2^\Delta + \log^2 |U|))$.

Next, we analyze the complexity of $\text{RECONSTRUCT-CHORDAL}$. Let $q(m)$ be the number of queries when $|U| = m$. We have

$$q(m) = O\left(\Delta^2 m(2^\Delta + \log^2 m)\right) + \sum_i q(|U_i| + |K|),$$

where $\{U_i\}_i$ is a β -balanced partition of U by the separator K . Hence

$$q(n) = O\left(\Delta^2 n(2^\Delta + \log^2 n) \log_{\frac{1}{\beta}} n\right) = O\left(\Delta^3 2^\Delta \cdot n(2^\Delta + \log^2 n) \log n\right).$$

This completes the proof of Theorem 4.10.

4.3 Reconstruction of Outerplanar Graphs

In this section, we prove the following theorem.

THEOREM 4.19. *For reconstruction of outerplanar graphs using a distance oracle, there is a randomized algorithm with query complexity $O(\Delta^2 \cdot n \log^3 n)$, which is $\tilde{O}(n)$ when the maximum degree Δ is $O(\text{polylog } n)$.*

Similar to reconstructing chordal graphs in Section 4.2, the algorithm first finds a separator to partition the graph into subgraphs and then recurses on each subgraph. However, the separator here may be a *polygon* (see Theorem 4.8). In Section 4.3.1, we show how to partition the graph with respect to a polygon.

4.3.1 Subroutine: Partitioning by a Polygon. In this section, we prove the following lemma.

LEMMA 4.20. *Suppose the graph G is outerplanar. Let U be a convex subset of V . Let $a, b, c \in U$ be consecutive nodes along some unknown polygon $(q_1, \dots, q_l)$ in U . The function $\text{PARTITION-BY-POLYGON}(U, a, b, c)$ outputs the partition of $G[U \setminus \{q_1, \dots, q_l\}]$ into connected components. Its query complexity is $O(\Delta|U| \log |U|)$.*

The function $\text{PARTITION-BY-POLYGON}$ consists of the following two phases.

Phase 1: Finding the Polygon. The algorithm is given in Algorithm 11. The key is to compute the a -to- c path along the polygon that does not go through b . First, the algorithm computes the middle point z of this path. Next, it computes a shortest path P_1 between a and z and a shortest path P_2 between z and c . The polygon is the concatenation of the path P_1 , the path P_2 , the edge cb , and the edge ba . In the algorithm, $\text{PARTITION}(U, \{a, b\})$ and $\text{PARTITION}(U, \{b, c\})$ use $O(\Delta \cdot |U|)$ queries (Theorem 4.12), and computing the shortest paths P_1 and P_2 uses $O(|U| \log |U|)$ queries (Theorem 4.11). Thus, the overall query complexity of Phase 1 is $O(\Delta \cdot |U| \log |U|)$.

Phase 2: Partitioning by the Polygon. Given the polygon $(q_1, \dots, q_l)$, we compute the partition of $G[U \setminus \{q_1, \dots, q_l\}]$ into connected components. Let $m = \lfloor l/2 \rfloor$. First we compute the partition of $G[U \setminus \{q_1, q_m, q_{m+1}, q_l\}]$ into connected components. Then we make two calls to PARTITION-BY-PATH (Algorithm 12) for the path $q_1, \dots, q_m$ and the path $q_{m+1}, \dots, q_l$, respectively. Thus, we obtain the final partition (Figure 4).

The recursive procedure PARTITION-BY-PATH receives a convex subset $Z \subseteq U$ of vertices and two integers s, t such that $1 \leq s < t \leq l$, and returns the partition of $G[Z \setminus \{q_s, q_{s+1}, \dots, q_t\}]$ into

ALGORITHM 11: Finding a Polygon

```

1: procedure FIND-POLYGON( $U, a, b, c$ )
2:    $(A_1, \dots, A_{k_1}) \leftarrow \text{PARTITION}(U, \{a, b\})$  ▷ see Algorithm 9
3:   Let  $A_i$  be the component containing  $c$ 
4:    $(C_1, \dots, C_{k_2}) \leftarrow \text{PARTITION}(U, \{b, c\})$  ▷ see Algorithm 9
5:   Let  $C_j$  be the component containing  $a$ 
6:    $T \leftarrow (A_i \cap C_j) \cup \{a, c\}$ 
7:    $\text{QUERY}(a, T), \text{QUERY}(c, T)$ 
8:    $d \leftarrow \min_{u \in T} \{\delta(a, u) + \delta(u, c)\}$ 
9:   Let  $z \in T$  be such that  $\delta(a, z) + \delta(z, c) = d$  and  $\delta(a, z) = \lfloor d/2 \rfloor$ 
10:   $P_1 \leftarrow \text{SHORTEST-PATH}(T, a, z)$  ▷ see Algorithm 8
11:   $P_2 \leftarrow \text{SHORTEST-PATH}(T, z, c)$  ▷ see Algorithm 8
12:  return the concatenation of  $P_1, P_2, cb, ba$ 

```

ALGORITHM 12: Partition with respect to a path

```

1: procedure PARTITION-BY-PATH( $Z, s, t$ )
2:   if  $t > s + 1$  then
3:      $m \leftarrow \lfloor (s + t)/2 \rfloor$ 
4:      $\mathcal{P} \leftarrow \text{PARTITION}(Z, \{q_m\})$  ▷ see Algorithm 9
5:     Let  $Z_1$  be the component in  $\mathcal{P}$  containing  $q_s$ 
6:     Let  $Z_2$  be the component in  $\mathcal{P}$  containing  $q_t$ 
7:      $\mathcal{P}_1 \leftarrow \text{PARTITION-BY-PATH}(Z_1, s, m)$ 
8:      $\mathcal{P}_2 \leftarrow \text{PARTITION-BY-PATH}(Z_2, m, t)$ 
9:     return  $(\mathcal{P} \setminus \{Z_1, Z_2\}) \cup \mathcal{P}_1 \cup \mathcal{P}_2$ 
10:  else
11:    return  $\text{PARTITION}(Z, \{q_s, q_t\})$  ▷ see Algorithm 9

```

components. We analyze its query complexity. The number of queries of $\text{PARTITION}(Z, \{q_m\})$ and $\text{PARTITION}(Z, \{q_s, q_t\})$ is $O(\Delta \cdot |Z|)$ by Theorem 4.12. During the recursion, the length of the path $q_s, \dots, q_t$ is reduced by half at every level, so there are at most $\log |U|$ levels of the recursion. Since the Z sets on each level are disjoint, the overall query complexity on each level is $O(\Delta \cdot |U|)$. Thus, the query complexity of PARTITION-BY-PATH is $O(\Delta \cdot |U| \log |U|)$. Since the partition of $G[U \setminus \{q_1, q_m, q_{m+1}, q_t\}]$ into components takes $O(\Delta \cdot |U|)$ queries (Theorem 4.12), the overall query complexity of Phase 2 is $O(\Delta \cdot |U| \log |U|)$.

4.3.2 Algorithm and Analysis. The reconstruction algorithm (Algorithm 13) takes as input a convex subset $U \subseteq V$ of an outerplanar graph and returns the edge set of $G[U]$. Similar to Section 4.2, the function $\text{BALANCED-PARTITION-OUTERPLANAR}$ computes a β -balanced partition of U . We set $n_0 = 20$; $C_1 = 324 \log |U|$, and $\beta = \sqrt{11/12}$.

Correctness of the $\text{RECONSTRUCT-OUTERPLANAR}$ algorithm is a trivial adaptation from Theorem 4.15. To bound the query complexity, the key is the following lemma.

LEMMA 4.21. *In every **repeat** loop of the function $\text{BALANCED-PARTITION-OUTERPLANAR}$, a β -balanced partition is found with probability at least $2/3$.*

To prove Theorem 4.21, we need Theorem 4.18 and the following lemma (Theorem 4.22).

Fig. 4. Partition by the polygon $q_1, \dots, q_t$.**ALGORITHM 13:** Reconstruction of Outerplanar Graphs

```

1: procedure RECONSTRUCT-OUTERPLANAR( $U$ )
2:   if  $|U| > n_0$  then
3:      $(U_1, \dots, U_\ell) \leftarrow \text{BALANCED-PARTITION-OUTERPLANAR}(U)$ 
4:     return  $\bigcup_i \text{RECONSTRUCT-OUTERPLANAR}(U_i)$ 
5:   else
6:     return EXHAUSTIVE-QUERY( $U$ )

7: function BALANCED-PARTITION-OUTERPLANAR( $U$ )
8:   repeat
9:      $\{(a_i, b_i)\}_{1 \leq i \leq C_1} \leftarrow$  uniform and independent random pairs from  $U$ 
10:    for  $i \leftarrow 1$  to  $C_1$  do
11:       $P_i \leftarrow \text{SHORTEST-PATH}(U, a_i, b_i)$  ▷ see Algorithm 8
12:       $x \leftarrow$  the node in  $U$  with the most occurrences among  $\{P_i\}_i$ 
13:      QUERY( $x, U$ ) and obtain  $N(x) \cap U$ 
14:       $\mathcal{P} \leftarrow \text{PARTITION}(U, N(x) \cap U)$  ▷ see Algorithm 9
15:      if  $\mathcal{P}$  is  $\beta$ -balanced then return  $\mathcal{P}$  ▷ see Fig. 5
16:      if  $x$  has two consecutive neighbors  $y$  and  $y'$  such that the component between  $y$  and  $y'$  contains more than  $\beta|U|$  nodes then
17:         $\mathcal{P} \leftarrow \text{PARTITION-BY-POLYGON}(U, y, x, y')$  ▷ see Section 4.3.1
18:    until a  $\beta$ -balanced partition  $\mathcal{P}$  is found

```

LEMMA 4.22. For every $v \in U$, let p_v denote the fraction of pairs $(a, b) \in U^2$ such that v is on some shortest path between a and b . Then $\max_v p_v \geq 1/6$.

PROOF. Since $G[U]$ is outerplanar, it has treewidth at most 2. From Theorem 4.3, there exists some set $S \subseteq U$ of size at most 3 such that every connected component in $G[U \setminus S]$ has size at most $|U|/2$. The following argument is similar to that in the proof of Theorem 4.17. Notice that for any

Fig. 5. Partition by the neighbors $y_1, \dots, y_6$ of x .

pair of vertices a, b from different components of $G[U \setminus S]$, any shortest a -to- b path must go by some node in S . The number of such pairs is at least $|U|^2/2$. By the Pigeonhole Principle, there exists some $z \in S$ such that for at least $1/|S| \geq 1/3$ fraction of these pairs, their shortest paths go by z . Thus, $p_z \geq 1/6$. $\square$

PROOF OF THEOREM 4.21. Let x be the node computed in Line 12 of Algorithm 13. If the partition by the neighbors (Line 14) is β -balanced or the partition by the polygon (Line 17) is β -balanced, then the algorithm succeeds. Consider the case when no β -balanced partition is found. Then there must be a convex subset $S \subseteq U$ such that $x \notin S$ and $|S| \geq \beta|U|$. For every $(a, b) \in S^2$, any shortest path between a and b does not go by x . Thus, $p_x \leq 1 - (|S|/|U|)^2 \leq 1 - \beta^2 = 1/12$ by the definition of β . Therefore, it is sufficient to bound the probability of $p_x > 1/12$ by at least $2/3$. From Theorem 4.22, we have $C_1 \geq 9 \log |U| / (\max_v p_v)^2$. Thus, by Theorem 4.18, we have $p_x > 1/12$ with probability at least $2/3$. $\square$

Query Complexity Analysis. First, we analyze the complexity of BALANCED-PARTITION-OUTERPLANAR. Computing C_1 shortest paths takes $O(|U| \log^2 |U|)$ queries, since a shortest path between two given nodes can be computed using $O(|U| \log |U|)$ queries (Theorem 4.11). We note that the neighborhood $N(x)$ of x has size at most $\Delta + 1$. By Theorem 4.12, PARTITION($U, N(x)$) takes $O(\Delta^2 |U|)$ queries. The procedure PARTITION-BY-POLYGON takes $O(\Delta |U| \log |U|)$ queries. Therefore, every **repeat** loop takes $O(\Delta^2 |U| \log^2 |U|)$ queries. From Theorem 4.21, the expected number of **repeat** loops is constant. Thus, the overall query complexity is $O(\Delta^2 |U| \log^2 |U|)$.

Next, we analyze the complexity of RECONSTRUCT-OUTERPLANAR. Let $q(m)$ be the number of queries when $|U| = m$. We have

$$q(m) = O(\Delta^2 m \log^2 m) + \sum_i q(|U_i|),$$

where $\{U_i\}_i$ is a β -balanced partition of U . Hence $q(n) = O(\Delta^2 n \log^3 n)$.

Thus we complete the proof of Theorem 4.19.

Remark. An outerplanar graph is a generalization of a tree. The query complexity of our reconstruction algorithm for outerplanar graphs is only slightly worse than the optimal query complexity $O(\Delta n \log_\Delta n)$ for reconstructing trees [9]. On the other hand, the tree reconstruction model

typically restricts queries to pairs of tree leaves, but our model allows queries on any pair of vertices, not just leaves.

5 LOWER BOUNDS

In Section 5.1, we give a lower bound for general graphs where the maximum degree is unbounded; and in Section 5.2, we give a lower bound for graphs of maximum degree Δ .

5.1 General Graphs

LEMMA 5.1 (EXTENSION OF PROPOSITION 7 IN REFERENCE [17]). *For graph reconstruction or graph verification using a distance oracle or a shortest path oracle, any algorithm has query complexity $\Omega(n^2)$.*

PROOF. We extend the lower bound for graph reconstruction by Reyzin and Srivastava [17] to graph verification. Consider a graph G whose vertices are $v_1, \dots, v_n$ and whose edges form a star: there is an edge $v_1 v_i$ for every $2 \leq i \leq n$. In addition, G may or may not contain a new edge $v_i v_j$, for $2 \leq i, j \leq n$. In the graph verification problem, the star graph is given as $\hat{G}$. To check whether G contains a new edge, we need to perform $\Omega(n^2)$ distance or shortest path queries. $\square$

5.2 Graphs of Bounded Degree

In this section, we prove the following theorem.

THEOREM 5.2. *For graph reconstruction using a distance oracle, assuming that the maximum degree Δ is at least 3 and is $o(n^{1/2})$, any algorithm has query complexity $\Omega(\Delta n \log n / \log(\log n / \log \Delta))$.¹⁴*

PROOF. To provide intuition, we first show a lower bound of $\Omega(\Delta n \log n / \log \log n)$, assuming that $n = 3t - 1$, where $t = 2^k$ for some integer k . Consider a family $\mathcal{G}$ of graphs G as follows: the vertex set is $\{v_1, \dots, v_n\}$; the first $2t - 1$ vertices form a complete binary tree of height k (with leaves $v_t, \dots, v_{2t-1}$); the vertices $v_{2t}, \dots, v_{3t-1}$ induce an arbitrary subgraph of maximum degree $\Delta - 1$; there is an edge between v_i and v_{i+t} for every $i \in [t, 2t - 1]$ and there are no other edges. Then every vertex in G has degree at most Δ , and the diameter of the graph is at most $2k + 2 = O(\log n)$. Every distance query returns a number between 1 and $2k + 2$, so it gives $O(\log \log n)$ bits of information. From information theory, the number of queries to reconstruct the graph is at least the logarithm of the number of graphs in $\mathcal{G}$ divided by the maximum number of bits of information per query. The number of graphs in $\mathcal{G}$ is the number of different graphs of t vertices and of maximum degree $\Delta - 1$, which is $\Omega(n^{\Omega(\Delta n)})$ when $\Delta = o(\sqrt{n})$ (see Reference [15]). Therefore, we have a query lower bound:

$$\frac{\log \left(\Omega \left(n^{\Omega(\Delta n)} \right) \right)}{O(\log \log n)} = \Omega \left(\frac{\Delta n \log n}{\log \log n} \right).$$

To prove the bound as stated in Theorem 5.2, we only need to replace the above complete binary tree by a complete $(\Delta - 1)$ -ary tree. The diameter of the graph is now $O(\log n / \log \Delta)$. The theorem statement follows. $\square$

6 APPROXIMATE RECONSTRUCTION

In this section, we study the approximate version of the reconstruction problem using a distance oracle on general graphs (not necessarily of bounded degree). We first give a simple algorithm

¹⁴This result is not comparable with the lower bound by King et al. [13] for tree reconstruction, since we allow querying any pairs of nodes, not only pairs of leaves in a tree.

(Algorithm 14, see Theorem 6.1), and then show that this algorithm is optimal by providing a query lower bound of the same complexity (see Theorem 6.2).

Let $G = (V, E)$ be a connected, undirected, and unweighted graph. Let δ be the distance metric of G . Let f be any sublinear function of n . An f -approximation $\tilde{\delta}$ of the metric δ is such that, for every $(u, v) \in V^2$, $\tilde{\delta}(u, v) \leq \delta(u, v) \leq f \cdot \tilde{\delta}(u, v)$.

Algorithm 14 receives the vertex set V and outputs the approximate metric $\tilde{\delta}$. The algorithm repeatedly picks a node u such that the distances between u and some other nodes are not yet estimated. It then makes queries between u and the other nodes and obtains an estimate $\tilde{\delta}(x, y)$ for every node x within distance $f/2$ from u and for every node y in the graph. The algorithm repeats the above until all distances are estimated.

ALGORITHM 14

```

1: procedure APPROX-RECONSTRUCT( $V$ )
2:    $S \leftarrow V$ 
3:   while  $S \neq \emptyset$  do
4:      $u \leftarrow$  arbitrary node from  $S$ 
5:     QUERY( $u, S$ )
6:     Let  $B$  be the set of nodes whose distance to  $u$  is strictly less than  $f/2$ 
7:     Set  $\tilde{\delta}(x, y) = 1$  for all pairs  $(x, y) \in B \times B$  with  $x \neq y$ 
8:     Set  $\tilde{\delta}(x, y) = \delta(u, y) - \delta(u, x)$  for all pairs  $(x, y) \in B \times (S \setminus B)$ 
9:     Remove  $B$  from  $S$ 
10:  return  $\tilde{\delta}$ 

```

THEOREM 6.1. *The algorithm APPROX-RECONSTRUCT(V) computes an f -approximation of the graph metric δ using $O(n^2/f)$ distance queries.*

PROOF. We prove that in the end of the algorithm, for every $(x, y) \in V^2$, we have $\tilde{\delta}(x, y) \leq \delta(x, y) \leq f \cdot \tilde{\delta}(x, y)$. Look at any iteration of the **while** loop. First, consider any pair $(x, y) \in B \times B$ with $x \neq y$. We have

$$\tilde{\delta}(x, y) = 1 \leq \delta(x, y) \leq \delta(x, u) + \delta(u, y) < (f/2) + (f/2) = f = f \cdot \tilde{\delta}(x, y).$$

Next, consider any pair $(x, y) \in B \times (S \setminus B)$. On the one hand, by the triangular inequality,

$$\delta(x, y) \geq \delta(u, y) - \delta(u, x) = \tilde{\delta}(x, y).$$

On the other hand, by the triangular inequality,

$$\delta(x, y) \leq (\delta(u, y) - \delta(u, x)) + 2\delta(u, x).$$

The first term is $\tilde{\delta}(x, y)$. The second term, by the definition of B , is at most $(f - 1)$. Since $x \in B$ and $y \notin B$, we have $\tilde{\delta}(x, y) \geq 1$, so the second term can be bounded by $f - 1 \leq (f - 1) \cdot \tilde{\delta}(x, y)$. Adding completes the proof of the upper bound.

Now we analyze the query complexity of the algorithm. Let $U \subseteq V$ be the set of nodes u chosen in Line 4 during the algorithm. For every pair of distinct nodes $u, u' \in U$, we have $\delta(u, u') \geq f/2$. For every $u \in U$, define $N(u, f/4)$ as the neighborhood of u within distance $f/4$. Then we have $|N(u, f/4)| \geq f/4$, since G is connected. Observe that the sets $\{N(u, f/4)\}_{u \in U}$ are disjoint, so there are at most $4n/f$ sets, i.e., $|U| \leq 4n/f$. For every $u \in U$, the algorithm makes $O(n)$ queries. Thus, the total number of queries is $O(n^2/f)$. $\square$

Fig. 6. In this example, $f = 2$; The construction of the tree is based on two permutations σ_1 and σ_2 .

For the lower bound, we extend the arguments for the case of $f = 1$ by Reyzin and Srivastava [17, Proposition 7] and obtain the following theorem.

THEOREM 6.2. *To compute an f -approximation of the graph metric δ using a distance oracle, any algorithm requires $\Omega(n^2/f)$ queries.*

PROOF. To simplify the proof, we assume $n = 2fk + 1$, for $k \in \mathbb{N}$. We define a family of instances as follows. For each f -tuple $(\sigma_1, \dots, \sigma_f)$ such that every σ_i (where $i \in [1, f]$) is a permutation of $\{1, \dots, k\}$, we define a tree instance: it has one vertex a_0 as the root (on the first level), k vertices $a_1, \dots, a_k$ on the second level, k vertices $a_{k+1}, \dots, a_{2k}$ on the third level, $\dots$, and k vertices $a_{n-k}, \dots, a_{n-1}$ on the $(2f + 1)^{\text{th}}$ level. For every $l \in [2, f]$ and every $i \in [1, k]$, there is an edge between the i^{th} node on level l and the i^{th} node on level $l + 1$. For every $l \in [f + 1, 2f]$ and every $i \in [1, k]$, there is an edge between the i^{th} node on level l and the $\sigma_{l-f}(i)^{\text{th}}$ node on level $l + 1$. See Figure 6. We observe that every tree constructed above has k branches from the root, and every branch is a path of $2f + 1$ nodes. We will show that any algorithm requires $\Omega(n^2/f)$ queries to compute an f -approximation of the metric on these instances.

First, notice that for these instances, any f -approximation $\tilde{\delta}$ of the metric can be transformed into the metric δ without queries: For every nodes u and v on consecutive levels between level $f + 1$ and level $2f + 1$, uv is an edge of G if and only if $\tilde{\delta}(u, v) < 2f$. This is because, if u and v are in the same branch, we have $\delta(u, v) = 1$, so $\tilde{\delta}(u, v) \leq f$; and if u and v are in different branches, we have $\delta(u, v) = \delta(a_0, u) + \delta(a_0, v) \geq 2f$, so $\tilde{\delta}(u, v) \geq 2f$. Therefore, we only need to prove that any algorithm for the exact reconstruction problem requires $\Omega(n^2/f)$ queries on these instances.

Given that an instance is from the constructed family, let $\mathcal{A}$ be any algorithm for the exact reconstruction problem. We assume that $\mathcal{A}$ does not make *redundant queries*, i.e., queries whose answers can be deduced in advance. Obviously, any query with the root is redundant. For any two nodes u and v , let l_u and l_v be their levels. The query (u, v) is redundant when $l_u \leq f + 1$ and $l_v \leq f + 1$, since the first $f + 1$ levels are fixed. Thus, every query (u, v) is such that $l_u > f + 1$ and $l_v \geq 2$ (we assume $l_u \geq l_v$ without loss of generality). The answer is either $l_u - l_v$, if u and v are in the same branch; or $l_u + l_v - 2$, if u and v are in different branches. We can equivalently identify the answer as *Yes* or *No* to the question: *Are u and v in the same branch?* The key is to upper bound the number of *Yes* answers during the algorithm. We introduce the *component graph* H , which represents the information collected from the *Yes* answers during the algorithm. The vertex set of H is defined to be the set of all nodes of level between $f + 1$ and $2f + 1$. At the beginning, the edge set of H is empty. Each time when $\mathcal{A}$ receives a *Yes* answer to a query (u, v) , we add an edge to H as follows:

Fig. 7. Decision tree of $\mathcal{A}$.

- (1) If $l_u > f + 1$ and $l_v \geq f + 1$, then we add the edge uv to H ;
- (2) If $l_u > f + 1$ and $2 \leq l_v < f + 1$, then we add the edge uw to H , where w is the node on level $f + 1$ that is in the same branch as v .

There could not be cycles in H , otherwise there are redundant queries. The number of connected components in H is at least k , since every connected component in H contains nodes from the same branch of the tree and there are k branches. The number of edges in H is the number of vertices minus the number of connected components, which is at most $k(f + 1) - k = kf$. Since every *Yes* answer adds an edge into H , the algorithm $\mathcal{A}$ stops after at most kf *Yes* answers.

Next, we show the lower bound by a decision tree argument. See Figure 7. First, $\mathcal{A}$ queries some pair (u_1, v_1) . If the answer is *Yes*, it queries some pair (u_2, v_2) , otherwise it queries some pair (u_3, v_3) , and so on. $\mathcal{A}$ stops if and only if it arrives at a leaf of the decision tree. Let h be the height of the decision tree. We only need to prove that $h = \Omega(n^2/f)$. A leaf of the decision tree is identified by its root-leaf path, a word over $\{\text{Yes}, \text{No}\}$ of length at most h and with at most kf *Yes*s.¹⁵ The total number of leaves in the decision tree is at most

$$\sum_{0 \leq j \leq kf} \binom{h}{j} \leq 2 \cdot \binom{h}{kf} \leq \frac{2h^{kf}}{(kf)!}.$$

On the other hand, the number of leaves in the decision tree is the number of instances, which is $(k!)^f$. Therefore,

$$(k!)^f \leq \frac{2h^{kf}}{(kf)!}.$$

Using Stirling's formula, we have $h = \Omega(k^2 f) = \Omega(n^2/f)$. □

7 OPEN QUESTION

The central open question is the following: *Is there an algorithm for graph reconstruction using a near-linear number of queries to a distance oracle?* We believe that the resolution of this question in either direction would lead to new techniques on metric embeddings and the shortest paths structure of degree-bounded graphs, which are both considered very important by the community.

¹⁵We assume that the length of this path is exactly h by appending unnecessary *No*'s.

ACKNOWLEDGMENTS

We thank Fabrice Benhamouda, Jacob Holm, Mathias Bæk Tejs Knudsen, and Mikkel Thorup for discussions. We thank Artur Czumaj, Cyril Gavoille, Philip N. Klein, and anonymous reviewers for useful comments and suggestions on a preliminary version of the article.

REFERENCES

- [1] Mikkel Abrahamsen, Greg Bodwin, Eva Rotenberg, and Morten Stöckel. 2016. Graph reconstruction with a betweenness oracle. In *33rd Symposium on Theoretical Aspects of Computer Science (STACS'16)*, Nicolas Ollinger and Heribert Vollmer (Eds.). Vol. 47. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
- [2] Dimitris Achlioptas, Aaron Clauset, David Kempe, and Cristopher Moore. 2009. On the bias of traceroute sampling: Or, power-law degree distributions in regular graphs. *J. ACM* 56, 4, Article 21 (2009), 21:1–21:28.
- [3] Zuzana Beerliova, Felix Eberhard, Thomas Erlebach, Alexander Hall, Michael Hoffmann, Matús Mihal'ak, and L. Shankar Ram. 2006. Network discovery and verification. *IEEE J. Sel. Areas Commun.* 24, 12 (2006), 2168–2181.
- [4] Jean R. S. Blair and Barry Peyton. 1993. An introduction to chordal graphs and clique trees. In *Graph Theory and Sparse Matrix Computation*. Springer, 1–29.
- [5] Rui Castro, Mark Coates, Gang Liang, Robert Nowak, and Bin Yu. 2004. Network tomography: Recent developments. *Statist. Sci.* 19 (2004), 499–517.
- [6] Gary Chartrand and Frank Harary. 1967. Planar permutation graphs. *Ann. l'Institut Henri Poincaré (B) Prob. Statist.* 3, 4 (1967), 433–438.
- [7] F. Chung, M. Garrett, R. Graham, and D. Shallcross. 2001. Distance realization problems with applications to internet tomography. *J. Comput. Syst. Sci.* 63 (2001), 432–448.
- [8] Luca Dall'Asta, Ignacio Alvarez-Hamelin, Alain Barrat, Alexei Vázquez, and Alessandro Vespignani. 2006. Exploring networks with traceroute-like probes: Theory and simulations. *Theor. Comput. Sci.* 355, 1 (2006), 6–24.
- [9] Jotun J. Hein. 1989. An optimal algorithm to reconstruct trees from additive distance data. *Bull. Math. Biol.* 51, 5 (1989), 597–603.
- [10] Shinichi Honiden, Michael E. Houle, and Christian Sommer. 2009. Balancing graph voronoi diagrams. In *International Symposium on Voronoi Diagrams*. IEEE, 183–191.
- [11] David S. Johnson. 1974. Approximation algorithms for combinatorial problems. *J. Comput. Syst. Sci.* 9, 3 (1974), 256–278.
- [12] Sampath Kannan, Claire Mathieu, and Hang Zhou. 2015. Near-linear query complexity for graph inference. In *International Colloquium on Automata, Languages and Programming*. Springer, 773–784.
- [13] Valerie King, Li Zhang, and Yunhong Zhou. 2003. On the complexity of distance-based evolutionary tree reconstruction. In *Symposium on Discrete Algorithms*. SIAM, 444–453.
- [14] Claire Mathieu and Hang Zhou. 2013. Graph reconstruction via distance oracles. In *International Colloquium on Automata, Languages and Programming*. Springer, 733–744.
- [15] Brendan D. McKay and Nicholas C. Wormald. 1991. Asymptotic enumeration by degree sequence of graphs with degrees $o(n^{1/2})$. *Combinatorica* 11, 4 (1991), 369–382.
- [16] Bruce A. Reed. 2003. Algorithmic aspects of tree width. In *Recent Advances in Algorithms and Combinatorics*. Springer, 85–107.
- [17] Lev Reyzin and Nikhil Srivastava. 2007. Learning and verifying graphs using queries with a focus on edge counting. In *Algorithmic Learning Theory*. Springer, 285–297.
- [18] Sandeep Sen and V. N. Muralidhara. 2010. The covert set-cover problem with application to network discovery. In *International Workshop on Algorithms and Computation*. Springer, 228–239.
- [19] Fabien Tarissan, Matthieu Latapy, and Christophe Prieur. 2009. Efficient measurement of complex networks using link queries. In *INFOCOM Workshops*. IEEE, 254–259.
- [20] Mikkel Thorup and Uri Zwick. 2001. Compact routing schemes. In *Symposium on Parallel Algorithms and Architectures*. ACM, 1–10.
- [21] Michael S. Waterman, Temple F. Smith, M. Singh, and W. A. Beyer. 1977. Additive evolutionary trees. *J. Theor. Biol.* 64, 2 (1977), 199–213.

Received February 2017; revised October 2017; accepted March 2018