Near-Linear Query Complexity for Graph Inference

Sampath Kannan[1], Claire Mathieu[2], and Hang Zhou[2(✉)]

[1] Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA, USA
kannan@cis.upenn.edu
[2] Département d'Informatique UMR CNRS 8548, École Normale Supérieure, Paris, France
{cmathieu,hangzhou}@di.ens.fr

Abstract. How efficiently can we find an unknown graph using distance or shortest path queries between its vertices? Let $G = (V, E)$ be a connected, undirected, and unweighted graph of bounded degree. The edge set E is initially unknown, and the graph can be accessed using a *distance oracle*, which receives a pair of vertices (u, v) and returns the distance between u and v. In the *verification* problem, we are given a hypothetical graph $\hat{G} = (V, \hat{E})$ and want to check whether G is equal to $\hat{G}$. We analyze a natural greedy algorithm and prove that it uses $n^{1+o(1)}$ distance queries. In the more difficult *reconstruction* problem, $\hat{G}$ is not given, and the goal is to find the graph G. If the graph can be accessed using a *shortest path oracle*, which returns not just the distance but an actual shortest path between u and v, we show that extending the idea of greedy gives a reconstruction algorithm that uses $n^{1+o(1)}$ shortest path queries. When the graph has bounded treewidth, we further bound the query complexity of the greedy algorithms for both problems by $\tilde{O}(n)$. When the graph is chordal, we provide a randomized algorithm for reconstruction using $\tilde{O}(n)$ distance queries.

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. The network reconstruction problem has been studied extensively [1,2,5,6,10,12]. Sometimes we have some idea of what the network should be like, based perhaps on its state at some past time, and we want to check whether our image of the network is correct. This is network *verification* and has received

The full version of the paper is available on the authors' websites.

M.M. Halldórsson et al. (Eds.): ICALP 2015, Part I, LNCS 9134, pp. 773–784, 2015.
DOI: 10.1007/978-3-662-47672-7_63

attention recently [2,3,6]. This is an important task for routing, error detection, or ensuring service-level agreement (SLA) compliance, etc. For example, Internet service providers (ISPs) offer their customers services that require quality of service (QoS) 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 and autonomous system (AS) level, where the set of routers (ASs) and their physical connections (peering relations) 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 the shortest one, i.e., minimizes the hop distance between that pair. In our first theoretical model, we assume that we have access to any pair of vertices and get in return their shortest path in the graph. Sometimes routers block traceroute and mtrace requests (e.g., due to privacy and security concerns), thus the inference of topology can only rely on delay information. In our second theoretical model, we assume that we get in return the hop distance between a pair of vertices. The second model was introduced in [10].

Graph inference using queries that reveal partial information has been studied extensively in different contexts, independently stemming from a number of applications. Beerliova et al. [2] studied network verification and reconstruction using an oracle, which, upon receiving a node q, returns all shortest paths from q to all other nodes, instead of one shortest path between a pair of nodes as in our first model. Erlebach et al. [6] studied network verification and reconstruction using an oracle which, upon receiving a node q, returns the distances from q to all other nodes in the graph, instead of the distance between a pair of nodes as in our second model. They showed that minimizing the number of queries for verification is NP-hard and admits an $O(\log n)$-approximation algorithm. In the *network realization* problem, we are given the distances between certain pairs of vertices and asked to determine the sparsest graph (in the unweighted case) or the graph of least total weight that realizes these distances. This problem was shown to be NP-hard [4]. In evolutionary biology, a well-studied problem is reconstructing evolutionary trees, thus the hidden graph has a tree structure. See for example [7,9,11]. One may query a pair of species and get in return the distance between them in the (unknown) tree. In our reconstruction problem, we allow the hidden graph to have an arbitrarily connected topology, not necessarily a tree structure.

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*

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

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. We focus on query complexity, while all our algorithms are of polynomial time and space. We note that $O(n^2)$ queries are enough for both reconstruction and verification via a distance oracle or a shortest path oracle: we only need to query every pair of vertices.

Let Δ denote the maximum degree of any vertex in the graph G. Unless otherwise stated, we assume that Δ is bounded, which is reasonable for real networks that we want to reconstruct or verify. Indeed, when Δ is $\Omega(n)$, both reconstruction and verification require $\Omega(n^2)$ distance or shortest path queries.

Let us focus on bounded degree graphs. It is not hard to see that $\Omega(n)$ distance or shortest path queries are required. The central question in this line of work is therefore: **Is the query complexity linear, quadratic, or somewhere in between?** In [10], Mathieu and Zhou provide a first answer: the query complexity for reconstruction via a distance oracle is subquadratic: $\tilde{O}(n^{3/2})$. In this paper, we show that the query complexity for reconstruction via a shortest path oracle or verification via either oracle is near-linear: $n^{1+o(1)}$. It is open whether there is an algorithm for reconstruction using a near-linear number of distance queries.

1.2 Our Results

Verification

Theorem 1. *For graph verification using a distance oracle, there is a deterministic algorithm (Algorithm 1) with query complexity* $n^{1+O\left(\sqrt{(\log\log n + \log\Delta)/\log n}\right)}$, *which is* $n^{1+o(1)}$ *when the maximum degree* $\Delta = n^{o(1)}$. *If the graph has treewidth* w, *the query complexity can be further bounded by* $O(\Delta(\Delta + w\log n)n\log^2 n)$, *which is* $\tilde{O}(n)$ *when* Δ *and* w *are* $O(\text{polylog } n)$.

The main task for verification is to confirm the *non-edges* of the graph. Algorithm 1 is greedy: every time it makes a query that confirms the largest number of non-edges that are not yet confirmed. To analyze the algorithm, first, we show that its query complexity is roughly $\ln n$ times the optimal number of queries OPT for verification. This is based on a reduction to the Set-Cover problem, see Section 3.1. It only remains to bound OPT.

[2] Expected query complexity in the case of randomized algorithms.

Table 1. Results (for bounded degree graphs). New results are in bold.

Objective	*Query complexity*
verification via either oracle	$\boldsymbol{n^{1+o(1)}}$ **bounded treewidth:** $\tilde{O}(n)$ **(Thm 1, Cor 2, and Thm 3)**
reconstruction via a shortest path oracle	
reconstruction via a distance oracle	$\tilde{O}(n^{3/2})$ [10] $\boldsymbol{\Omega(n \log n / \log\log n)}$ **(Thm 5)** outerplanar: $\tilde{O}(n)$ [10] **chordal:** $\boldsymbol{\tilde{O}(n)}$ **(Thm 4)**

To bound OPT and get the first statement in Theorem 1, it is enough to prove the desired bound for a different verification algorithm. This algorithm is a more sophisticated recursive version of the algorithm in [10]. Recursion is a challenge because, when we query a pair (u, v) in a recursive subgraph, the oracle returns the distance between u and v in the entire graph, not just within the subgraph. Thus new ideas are introduced for the algorithmic design. See Section 3.3.

To show the second statement in Theorem 1, similarly, we design another recursive verification algorithm with query complexity $\tilde{O}(n)$ for graphs of bounded treewidth. The algorithm uses some bag of a tree decomposition to separate the graph into balanced subgraphs, and then recursively verifies each subgraph. The same obstacle to recursion occurs. Our approach here is to add a few weighted edges to each subgraph in order to preserve the distance metric. The complete proof is in the full version of the paper.

We note that each query to a distance oracle can be simulated by the same query to a shortest path oracle. So from Theorem 1, we have:

Corollary 2. *For graph verification using a shortest path oracle, Algorithm 1 achieves the same query complexity as in Theorem 1.*

Reconstruction

Theorem 3. *For graph reconstruction using a shortest path oracle, there is a deterministic algorithm (Algorithm 4) that achieves the same query complexity as in Theorem 1.*

The key is to formulate this problem as a problem of verification using a distance oracle, so that we get the same query complexity as in Theorem 1. We extend the idea of greedy in Algorithm 1, and 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 realization here is that reconstruction can be viewed as the verification of a dynamically changing graph. See Section 4.

Theorem 4. *For reconstruction of chordal graphs using a distance oracle, there is a randomized algorithm with query complexity* $O\big(\Delta^3 2^{\Delta} \cdot n(2^{\Delta} + \log^2 n) \log n\big)$, *which is* $\tilde{O}(n)$ *when the maximum degree* Δ *is* $O(\log \log n)$.

The algorithm in Theorem 4 first finds a separator using random sampling and statistical estimates, as in [10]. Then it partitions the graph into subgraphs with respect to this separator and recurses on each subgraph. However, the separator here is a clique instead of an edge in [10] for outerplanar graphs. Thus the main difficulty is to design and analyze a more general tool for partitioning the graph. The proof of the theorem is in the full version of the paper.

Lower Bounds. For graphs of bounded degree, both reconstruction and verification require $\Omega(n)$ distance or shortest path queries. In addition, there is a slightly better lower bound for reconstruction using a distance oracle, as in the following theorem.

Theorem 5. *For graph reconstruction using a distance oracle, assuming the maximum degree* $\Delta \geq 3$ *is such that* $\Delta = o\left(n^{1/2}\right)$, *any algorithm has query complexity* $\Omega(\Delta n \log n / \log \log n)$.

The proof of Theorem 5 is in the full version of the paper.

2 Notation

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 $v \in V$, let $N(v) = \{u \in V : \delta(u, v) \leq 1\}$ and let $N_2(v) = \{u \in V : \delta(u, v) \leq 2\}$. We define $\hat{\delta}$, $\hat{N}$, and $\hat{N}_2$ similarly with respect to the graph $\hat{G}$.

For a graph $G = (V, E)$, a distinct pair of vertices $uv \in V^2$ 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 a vertex $s \in V$ and a subset $T \subseteq V$, define $\textsc{Query}(s, T)$ as $\textsc{Query}(s, t)$ for every $t \in T$. For subsets $S, T \subseteq V$, define $\textsc{Query}(S, T)$ as $\textsc{Query}(s, t)$ for every $(s, t) \in S \times T$.

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

3 Proof of Theorem 1

3.1 Greedy Algorithm

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 second part is

called *non-edge verification*. In the second part, we assume that the first part is already done, which guarantees that $\hat{E} \subseteq E$. For graphs of bounded degree, the first part requires only $O(\Delta n)$ queries, thus the focus is on non-edge verification.

Theorem 6. *For graph verification using a distance oracle, there is a deterministic greedy algorithm (Algorithm 1) that uses at most* $\Delta n + (\ln n + 1) \cdot OPT$ *queries, where* OPT *is the optimal number of queries for non-edge verification.*

Now we prove Theorem 6. 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 $S_{u,v} \subseteq \widehat{NE}$ as follows:

$$S_{u,v} = \left\{ ab \in \widehat{NE} : \hat{\delta}(u, a) + \hat{\delta}(b, v) + 1 < \hat{\delta}(u, v) \right\}. \tag{1}$$

The following two lemmas relate the sets $S_{u,v}$ with non-edge verification.

Lemma 7. *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. □

Lemma 8. *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 of $\hat{E}$: 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. □

From Lemmas 7 and 8, 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 instance can be solved using the well-known greedy algorithm [8], which gives a $(\ln n + 1)$-approximation. Hence our greedy algorithm for verification (Algorithm 1). 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 optimal number of queries. This proves Theorem 6.

3.2 Bounding *OPT* to Prove Theorem 1

From Theorems 6, in order to prove Theorem 1, we only need to bound OPT, as in the following two theorems.

Theorem 9. *For graph verification using a distance oracle, the optimal number of queries* OPT *for non-edge verification is* $n^{1+O\left(\sqrt{(\log\log n + \log\Delta)/\log n}\right)}$.

Algorithm 1. Greedy Verification

```
1: procedure VERIFY(Ĝ)
2:     for uv ∈ Ê do QUERY(u, v)
3:     Y ← ∅
4:     while Ê ∪ Y does not cover all vertex pairs do
5:         choose (u, v) that maximizes |S_{u,v} \ Y|        ▷ S_{u,v} defined in Equation (1)
6:         QUERY(u, v)
7:         Y ← Y ∪ S_{u,v}
```

Theorem 10. *For graph verification using a distance oracle, if the graph has treewidth w, then the optimal number of queries OPT for non-edge verification is $O(\Delta(\Delta + w \log n) n \log n)$.*

Theorem 1 follows trivially from Theorems 6, 9, and 10, by noting that both Δ and $\log n$ are smaller than $n^{\sqrt{(\log\log n + \log \Delta)/\log n}}$. The proof of Theorem 9 is in Section 3.3, and the proof of Theorem 10 is in the full version of the paper.

3.3 Proof of Theorem 9

To show Theorem 9, we provide a recursive algorithm for non-edge verification with the query complexity in the theorem statement. As in [10], the algorithm selects a set of *centers* partitioning V into Voronoi cells and expands them slightly so as to cover all edges of G. But unlike [10], instead of using exhaustive search inside each cell, the algorithm verifies each cell recursively. The recursion is a challenge because the distance oracle returns the distance in the entire graph, not in the cell. Straightforward attempts to use recursion lead either to subcells that do not cover all edges of the cell, or to excessively large subcells. Our approach is to allow selection of centers *outside* the cell, while still limiting the subcells to being contained *inside* the cell (Figure 1). This simple but subtle setup is one novelty of the algorithmic design.

The verification algorithm uses the function SUBSET-CENTERS (Algorithm 2), 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 CENTER algorithm by Thorup and Zwick [13]: when the subset U equals V, the SUBSET-CENTERS algorithm becomes their CENTER algorithm. For every $w \in V$, we define w's *cluster* in the graph G as $C_A(w) = \{v \in V : \delta(w, v) < \delta(A, v)\}$. We note that if $w \in A$, then $C_A(w) = \emptyset$, since $\delta(w, v) \geq \delta(A, v)$, for every $v \in V$. Similarly, we define w's cluster in 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 3.1 in [13].

Algorithm 2. Finding Centers for a Subset

```
function SUBSET-CENTERS(Ĝ, U, s)
    A ← ∅
    while there exists w ∈ V such that |Ĉ(w) ∩ U| > 4|U|/s do
        W ← {w ∈ V : |Ĉ(w) ∩ U| > 4|U|/s}
        Add each element of W to A with probability min(s/|W|, 1)
    return A
```

Lemma 11. *The function* SUBSET-CENTERS *(Algorithm 2) outputs a set* $A \subseteq V$*, such that, with probability at least* $1/2$*, we have* $|A| \leq 4s \log n$ *and* $|\hat{C}(w) \cap U| \leq 4|U|/s$ *for every* $w \in V$*. It uses no queries and its running time is polynomial.*

Next, we design a recursive algorithm for non-edge verification. Let $U \subseteq V$ represent the set of vertices for which we are currently verifying the induced subgraph. Verifying that every non-edge of $\hat{G}[U]$ is a non-edge of $G[U]$ 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* D_a as

$$D_a = \left(\bigcup \{C(b) : b \in N_2(a)\} \cup N_2(a)\right) \cap U. \tag{2}$$

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

$$\hat{D}_a = \left(\bigcup \left\{\hat{C}(b) : b \in \hat{N}_2(a)\right\} \cup \hat{N}_2(a)\right) \cap U. \tag{3}$$

The following lemma is a trivial extension of Lemma 3 in [10].

Lemma 12. $\bigcup_{a \in A} G[D_a]$ *covers every edge of* $G[U]$*.*

From Lemma 12, in order 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 $a \in A$. So we can 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 first 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. This is shown in Lemma 13, which is the main novelty of the algorithmic design.

Lemma 13. *Assume that* $\hat{E} \subseteq E$*. If* $\delta(u, v) = \hat{\delta}(u, v)$ *for every pair* (u, v) *from* $\bigcup_{a \in A} \hat{N}_2(a) \times U$*, then* $D_a = \hat{D}_a$ *for all* $a \in A$*.*

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

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

Algorithm 3. Recursive Verification

```
1: procedure VERIFY-SUBGRAPH(Ĝ, U)
2:     if |U| > n_0 then
3:         repeat
4:             A ← SUBSET-CENTERS(Ĝ, U, s)
5:         until |A| ≤ 4s log n and |Ĉ(w) ∩ U| ≤ 4|U|/s for every w ∈ V
6:         for a ∈ A do
7:             QUERY(N̂_2(a), U)
8:             VERIFY-SUBGRAPH(Ĝ, D̂_a)          ▷ D̂_a defined in Equation (3)
9:     else
10:        QUERY(U, U)
```

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

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:

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

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

The recursive algorithm for non-edge verification is in Algorithm 3. It 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 1. The parameters n_0 and s are defined later. Correctness of the algorithm follows from Lemmas 12 and 13.

Now we bound the query complexity of VERIFY-SUBGRAPH$(\hat{G}, V)$. To provide intuition, we analyze an algorithm of 4 recursive levels, and show that its query complexity is $\tilde{O}(n^{4/3})$. The complete proof of the complexity stated in Theorem 9 is in the full version of the paper.

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 VERIFY-SUBGRAPH$(\hat{G}, U)$ where $|U| > n_0$. Let $A \subseteq V$ be the centers at the end of the **repeat** loop. By Lemma 11, the expected number of **repeat** loops is constant. For every $a \in A$, $\hat{N}_2(a)$ has constant size, since the graph has bounded degree. 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 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

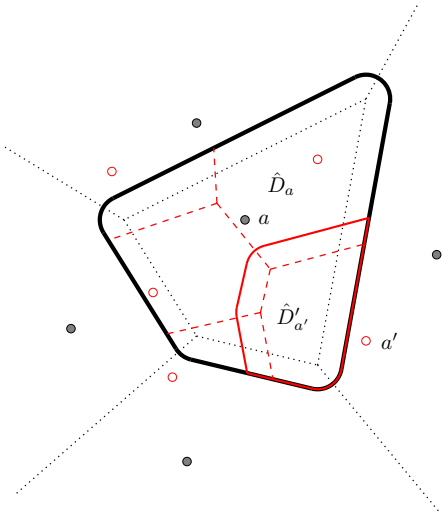

Fig. 1. Two levels of recursive calls of VERIFY-SUBGRAPH($\hat{G}, V$): The solid points are top-level centers returned by SUBSET-CENTERS($\hat{G}, V, s$). The dotted lines indicate the partition of V into Voronoi cells by those centers. The region inside the outer curve represents the extended Voronoi cell $\hat{D}_a$ of a center a. On the second level of the recursive call for $\hat{D}_a$, the hollow points are the centers returned by 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. The region inside the inner curve represents the extended Voronoi cell $\hat{D}'_{a'}$ of a second-level center a'.

with subset U uses $\tilde{O}(n^{1/3} \cdot |U|)$ queries. Therefore, the overall query complexity is $\tilde{O}(n^{4/3})$.

Remark. *The recursive algorithm (Algorithm 3) can be used for verification by itself. However, we only use its query complexity to provide guarantee for the greedy algorithm (Algorithm 1), because the greedy algorithm is much simpler.*

4 Proof of Theorems 3

The algorithm (Algorithm 4) constructs an increasing set X of edges so that in the end $X = E$. At any time, the candidate graph is X.[3] Initially, X is the union of the shortest paths given as answers by $n - 1$ queries, so that X is a connected subgraph spanning V. 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 $(u, v) \in V^2$,

$$S^X_{u,v} = \big\{ab \in \text{non-edges of } X : \delta_X(u, a) + \delta_X(b, v) + 1 < \delta_X(u, v)\big\}. \quad (4)$$

[3] We identify X with the subgraph induced by the edges of X.

Algorithm 4. Greedy Reconstruction

```
1: procedure RECONSTRUCT(V)
2:     u_0 ← an arbitrary vertex
3:     for u ∈ V \ {u_0} do QUERY(u, u_0) to get a shortest u-to-u_0 path
4:     X ← the union of the above paths,   Y ← ∅
5:     while X ∪ Y does not cover all vertex pairs do
6:         choose (u, v) that maximizes |S^X_{u,v} \ Y|        ▷ S^X_{u,v} defined in Equation (4)
7:         QUERY(u, v) to get a shortest u-to-v path
8:         if δ_G(u, v) = δ_X(u, v) then
9:             Y ← Y ∪ S^X_{u,v}
10:        else
11:            e ← some edge of the above u-to-v path that is not in X
12:            X ← X ∪ {e}
13:    return X
```

This is similar to $S_{u,v}$ defined in Equation (1). From Lemma 7, the pairs in $S^X_{u,v}$ 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 $S^X_{u,v} \setminus Y$. As a consequence, either all pairs in $S^X_{u,v} \setminus Y$ are confirmed as non-edges of G, or $\delta_G(u,v) \neq \delta_X(u,v)$, and in that case, the query reveals an edge along a shortest u-to-v path in G that is not in X; we then add this edge to X.

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

For the query complexity, first, consider the queries that lead to $\delta_G(u,v) \neq \delta_X(u,v)$. For each such query, an edge is added to X. This can happen at most $|E| \leq \Delta n$ times, because 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. For each such query, the size of R decreases by $|S^X_{u,v} \setminus Y|$. To lower bound $|S^X_{u,v} \setminus Y|$, 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. By Theorem 9, $|T|$ is at most $f(n,\Delta) = n^{1+O\left(\sqrt{(\log\log n + \log\Delta)/\log n}\right)}$. By Lemma 8, the sets $S^X_{u,v}$ for all pairs $(u,v) \in T$ together cover $R \cup Y$, hence R. Therefore, at least one of these pairs satisfies

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

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 $O(\Delta n + \log n \cdot f(n,\Delta))$. Thus we obtained the same query bound as in the first statement of Theorem 1. To prove the query bound for graphs of treewidth w as in the second statement, the

analysis is identical as above, except that $f(n, \Delta) = O(\Delta(\Delta + w \log n)n \log n)$, which comes from Theorem 10.

Remark. *Note that the above proof depends crucially on the fact that $f(n, \Delta)$ is a uniform bound on the number of distance queries for the non-edge verification of* any *n-vertex graph of maximum degree Δ. Thus, even though the graph X changes during the course of the algorithm because of queries (u, v) such that $\delta_G(u, v) \neq \delta_X(u, v)$, each query for which the distance in G and the current X are equal confirms $1/f(n, \Delta)$ fraction of non-edges.*

Acknowledgments. We thank Uri Zwick for Theorem 5. We thank Fabrice Benhamouda, Mathias Bæk Tejs Knudsen, Mikkel Thorup, and Jacob Holm for discussions. The first author was partially supported by NSF Grant NRI 1317788. The last two authors were partially supported by the French *Agence Nationale de la Recherche* under reference ANR-12-BS02-005 (RDAM project).

References

1. Achlioptas, D., Clauset, A., Kempe, D., Moore, C.: On the bias of traceroute sampling: or, power-law degree distributions in regular graphs. Journal of the ACM (JACM) **56**(4), 21 (2009)
2. Beerliova, Z., Eberhard, F., Erlebach, T., Hall, A., Hoffmann, M., Mihaľák, M., Shankar Ram, L.: Network discovery and verification. In: Kratsch, D. (ed.) WG 2005. LNCS, vol. 3787, pp. 127–138. Springer, Heidelberg (2005)
3. Castro, R., Coates, M., Liang, G., Nowak, R., Yu, B.: Network tomography: recent developments. Statistical Science **19**, 499–517 (2004)
4. Chung, F., Garrett, M., Graham, R., Shallcross, D.: Distance realization problems with applications to internet tomography. Journal of Computer and System Sciences **63**, 432–448 (2001)
5. Dall'Asta, L., Alvarez-Hamelin, I., Barrat, A., Vázquez, A., Vespignani, A.: Exploring networks with traceroute-like probes: Theory and simulations. Theoretical Computer Science **355**(1), 6–24 (2006)
6. Erlebach, T., Hall, A., Hoffmann, M., Mihaľák, M.: Network discovery and verification with distance queries. In: Calamoneri, T., Finocchi, I., Italiano, G.F. (eds.) CIAC 2006. LNCS, vol. 3998, pp. 69–80. Springer, Heidelberg (2006)
7. Hein, J.J.: An optimal algorithm to reconstruct trees from additive distance data. Bulletin of Mathematical Biology **51**(5), 597–603 (1989)
8. Johnson, D.S.: Approximation algorithms for combinatorial problems. Journal of computer and system sciences **9**(3), 256–278 (1974)
9. King, V., Zhang, L., Zhou, Y.: On the complexity of distance-based evolutionary tree reconstruction. In: SODA, pp. 444–453. SIAM (2003)
10. Mathieu, C., Zhou, H.: Graph reconstruction via distance oracles. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M., Peleg, D. (eds.) ICALP 2013, Part I. LNCS, vol. 7965, pp. 733–744. Springer, Heidelberg (2013)
11. Reyzin, L., Srivastava, N.: On the longest path algorithm for reconstructing trees from distance matrices. Information processing letters **101**(3), 98–100 (2007)
12. Tarissan, F., Latapy, M., Prieur, C.: Efficient measurement of complex networks using link queries. In: INFOCOM Workshops, pp. 254–259. IEEE (2009)
13. Thorup, M., Zwick, U.: Compact routing schemes. In: Symposium on Parallel Algorithms and Architectures, pp. 1–10. ACM (2001)