Graph Verification with a Betweenness Oracle

Mano Vikash Janardhanan MJANAR2@UIC.EDU
Department of Mathematics, Statistics, and Computer Science
University of Illinois at Chicago
Chicago, IL 60607

Editors: Steve Hanneke and Lev Reyzin

Abstract

In this paper, we examine the query complexity of verifying a hidden graph G with a betweenness oracle. Let $G = (V, E)$ be a hidden graph and $\hat{G} = (V, \hat{E})$ be a known graph. V and $\hat{E}$ are known and E is not known. The graphs are connected, unweighted and have bounded maximum degree Δ. The task of the graph verification problem is to verify that $E = \hat{E}$. We have access to G through a black-box betweenness oracle. A betweenness oracle returns whether a vertex lies along a shortest path between two other vertices. The betweenness oracle nicely captures many real-world problems. We prove that graph verification can be done using $n^{1+o(1)}$ betweenness queries. Surprisingly, this matches the state of the art for the graph verification problem with the much stronger distance oracle. We also prove that graph verification requires $\Omega(n)$ betweenness queries – a matching lower bound.

1. Introduction and previous work

Graph learning and graph verification problems arise in various situations. Consider the internet graph where vertices correspond to routers and edges correspond to physical connections. It is often the case that one knows the set of vertices in the network (routers) but does not know the edges (physical connections). To learn the physical connections, one has to use computer network diagnostic tools (such as traceroute and mtrace) which give information about the shortest paths in the network. Assume one has access to the internet graph through such an oracle. A natural question to ask is what is the best way to use the oracle to find the physical connections between the routers. In other words, what is the minimum number of queries needed to learn the edge set of the graph?

Graph learning and graph verification problems are well-studied problems in the area of graph algorithms. In both these problems, there is a hidden graph which one has access to through a black-box oracle. In graph learning problems, the task is to use this oracle to learn the edge set of the graph. Graph learning problems are also referred to as the graph reconstruction problems. Graph learning problems has been studied extensively (Alon and Asodi, 2005; Alon et al., 2004; Angluin and Chen, 2008; Beerliova et al., 2006; Dall'Asta et al., 2006; Hein, 1989; Reyzin and Srivastava, 2007a).

In graph verification problems, one is given another input graph and the task is to verify that the graph one has access to through an oracle is the same as the input graph. Graph verification problems have received a lot of attention recently (Beerliova et al., 2006; Erlebach et al., 2006; Kannan et al., 2015; Mathieu and Zhou, 2013; Reyzin and Srivastava,

2007a). Graph verification problems have many applications for Internet Service Providers (ISPs). The ISPs have knowledge about the structure of the network based on past information. And at any point of time, they might wish to verify that there is no fault in the network. In any internet protocol network, fault detection methods are critical for providing quality of service guarantees.

Some of the oracles studied in literature include the distance oracle by Kannan et al. (2015), layered-graph oracle by Beerliova et al. (2006), edge detection and edge counting oracle by Reyzin and Srivastava (2007a), etc. Among these, the most natural and well-studied one is the distance oracle (Erlebach et al., 2006; Kannan et al., 2015; Mathieu and Zhou, 2013; Reyzin and Srivastava, 2007b). The distance oracle takes as input two vertices and returns the distance between the two vertices. This oracle nicely captures applications in computational biology. One example in computational biology is the problem of learning evolutionary trees (Hein, 1989; King et al., 2003; Waterman et al., 1977). Researchers can obtain the distance between two species in an unknown evolutionary tree. This can be thought of as making a distance query. But each query requires a lot of research effort. Hence the objective is to learn the evolutionary tree with the minimum number of queries. In general, for both graph learning and graph verification problems, we assume that making queries is costly. Hence, we are concerned with optimizing the worst-case query complexity herein.

In this paper, we look at the query complexity of the graph verification problem with a betweenness oracle. The betweenness oracle, introduced by Abrahamsen et al. (2016), returns whether a given vertex lies along a shortest path between two other vertices. When the graphs are connected, unweighted, and have bounded maximum degree Δ, we prove the worst-case query complexity has an upper bound of $n^{1+o(1)}$. We also prove a lower bound of $\Omega(n)$. The betweenness oracle also has many natural applications in the study of evolutionary trees. For evolutionary trees, the method for calculating evolutionary distance is error-prone. If we use the betweenness oracle approach, we only need to query whether one species lies in the shortest path connecting two other species. Such a query is more natural for evolutionary trees.

Intuitively, the betweenness oracle is expected to be much weaker than the distance oracle. Notice that a betweenness query can be simulated by three distance queries. Let x, y and z be vertices of a graph and $d(\cdot, \cdot)$ denote the distance between two vertices in the graph. Then, $d(x, y) + d(y, z) = d(x, z)$ if and only if y lies on a shortest path between x and z. Conversely, it is easy to see that in a path graph, one needs $\Omega(n)$ betweenness queries to simulate a single distance query.

For degree-bounded graphs, Abrahamsen et al. (2016) showed that the graph learning problem with a betweenness oracle has the same worst-case query complexity as its analogue with a distance oracle. However, the problem of verifying a graph with a betweenness oracle remained open. In this paper, we give matching lower and upper bounds for this problem. The main result of Abrahamsen et al. (2016) is the following:

Theorem 1 *Learning a graph can be done with $\tilde{O}(n^{3/2} \cdot \Delta^4)$ betweenness queries, where Δ is the maximum degree of the graph.*

1.1. The problem

The hidden graph to be verified is denoted as $G = (V, E)$. A pair of vertices $(u, v) \in V^2$ is called a non-edge if $(u, v) \notin E$. This can be also denoted as $(u, v) \in NE$ where NE is the set of non-edges of G. Similarly, the given graph $\hat{G} = (V, \hat{E})$ has non-edge set $\hat{NE}$ defined in a similar way.

In graph learning problems, we are given an oracle access to G, and the task is to determine E. In graph verification problems we are given $\hat{G}$ and an oracle access to G, and asked to verify that $E = \hat{E}$.

Given a subset $U \subset V$, $G[U]$ is the subgraph induced by U. For the rest of the paper, we assume that the graph is connected, unweighted and has maximum degree Δ. We have access to G through a betweenness oracle.

Definition 2 *A **betweenness query** denoted as* $\text{bet}_G(u, v, w)$ *is true if and only if there exists a shortest path in G between u and w that passes through v. Often, the subscript will be dropped when G is clear from context.*

We prove matching lower bound and upper bound for the query complexity of the graph verification problem with a betweenness oracle.

2. Lower bound

In this section, we consider an instance of the graph verification problem where the input graph $\hat{G}$ is a caterpillar tree and the hidden graph G is a slight modification of $\hat{G}$. Then, we show that $\Omega(n)$ betweenness queries are required to catch this modification.

Theorem 3 *Graph verification requires $\Omega(n)$ betweenness queries.*

Proof Let $\hat{G}$ be the caterpillar tree with spine from v_1 to $v_{n/2}$ and one vertex connected to each vertex in the spine (Figure 1). $|\hat{G}| = n - 1$ and n is an even number. Consider a new graph H obtained from $\hat{G}$ by doing the following (Figure 2):

- Fix some $i \in [1, n/2 - 1]$
- Delete the edge between v_i and v_{i+1}
- Add an edge between $v_{n/2+i}$ and v_{i+1}

Suppose the hidden graph G is H (Figure 2). The betweenness queries that give away the difference between $\hat{G}$ and H contain $v_{n/2+i}$ as one of its three arguments in $\text{bet}_G(\cdot, \cdot, \cdot)$. There are $n/2 - 1$ vertices of the form $v_{n/2+i}$ and each query can cover at most 3 of them. This gives the desired lower bound. ■

Observation 1 *When the target graph G has maximum degree Δ, graph learning requires $\Omega(n\Delta \log n)$ betweenness queries. This is because the number of connected graphs with maximum degree Δ is $\Omega(n^{\Omega(\Delta n)})$ (see McKay and Wormald, 1990). Hence, information theoretically, we get the desired lower bound.*

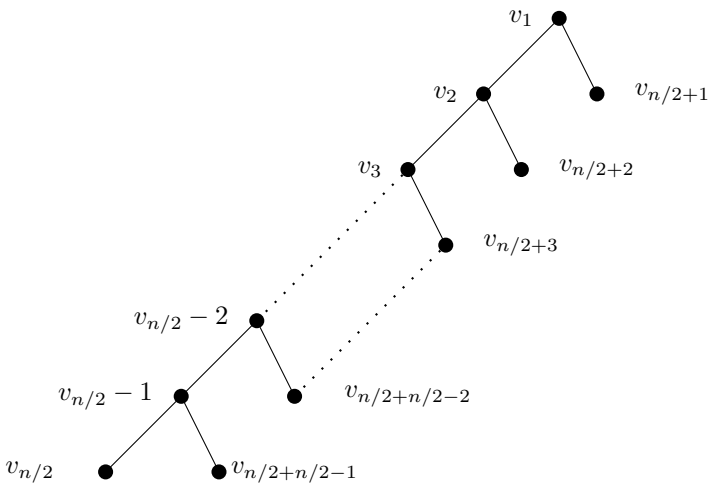

Figure 1: Representation of $\hat{G}$

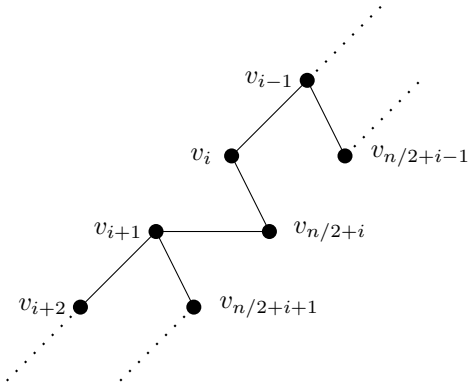

Figure 2: Representation of H

3. Definitions

The full generality of the following definitions are not necessary for this paper. But we do it to maintain consistency with Abrahamsen et al. (2016).

Definition 4 *Let $G = (V, E)$ and $v \in V$. Let $N_i(v)$ denote the set of vertices that are distance i or less from v. Let $N(v) = N_1(v)$. Hence $N(v)$ is the set that contains v and all its neighbours. When $x, y \in V$, let $\delta(x, y)$ denote the distance between x and y in G.*

Definition 5 *Given a graph $G = (V, E)$, a subset $X \subset V$ is said to be **starshaped** with respect to centre $x \in X$ if for all $v \in X$, every shortest path from x to v is entirely contained in X.*

Definition 6 *Given a graph $G = (V, E)$ and a starshaped set $X \subset V$ with centre $x \in X$, a node $v \in X$ is said to be in layer i if $\delta(x, v) = i$. The set of nodes in layer i is denoted $L(x)_i^X$. When $X = V$, the superscript is dropped and written as $L(x)_i$.*

Definition 7 *Given a graph $G = (V, E)$ and a starshaped set $X \subset V$ with centre $x \in X$, a subgraph $\tau(x)_X$ is a **spanning tree** with respect to centre x if it is a tree such that for all $v \in X$, $\tau(x)_X$ contains a shortest path from x to v. Given a starshaped set $X \subset V$ with centre $x \in X$, the subgraph $S(x)_X$ obtained by removing all edges in the same layer $L(x)_i^X$ is called the **shortest path graph** with respect to centre x.*

Definition 8 *Given a starshaped set $X \subset V$ with centre $x \in X$, if $v \in L(x)_i^X$, then u is a **parent** of v with respect to centre x if $u \in N(v) \cap L(x)_{i-1}^X$. This can be written as $u \in p_x(v)$. Note that $p_x(v)$ is a set. Given a starshaped set $X \subset V$ with centre $x \in X$, if $v \in L(x)_i^X$, then u is a **child** of v with respect to centre x if $u \in N(v) \cap L(x)_{i+1}^X$.*

Definition 9 *The **ancestor** relation is the transitive closure of the parent relation and the **descendant** relation is the transitive closure of the child relation. The set of ancestors of a vertex v with respect to centre x is denoted $A_x(v)$ and the set of descendants is denoted $D_x(v)$. The subscript is dropped when the centres x is clear from context.*

For $\hat{G} = (V, \hat{E})$, we define $\hat{\delta}(x, y)$, $\hat{N}_i(v)$, $\hat{N}(v)$, $\hat{L}(x)_i^X$, $\hat{\tau}(x)_X$, $\hat{S}(x)_X$, $\hat{p}_x(v)$, $\hat{A}_x(v)$ and $\hat{D}_x(v)$ in a similar way.

4. Main result

The main result is Theorem 10 and its proof proceeds in two steps- edge verification and non-edge verification. The proof relies on some techniques developed earlier for graph verification with a distance oracle and graph learning with a betweenness oracle. For edge verification, we use some results from Abrahamsen et al. (2016). For non-edge verification, we use some results from Kannan et al. (2015). These results are stated without proof before being used.

Theorem 10 *Let $\hat{G}$ be a connected graph with maximum degree Δ. For graph verification using a betweenness oracle, there is a deterministic algorithm with a query complexity of $n^{1+O\left(\sqrt{(\log\log n + \log\Delta)/\log n}\right)}$. When $\Delta = n^{o(1)}$, this gives us a query complexity of $n^{1+o(1)}$.*

When $\Delta = n^{o(1)}$, Kannan et al. (2015) devised a recursive algorithm that does non-edge verification using $n^{1+o(1)}$ distance queries. To simulate a distance query, we need $\Omega(n)$ betweenness queries. Hence, a brute force generalization of their approach can not give query complexity better than $n^{2+o(1)}$. The main contributions of this paper are the following:

- Edge verification can be done using $O(n\Delta^2)$ betweenness queries. This is proved in Lemma 15.
- We prove that the recursive approach for non-edge verification developed in Kannan et al. (2015) can be implemented using $n^{1+o(1)}$ betweenness queries.

4.1. Edge verification

We start by proving a bound on the number of betweenness queries required for edge verification. Before doing that, we need the following three results (stated without proof) from Abrahamsen et al. (2016).

Lemma 11 (Abrahamsen et al. (2016)) *Every starshaped set X with centre x has a node $s \in X$ with the property*

$$\left\lceil \frac{|X|}{3\Delta} \right\rceil \leq |D(s)| \leq \left\lceil \frac{|X|}{3} \right\rceil$$

Further, $D(s)$ and $(X - D(s)) \cup \{s\}$ are both starshaped with centres s and x respectively.

Lemma 12 (Abrahamsen et al. (2016)) *Let $X \subset V$ be a starshaped set with centre x. One can discover all edges in $G[X]$ using $O(|X|^2)$ betweenness queries.*

Lemma 13 (Abrahamsen et al. (2016)) *Given a starshaped set X with centre x, and the shortest path graph of X, one can decide whether or not there exists an edge between any two nodes u and v in the hidden graph using $O(1)$ betweenness queries.*

For doing edge verification, we start by recursively applying Lemma 11 to partition the edge set of a spanning tree of $\hat{G}$ and then apply Lemma 12 to verify that all edges of the spanning tree of $\hat{G}$ is present in G. Then, we use Lemma 14 to show that G and $\hat{G}$ have the same layer structure. After this, we only need to verify edges within the same layer and edges between adjacent layers. These type of edges require $O(1)$ query per edge.

Lemma 14 (Layer Structure Verification) *Let $\hat{G} = (V, \hat{E})$ be a connected graph. Suppose $\hat{\tau}(x)_V$ is a spanning tree of $\hat{G}$ with respect to centre x and every edge in $\hat{\tau}(x)_V$ has been verified to be present in G. Then, n betweenness queries are sufficient to verify that $L(x)_i^V = \hat{L}(x)_i^V$ for all i.*

Proof To show $L(x)_i^V = \hat{L}(x)_i^V$ for $i \leq k$, we need to establish there is no edge in G going from $\hat{L}(x)_i^V$ to $\hat{L}(x)_{i-s}^V$ for $i \leq k$ and $s > 1$. We prove this by induction on k.

Query $\text{bet}(x, p, v)$ for $v \in \hat{L}(x)_k^V$ and some $p \in \hat{p}_x(v)$. Because every edge in $\hat{\tau}(x)_V$ has been verified to be present in G, $\text{bet}(x, p, v)$ will return false if and only if there is an edge in G from v to some vertex in $\hat{L}(x)_{k-s}^V$ for $s > 1$. Hence, it takes $|\hat{L}(x)_k^V|$ queries to show there is no edge in G from $\hat{L}(x)_k^V$ to $\hat{L}(x)_{i-s}^V$ for $s > 1$. Continuing for all layers takes at most n queries. ■

Lemma 15 *Given $\hat{G}$ and access to G through a betweenness oracle, verifying all edges of $\hat{G}$ are present in G can be done with $O(n\Delta^2)$ betweenness queries.*

Proof We start by proving that every edge of a spanning tree of $\hat{G}$ is present in G. Fix a centre x in $\hat{G}$ and let $\hat{\tau}(x)_V$ be a spanning tree with respect to centre x. Now, we partition the edge set of $\hat{\tau}(x)_V$ by doing the following. Recursively apply Lemma 11 to obtain starshaped sets $\{S_1, S_2, \ldots, S_h\}$ that satisfy the following properties:

- $\frac{k}{3\Delta} \leq |S_i| \leq k$ for all i where $k = c\Delta$ and c is a constant.
- Every edge of $\hat{\tau}(x)_V$ is present inside exactly one S_i.

In other words, S_i's partition the edge set of $\hat{\tau}(x)_V$. Note that the S_i's are sets of vertices that are not disjoint.

Verifying all edges inside a S_i takes $O(k^2)$ queries by Lemma 12. The total number of starshaped sets is h which is at most $3\Delta n/k$. Hence, the total number of queries to verify the edges within every S_i is $O(n\Delta^2)$. Note that we have not verified the shortest path graph as there may be edges from layer i to layer $i+1$ that are not contained in any starshaped set. However, since every edge of $\hat{\tau}(x)_V$ is inside some S_i, we have shown that every edge of $\hat{\tau}(x)_V$ is present in G.

Using Lemma 14, we get that G has the same layer structure as $\hat{G}$ by making at most n queries. Now, use Lemma 13 to verify all edges in the same layer using $O(1)$ query per edge. Finally, to verify edges $e = (y, z)$ from layer i to layer $i+1$, that are not contained in $\hat{\tau}(x)_V$, note that $\text{bet}(x, y, z)$ is true if and only if e is present in G. This takes 1 query per edge. The total number of such edges is at most $n\Delta$. Hence, we get the desired bound. ■

4.2. Non-edge verification

The algorithm for non-edge verification proceeds as follows. We start with V and split it into cells $U_1, U_2, \ldots, U_k$ such that every edge in $\hat{G}$ is completely contained in some U_i. These U_i's are called extended Voronoi cells and will be defined soon. Using Lemma 21, we can verify (with few queries) that every edge in G is completely contained in some U_i. Then, we can recursively apply the splitting technique to each U_i.

Given a cell U, the algorithm for splitting U into extended Voronoi cells first selects a set of centres $\{a_1, a_2, \ldots, a_k\} = A \subseteq V$ using Algorithm 1. Then, it builds Voronoi cells around these centres.

Definition 16 *Given $A \subseteq V$ and $w \in V$, the Voronoi cell of w with respect to A is defined as*

$$\hat{C}_A(w) = \{v \in V : \hat{\delta}(w, v) < \hat{\delta}(A, v)\}$$

We expand the Voronoi cells slightly so that every edge of $\hat{G}$ contained in U is completely contained in one of the extended Voronoi cells produced by splitting U. Note that the extended Voronoi cells are not disjoint.

Definition 17 *Let $A \subseteq V$ be the set of centres and $U \subseteq V$. Define for each $a \in A$, its extended Voronoi cell $\hat{D}_a \subseteq U$ as*

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

The superscript U is dropped when clear from context. The following lemma as stated in Kannan et al. (2015) guarantees that the splitting done using the centres algorithm does not return too many cells and the size of each cell goes down significantly compared to the size of U.

Lemma 18 *Given a graph $\hat{G} = (V, \hat{E})$, a subset of vertices $U \subseteq V$, and an integer $s \in [1, n]$, Algorithm 1 computes a subset of vertices $A \subseteq V$ such that the following conditions hold:*

- *The expected size of the set A is at most $2s \log n$*
- *For every vertex $w \in V$, we have $|\hat{C}_A(w) \cap U| \leq 4|U|/s$*

Remark 19 *Lemma 18 does not hold for arbitrary graphs. The bounded degree condition is necessary for its proof to go through. One obvious example where the bounded degree condition is not satisfied and the conclusion of the lemma is not true is the star graph.*

Also note that Algorithm 1 is randomized. Thorup and Zwick (2001) showed that it is possible to derandomize it and the running time is still polynomial.

Algorithm 1: Finding Centres for a Subset (Kannan et al. (2015))

Function *Subset-Centres*$(\hat{G}, U, s)$
  $A \leftarrow \emptyset$
  while *there exists* $w \in V$ *such that* $|\hat{C}_A(w) \cap U| > 4|U|/s$ **do**
    $W \leftarrow \{w \in V : |\hat{C}_A(w) \cap U| > 4|U|/s\}$
    Add each element of W to A with probability $\min(s/|W|, 1)$
  end
  return A

Now, we can recursively apply this technique for each extended Voronoi cell $U \in \{\hat{D}_{a_1}, \hat{D}_{a_2}, \ldots, \hat{D}_{a_s}\}$. When applying the centres algorithm recursively, the following definitions are useful. Each node of the recursion tree is a subset of V. The root is V and it is in level 1 of the recursion. We use N_k to denote the set of nodes in level k. Hence $N_1 = \{V\}$. Let $U \in N_i$ be a node in level i. If the centres algorithm returns $A(U) = \{a_1, a_2, \ldots a_k\}$ when run on U, then $C(U) = \{\hat{D}^U_{a_1}, \hat{D}^U_{a_2}, \ldots, \hat{D}^U_{a_k}\}$ are the children of U.

Definition 20

$$S^U_{\{a\}} = \{(a', p, u) : a' \in \hat{N}_2(a), u \in U, p \in \hat{p}_{a'}(u)\}$$

$$S^U_A = \bigcup_{a \in A} S^U_{\{a\}}$$

Lemma 21 (Recursion step) *Assume that $\hat{E} \subseteq E$. Let $U \in N_k$ and A be the centres returned by the algorithm on U. If $\mathrm{bet}(a, u, v) = \hat{\mathrm{bet}}(a, u, v)$ for all $(a, u, v) \in S^U_A$, then every edge of G contained in U is contained in some $\hat{D}^U_{a_i} \in C(U)$.*

Proof Suppose there exists an edge $e = (v_1, v_2)$ in G that is not completely contained in any of the $C(U)$. Let $v_1 \in \hat{D}_{a_1}$ and $v_2 \in \hat{D}_{a_2}$. By assumption, $v_1 \notin \hat{D}_{a_2}$ and $v_2 \notin \hat{D}_{a_1}$.

If $v_1 \in \hat{N}_2(a_1)$, then $\mathrm{bet}(v_1, p, v_2) \neq \hat{\mathrm{bet}}(v_1, p, v_2)$ for $p \in \hat{p}_{v_1}(v_2)$. Hence, $v_1 \notin \hat{N}_2(a_1)$. By the same argument, $v_2 \notin \hat{N}_2(a_2)$. For the rest of the proof, we assume $v_1 \notin \hat{N}_2(a_1)$ and $v_2 \notin \hat{N}_2(a_2)$. The definitions below are also represented in Figure 3.

1. $\hat{\delta}(a_1, v_1) = m_1$
2. $\hat{\delta}(a_1, v_2) = m_2$
3. $\hat{\delta}(a_2, v_1) = l_1$
4. $\hat{\delta}(a_2, v_2) = l_2$

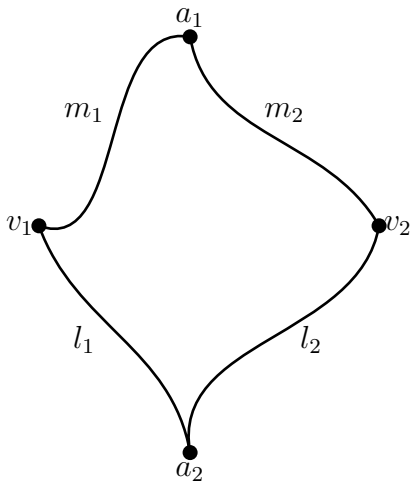

Figure 3: Pictorial representation of the definitions

Let b be a vertex at distance 2 from a_2 in the shortest path from a_2 to v_1. Then, $\hat{\delta}(b, v_1) \geq \hat{\delta}(a_1, v_1)$ because $v_1 \in \hat{D}_{a_1}$ and $v_1 \notin \hat{D}_{a_2}$. Hence, we get $l_1 - 2 \geq m_1$. Similarly, $m_2 - 2 \geq l_2$.

We claim that at least one of the following statements is true:

1. $v_1 \in \hat{L}(a_1)_k$ and $v_2 \in \hat{L}(a_1)_{k+s}$ for $s > 1$.
2. $v_2 \in \hat{L}(a_2)_k$ and $v_1 \in \hat{L}(a_2)_{k+s}$ for $s > 1$.

Suppose the first statement is false. Then, $m_1 - 1 \leq m_2 \leq m_1 + 1$. Using $l_1 \geq m_1 + 2$ and $m_2 \geq l_2 + 2$, we get that $l_2 + 3 \leq l_1$. Hence, the second statement is true.

If statement i is true, then with a_i as centre, v_i and v_j belong to layers that are far apart in $\hat{G}$ (where $i \in \{1, 2\}$, $j \neq i$ and $j \in \{1, 2\}$). But they are close in G because there is an edge between v_i and v_j. We exploit this to get a contradiction. Let $v_i \in \hat{L}(a_i)^V_k$ and $v_j \in \hat{L}(a_i)^V_{k+s}$ for $s > 1$. Because there is an edge between v_i and v_j, $v_j \in L(a_i)^V_t$ for some $t \leq k + 1$. Hence, with a_i as center, the shortest path to v_j has changed in G. This changes the output of some betweenness query in G. In particular, if P denotes the set of vertices along a shortest path from a_i to v_j in $\hat{G}$, $\text{bet}(a_i, p_v, v) \neq \hat{\text{bet}}(a_i, p_v, v)$ for some $v \in P$ and $p_v \in \hat{p}_{a_i}(v)$. This contradicts $\text{bet}(a, u, v) = \hat{\text{bet}}(a, u, v)$ for all $(a, u, v) \in S^U_A$ concluding the proof.

■

Finally, we have all the machinery required to prove the main result. We need Lemma 15 for edge verification and Lemma 21 for recursion.

Proof [Proof of Theorem 10] First we do the edge verification using Lemma 15. For non-edge verification, the proof follows closely the recursive verification analysis done in Kannan et al. (2015).

Algorithm 2 is the recursive algorithm for non-edge verification. It starts with $U = V$ and queries every $(u, v, w) \in S_A^U$ where A is the set of centres returned by the centres algorithm. Then, it repeats the process for each $\hat{D}_a$. The tree interpretation of the recursion process discussed earlier will be useful for the rest of the proof. In Algorithm 2, $\textsc{Query}\left(S_{\{a\}}^U\right)$ means querying every $(u, v, w) \in S_{\{a\}}^U$ and $\textsc{Query}(X, Y, Z)$ means querying every 3-tuple of the form (x, y, z) such that $x \in X$, $y \in Y$ and $z \in Z$. If the queries in $\textsc{Query}\left(S_{\{a\}}^U\right)$ returns the expected result for all $a \in A(U)$, by Lemma 21, we conclude that every edge of G contained in U is contained in some $\hat{D}_{a_i}^U \in C(U)$. Now, we need to fix the constants in Algorithm 2 and compute its query complexity.

Algorithm 2: Recursive Verification

Procedure *Verify-Subgraph*$(\hat{G}, U)$
- **if** $|U| > n_0$ **then**
  - $A \leftarrow \textsc{Subset-Centres}(\hat{G}, U, s)$
  - **for** $a \in A$ **do**
    - $\textsc{Query}\left(S_{\{a\}}^U\right)$
    - $\textsc{Verify-Subgraph}(\hat{G}, \hat{D}_a)$
  - **end**
- **else**
  - $\textsc{Query}(U, U, U)$
- **end**

Define

$$k_0 = \left\lfloor \sqrt{\frac{\log n}{\log(\log n \cdot 128(\Delta^2+1)^3)}} \right\rfloor$$

Define $s = n^{1/k_0}$, $n_0 = (4(\Delta^2+1))^{k_0}$. n_0 is going to be a threshold on |U|. If the size of $|U|$ falls below this, we stop the recursion. If $|U| > n_0$, the number of centres returned by the algorithm is $|A(U)| \leq 2s \log n$ and for any $W \in C(U)$, $|W| \leq (\Delta^2+1) \cdot \max(4|U|/s, 1)$. By induction, we get for any $U \in N_k$, $|U| \leq n(4(\Delta^2+1)/s)^{k-1}$ for all $1 \leq k \leq k_0 + 1$.

Consider the queries made by the leaf nodes of the tree (i.e. $|U| \leq n_0$). The depth of the tree is at most $k_0 + 1$. Hence, there are at most $(2s \log n)^{k_0}$ leaves. Each leaf node makes at most $|U|^3 \leq (4(\Delta^2+1))^{3k_0}$ queries. Hence, the total number of queries in this step is at most $n(\log n \cdot 128(\Delta^2+1)^3)^{k_0} \leq n^{1+1/k_0}$.

Now consider the recursive calls made by non-leaf nodes (i.e. $|U| > n_0$). Here, $k \in [1, k_0]$. For a fixed k, there are at most $|N_k| = (2s \log n)^{k-1}$ calls at level k. Each such call takes at most $|A(U)||S_{\{a\}}^U| = (\Delta^2+1)\Delta|A(U)||U|$ queries where $U \in N_k$. Hence, the total number of queries for a fixed k is at most $\Delta \cdot n^{1+1/k_0}(\log n \cdot 8(\Delta^2+1))^k$. Summing over $k \in [1, k_0]$, we find the total number of queries made by non-leaf nodes is at most $2\Delta \cdot n^{1+1/k_0}(\log n \cdot 8(\Delta^2+1))^{k_0} \leq 2\Delta \cdot n^{1+2/k_0}$. This completes the proof. ■

5. Open problems

For the graph learning problem with a distance oracle, there is an upper bound of $\tilde{O}(n^{3/2}\Delta^4)$ for graphs with constant Δ and the lower bound is $\tilde{\Omega}(n\Delta)$. The main open problem is to find an algorithm with an upper bound of $n^{1+o(1)} \cdot f(\Delta)$ where $f(\Delta)$ is some function of Δ. For the graph verification problem with a distance oracle, there is an upper bound of $n^{1+O\left(\sqrt{(\log\log n+\log\Delta)/\log n}\right)}$ and a lower bound of $\Omega(n)$. It would be interesting to find an algorithm where the dependence on Δ is of the form $\tilde{O}(nf(\Delta))$.

For the betweenness oracle, the state of the art for degree-bounded graphs is almost the same as that of the distance oracle stated above. The main difference comes from the dependence on Δ. Because betweenness queries are weaker, the dependence on Δ becomes worse. But the fact that the weaker query can get us almost the same upper bound gives motivation for improving the upper bounds for distance oracle. This paper shows how the partitioning technique developed for the graph verification problem in Kannan et al. (2015) is robust enough to be extended to a betweenness oracle. It would be interesting to see if this technique can be extended to other oracles.

Acknowledgements

I thank Lev Reyzin for directing me towards this problem and for many helpful discussions.

References

Mikkel Abrahamsen, Greg Bodwin, Eva Rotenberg, and Morten Stöckel. Graph reconstruction with a betweenness oracle. In *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, pages 5:1–5:14, 2016. doi: 10.4230/LIPIcs.STACS.2016.5. URL http://dx.doi.org/10.4230/LIPIcs.STACS.2016.5.

Noga Alon and Vera Asodi. Learning a hidden subgraph. *SIAM Journal on Discrete Mathematics*, 18(4):697–712, 2005.

Noga Alon, Richard Beigel, Simon Kasif, Steven Rudich, and Benny Sudakov. Learning a hidden matching. *SIAM Journal on Computing*, 33(2):487–501, 2004.

Dana Angluin and Jiang Chen. Learning a hidden graph using $o(\log n)$ queries per edge. *Journal of Computer and System Sciences*, 74(4):546–556, 2008.

Zuzana Beerliova, Felix Eberhard, Thomas Erlebach, Alexander Hall, Michael Hoffmann, Matúš Mihal'ak, and L Shankar Ram. Network discovery and verification. *IEEE Journal on selected areas in communications*, 24(12):2168–2181, 2006.

Luca Dall'Asta, Ignacio Alvarez-Hamelin, Alain Barrat, Alexei Vázquez, and Alessandro Vespignani. Exploring networks with traceroute-like probes: Theory and simulations. *Theoretical Computer Science*, 355(1):6–24, 2006.

Thomas Erlebach, Alexander Hall, Michael Hoffmann, and Matúš Mihal'ák. Network discovery and verification with distance queries. In *Italian Conference on Algorithms and Complexity*, pages 69–80. Springer, 2006.

Jotun J Hein. An optimal algorithm to reconstruct trees from additive distance data. *Bulletin of mathematical biology*, 51(5):597–603, 1989.

Sampath Kannan, Claire Mathieu, and Hang Zhou. Near-linear query complexity for graph inference. pages 773–784, 2015.

Valerie King, Li Zhang, and Yunhong Zhou. On the complexity of distance-based evolutionary tree reconstruction. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 444–453. Society for Industrial and Applied Mathematics, 2003.

Claire Mathieu and Hang Zhou. *Graph Reconstruction via Distance Oracles*, pages 733–744. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. ISBN 978-3-642-39206-1. doi: 10.1007/978-3-642-39206-1_62. URL http://dx.doi.org/10.1007/978-3-642-39206-1_62.

Brendan D. McKay and Nicholas C. Wormald. Asymptotic enumeration by degree sequence of graphs of high degree. *European Journal of Combinatorics*, 11(6):565 – 580, 1990. ISSN 0195-6698. doi: http://dx.doi.org/10.1016/S0195-6698(13)80042-X. URL http://www.sciencedirect.com/science/article/pii/S019566981380042X.

Lev Reyzin and Nikhil Srivastava. Learning and verifying graphs using queries with a focus on edge counting. In *International Conference on Algorithmic Learning Theory*, pages 285–297. Springer, 2007a.

Lev Reyzin and Nikhil Srivastava. On the longest path algorithm for reconstructing trees from distance matrices. *Information processing letters*, 101(3):98–100, 2007b.

Mikkel Thorup and Uri Zwick. Compact routing schemes. In *Proceedings of the thirteenth annual ACM symposium on Parallel algorithms and architectures*, pages 1–10. ACM, 2001.

Michael S Waterman, Temple F Smith, M Singh, and WA Beyer. Additive evolutionary trees. *Journal of theoretical Biology*, 64(2):199–213, 1977.