Network Discovery and Verification with Distance Queries*

Thomas Erlebach[†] Alexander Hall[‡] Michael Hoffmann[†] Matúš Mihal'ák[†]

March 31, 2006

Abstract

The network discovery (verification) problem asks for a minimum subset $Q \subseteq V$ of queries in an undirected graph $G = (V, E)$ such that these queries discover all edges and non-edges of the graph. This is motivated by the common approach of combining local measurements in order to obtain maps of the Internet or other dynamically growing networks. In the distance query model, a query at node q returns the distances from q to all other nodes in the graph. We describe how the existence of an individual edge or non-edge in G can be deduced by potentially combining the results of several queries. This leads to a characterization of when a set of queries Q "discovers" the graph G. In the on-line network discovery problem, the graph is initially unknown, and the algorithm has to select queries one by one based only on the results of the previous ones. We study the problem using competitive analysis and give a randomized on-line algorithm with competitive ratio $O(\sqrt{n \log n})$ for graphs on n nodes. We also show lower bounds $\Omega(\sqrt{n})$ and $\Omega(\log n)$ on competitive ratios of deterministic on-line algorithms and randomized on-line algorithms, respectively. In the off-line network verification problem, the graph is known in the beginning and the problem asks for a minimum number of queries to verify all edges and non-edges. We show that the problem is $\mathcal{NP}$-hard and present an $O(\log n)$-approximation algorithm.

1 Introduction

The recent growing interest in decentralized networks (such as the Internet or sensor networks) introduced many new algorithmic aspects different from those in static, centrally planned networks. A key difference is that there is no central authority which holds a map of such a network. Obtaining an accurate map, usually modeled as a graph, is generally not easy due to the network's dynamic growth process. That is, before the structure of a network can be analyzed and interpreted, one needs to measure the network in order to discover its nodes and links.

A common approach to obtain a map of a network, or at least a good approximation, is to make some local measurements—which could be seen as local views of the network from selected nodes (also referred to as vantage points)—and to combine these in an appropriate manner. There is an extensive body of related work studying various aspects of this approach, confer e.g. [16, 10, 18, 13, 14, 11, 3, 20, 9, 1, 7, 8].

As measuring at a node is usually very costly (in terms of time, energy consumption and money), the question of minimizing the number of such measurements arises naturally. Nevertheless, it was proposed only recently [4] to study this problem from a combinatorial optimization point of view.

Beerliova et al. [4] introduce the network discovery and verification problems, which ask to find a map of a network with a small number of queries (measurements). In the on-line network discovery problem only the nodes V of a graph G are known in the beginning. An algorithm can make queries at nodes of the graph

[*] Work partially supported by European Commission - Fet Open project DELIS IST-001907 Dynamically Evolving Large Scale Information Systems, for which funding in Switzerland is provided by SBF grant 03.0378-1.

[†] Department of Computer Science, University of Leicester, University Road, Leicester LE1 7RH, United Kingdom. {te17, mh55,mm215}@mcs.le.ac.uk.

[‡] Institute for Theoretical Computer Science, ETH Zurich, CH-8092 Zurich, Switzerland. alex.hall@inf.ethz.ch.

and each query returns a local view of the graph. The task of the algorithm is to choose a minimum subset $Q \subseteq V$ of queries, such that the whole graph is discovered, i.e., all edges and non-edges. The network verification problem is the off-line version of the problem. The whole graph is known to the algorithm and the task is to compute a minimum set of queries Q which verify all edges and non-edges. One possible motivation for the off-line version is to be able to check with as few measurements as possible that a given map is still correct.

Note that in order to discover a graph it might seem sufficient to discover only the edges of the graph. However, especially in view of the on-line setting it is also necessary to have a proof (i.e., discover) for unconnected node-pairs that there actually is no edge between them. An on-line algorithm can only know that it has finished discovering the graph when both edges and non-edges have been discovered. Taking both into account also makes it possible to in some sense quantify how much knowledge about the network is revealed by a given set of queries. Therefore, this small observation could also be helpful when e.g. investigating the quality of previously published maps of the Internet.

In [4], a very strong query model was used: a query at a node v reveals all edges and non-edges whose endpoints have a different distance from v. That query model was motivated by the consideration that in certain scenarios, it is possible to identify all edges on shortest paths between the query node and all other nodes. In this paper, we study the network discovery problem and network verification problem in the model where a query $q \in V$ gives all distances from q to any other node of the investigated graph G. We refer to the on-line problem as DIST–ALL–DISCOVERY and to the off-line problem as DIST–ALL–VERIFICATION. This *distance query model* is much weaker than the model used in [4], in the sense that typically a query reveals much less information about the network.

There are several reasons that motivate us to study the distance query model. First, in many networks it is realistically possible to obtain the distances between a node and all other nodes, while it is difficult or impossible to obtain information about edges or non-edges that are far away from the query node. For example, so-called distance-vector routing protocols work in such a way that each node informs its neighbors about upper bounds on the distances to all other nodes until these values converge; in the end, the routing table at a node contains the distances to all other nodes, and a query in our model would correspond to reading out the routing table. Another scenario is the discovery of the topology of peer-to-peer networks such as Gnutella [6]. There, with the Ping/Pong protocol it is possible to use a Ping command to ask all nodes within distance k (the TTL parameter of the Ping) to respond to the sender [2]. Repeated Pings could be used to determine the distances to all other nodes. Real peer-to-peer networks, however, are often so large that it becomes prohibitive to send Pings for larger values of k, and there are also many other aspects that make the actual discovery of the topology of a Gnutella network very difficult [2]. Nevertheless, we believe that our model is a good starting point for studying fundamental issues in the discovery of peer-to-peer networks or other networks that support Ping/Pong-like protocols.

Related Work. There are several ongoing large scale efforts to collect data representing local views of the Internet; here we will only mention two. The most prominent one is probably the RouteViews project [18] by the University of Oregon. It collects data from a large number of so-called border gateway protocol routers. Essentially for each router—which can be seen as a node in the Internet graph—the list of paths it knows (to all other nodes in the network) is retrieved. More recently, and due to good publicity very successfully, the DIMES project [10] has started collecting data with the help of a volunteer community, similar in spirit to SETI@Home [19]. Users can download a client which collects paths in the Internet by executing successive traceroute commands. A central server can direct each client individually by specifying which routes to investigate.

Data obtained by these or similar projects has been used in heuristics to obtain maps of the Internet, basically by simply overlaying possible paths found by the respective project, see e.g. [14, 18, 10, 16]. Another line of research aims at inferring from such local views the types of economic relationships between nodes in the Internet graph, cf. [11, 20, 9].

Beerliova et al. [4] propose the general problem of network discovery (verification) and study it for the "layered graph" query model: a query $q \in V$ returns all edges and non-edges between nodes of different distances from q. They present an $o(\log n)$ inapproximability result for the off-line version and give a randomized on-line algorithm with competitive ratio $O(\sqrt{n \log n})$. The on-line algorithm presented in this paper is based on a similar approach, but requires new ideas.

Network verification in the layered graph query model is closely related to the problem of finding the metric dimension of a graph, which can be defined as the cardinality of a minimum subset of nodes $Q \subset V$ such that every node in the graph has a unique vector of distances to Q (see [12]). Such a minimum subset Q is also called a basis of the graph and it is easy to see that this is the same as a minimum query set in the layered graph query model. Khuller et al. [17] investigate the problem of finding such a basis (and thus a minimum query set). They present an $O(\log n)$-approximation algorithm and investigate special graph classes. Cáceres et al. [5] study the metric dimension in Cartesian products of graphs and give many helpful references, pointing out interesting connections to other closely related problems.

Our Results and Outline. In Section 2 we start with some basic definitions concerning network discovery and verification in the distance query model. We then give a characterization of the queries that discover an individual non-edge and the sets of queries that together discover an individual edge.[1] These characterizations will be very helpful in the remainder of the paper.

In Section 3 we show lower bounds on the number of queries needed to discover or verify a graph, based on the independence number $\alpha(G)$, clique number $\omega(G)$ and size of the edge-set of the graph, $|E(G)|$.

For DIST–ALL–VERIFICATION we present polynomial time algorithms for basic graph classes—chains, cliques, trees, cycles, and hypercubes. For general graphs, the problem turns out to be $\mathcal{NP}$-hard and an $O(\log n)$-approximation algorithm is presented; see Section 4.

For DIST–ALL–DISCOVERY we show in Section 5 that no deterministic on-line algorithm can be better than $O(\sqrt{n})$-competitive and no randomized on-line algorithm can be better than $O(\log n)$-competitive. Finally, we present our main result, a randomized on-line algorithm with competitive ratio $O(\sqrt{n \log n})$.

2 Definitions and Preliminaries

Throughout this paper we assume graphs to be undirected and connected. For a given graph $G = (V, E)$, we denote the number of nodes by $n = |V|$ and the number of edges by $m = |E|$. For two distinct nodes $u, v \in V$, we say that $\{u, v\}$ is an *edge* if $\{u, v\} \in E$ and a *non-edge* if $\{u, v\} \notin E$. The set of non-edges is denoted by $\overline{E}$. By $\overline{G}$ we denote the complement of G, i.e., $\overline{G} = (V, \overline{E})$.

A *query* is specified by a node $v \in V$ and is called a query *at* v or simply the query v. In the *distance query model* the answer of a query at v consists of the distances from v to every node of G. We refer to sets of nodes with the same distance from v as *layers*. We use L_i or simply *layer* i to refer to the layer of nodes at distance i from the query node. By $d_G(u, v)$ we denote the distance from u to v in G. We may omit the subscript G if it is clear from context to which graph the distance refers. Let $\mathbf{D}_G(Q)$, for $Q \subseteq V$, be a collection of distance vectors, one vector $d_G(Q, v)$ for each node $v \in V$. The vector $d_G(Q, v)$ has dimension $|Q|$, and each component gives the distance $d_G(q, v)$ of one of the (query) nodes $q \in Q$ to v; the i-th component corresponds to the i-th query node. Thus, we write $\mathbf{D}_G(Q) \neq \mathbf{D}_{G'}(Q)$, for $G' = (V, E')$, if there exists at least one query $q \in Q$ and a node $v \in V$ such that $d_G(q, v) \neq d_{G'}(q, v)$. Conversely, $\mathbf{D}_G(Q) = \mathbf{D}_{G'}(Q)$, if $d_G(q, v) = d_{G'}(q, v)$ holds for all queries $q \in Q$ and all nodes $v \in V$.

As opposed to the layered graph query model studied in [4], in the distance query model a query at node v does not explicitly return edges or non-edges. We shall show, however, how the information about the distances of nodes to (possibly a combination of several) queries can be utilized for discovering individual edges or non-edges of the graph. But first we give a formal notion of what we mean by "discovering" a graph in this model. Note that we use the two terms discover and verify to distinguish between the on-line and the off-line setting, they are otherwise equivalent (and we sometimes use the word "discover" also in

[1] Note: At first sight it may seem that the only way to discover an edge in the distance query model is to query one of its incident nodes. It turns out that the query model allows more intricate deductions and that also edges at a large distance from the query nodes can be discovered. This will be explained in detail in Section 2.

the off-line setting). I.e. the following definitions hold for both terms but for simplicity are stated only for the network discovery setting.

Discovering a Graph. A query set $Q \subseteq V$ for the graph $G = (V, E)$ *discovers the edge* $e \in E$ (*discovers the non-edge* $\overline{e} \in \overline{E}$), if for all graphs $G' = (V, E')$ with $\mathbf{D}_G(Q) = \mathbf{D}_{G'}(Q)$ it must hold that $e \in E'$ ($\overline{e} \in \overline{E'}$). $Q \subseteq V$ *discovers the graph* G, if it discovers all edges and non-edges of G.

Equivalent Definition. Q discovers G implies that any graph G' with $\mathbf{D}_G(Q) = \mathbf{D}_{G'}(Q)$ must have the same edges and non-edges as G, in other words: $G' = G$. Conversely, if a query set Q for G yields $\mathbf{D}_G(Q) = \mathbf{D}_{G'}(Q)$ only for $G' = G$ and for no other graph, then Q discovers G (since it clearly can discover each edge and non-edge individually).

This gives an equivalent definition: A query set $Q \subseteq V$ *discovers the graph* $G = (V, E)$, if for every graph $G' = (V, E') \neq G$ at least one of the resulting distances changes, i.e., $\mathbf{D}_G(Q) \neq \mathbf{D}_{G'}(Q)$. Intuitively, the queries Q which discover a graph G can distinguish it from any other graph G' (sufficient and necessary condition).

Characterizing the Queries Discovering a Non-Edge. If we look at a particular non-edge $e \in \overline{E}$, there exists a query $q \in Q$ that confirms this non-edge to be in G:

Observation 1 *For $G = (V, E)$ the queries $Q \subseteq V$ discover a non-edge $\{u, v\} \in \overline{E}$ if and only if there exists a query $q \in Q$ with $|d(q, u) - d(q, v)| \geq 2$.*

Proof. The implication "$\Leftarrow$" is easy to see: Clearly, if there is a query q such that $|d(q, u) - d(q, v)| \geq 2$, then $\{u, v\}$ is a non-edge. To see the second implication "$\Rightarrow$", assume that $\{u, v\}$ is a non-edge and that (for contradiction) every query node q gives $|d(q, u) - d(q, v)| \leq 1$. We show that if $\{u, v\}$ was an edge, the distances returned by Q would not change. Indeed, u and v are either in the same layer or in two consecutive layers of a query q. Therefore adding an edge $\{u, v\}$ to G cannot decrease a distance from q to any other node. □

For a query q and $\{u, v\} \in \overline{E}$ with $|d(q, u) - d(q, v)| \geq 2$, we say that *$q$ discovers the non-edge $\{u, v\}$*.

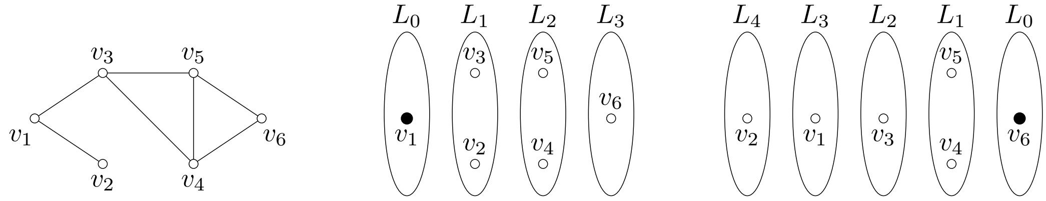

Figure 1: Edge $\{v_3, v_4\}$ of a graph (left) is discovered by the combination of queries at nodes v_1 and v_6; the distances to the query node v_1 (middle) and v_6 (right) are depicted via layers of the graph

Characterizing the Sets of Queries Discovering an Edge. An edge may be discovered by a combination of several queries (this is a major difference to the layered graph query model of [4], where the set of edges and non-edges discovered by a set of queries is simply the union of the edges and non-edges discovered by the individual queries). If a node w is in layer $i + 1$ of a query q, this shows that w must be adjacent to at least one node from layer i. If layer i has more than one node, then in general it is not clear which node from layer i is adjacent to w. Figure 1 shows an example of how a combination of two queries can discover an edge even if each of the two queries alone does not discover the edge: The edge $\{v_3, v_4\}$ is neither discovered by a query at v_1 nor by a query at v_6 alone. The query at v_1 reveals that v_4 is connected to v_2 or to v_3. The query at v_6 identifies $\{v_2, v_4\}$ as a non-edge. From these two facts one can deduce that v_4 must be connected to v_3, i.e., $\{v_3, v_4\}$ is an edge. This discussion is generalized by the following observation.

Observation 2 *For $G = (V, E)$ the queries $Q \subseteq V$ discover an edge $\{u, v\} \in E$ if and only if there is a query $q \in Q$ with the following two properties:*

(i) The nodes u and v are in consecutive layers of query q, say, u in the i-th layer L_i and v in the $(i+1)$-th layer L_{i+1}, and $L_i \setminus \{u\}$ does not contain any neighbor of v.

(ii) The queries Q discover all non-edges between v and the nodes in $L_i \setminus \{u\}$.

Proof. We again start with the easy direction "$\Leftarrow$": From the result of query q in (i) one can deduce that there must be an edge from some node in L_i to v. From (ii) it follows that $\{u, v\}$ is the only possibility for such an edge.

For the implication "$\Rightarrow$", we give a proof by contradiction. Assume that the query set Q discovers the edge $\{u, v\}$. Observe that if (i) does not hold, then all queries yield the same results if $\{u, v\}$ is removed from G. To see this, consider an arbitrary query $q' \in Q$. If u, v are originally at the same distance from q', they also will be at the same distance after removing $\{u, v\}$. If u, v are originally at different distances from q', say $u \in L_{i'}$ and $v \in L_{i'+1}$, we know that since (i) does not hold, v has another neighbor in $L_{i'} \setminus \{u\}$. Therefore, we know that v is in $L_{i'+1}$ even after removing the edge $\{u, v\}$. So in this case as well, $\mathbf{D}_G(Q)$ does not change if we remove $\{u, v\}$. This contradicts our assumption that Q discovers $\{u, v\}$.

Thus, we can assume that (i) holds. For each $q \in Q$ for which (i) holds, assume that (ii) does not hold. Let q be a query for which (i) holds. Assume that u is in layer L_i of that query and v is in layer L_{i+1}. As (ii) does not hold, there must be at least one non-edge $e_q = \{u', v\}$ for some $u' \in L_i$ that is not discovered by Q. We modify the graph G as follows: We remove the edge $\{u, v\}$, and we add the edges e_q for all $q \in Q$ for which (i) holds (these edges e_q are not necessarily distinct). It is easy to see that the resulting graph G' satisfies $\mathbf{D}_{G'}(Q) = \mathbf{D}_G(Q)$, proving that Q does not discover the edge $\{u, v\}$ in G, a contradiction. □

We say that a query for which (i) holds is a *partial witness* for the edge $\{u, v\}$. The word "partial" indicates that the query alone is not necessarily sufficient to discover the edge; additional queries may be necessary to discover the non-edges required by (ii).

We conclude that a set of queries discovers a graph G if and only if it discovers all non-edges and contains a partial witness for every edge.

3 Lower bounds

In this section we show lower bounds on the number of queries needed to discover G. We relate this number to the independence number α of the graph, to the clique number ω of the graph, and to the number of edges m.

Lemma 1 *For any graph G with independence number α and diameter $diam > 2$, at least $\log_{\lceil \frac{diam}{2} \rceil}(\alpha) - 1$ queries are needed to discover G. If $diam = 2$, we need at least $\alpha - 1$ queries.*

Proof. Let $A_0 \subseteq V$ be an independent set of size α. Any query q splits the nodes into at most $diam + 1$ layers. In layer 0 there is only q itself. We merge each pair of consecutive layers $2i - 1$ and $2i$, for $i \geq 1$, so that we obtain at most $\beta := \lceil \frac{diam}{2} \rceil$ new layers $\tilde{L}_i$ (the last new layer may consist of a single original layer). Query q does not discover any non-edge whose endpoints lie within the same new layer. At least $\alpha - 1$ nodes of the independent set A_0 are distributed among the β new layers (one node of A_0 may be the query node, which is not in the new layers). Thus, there must be a new layer $\tilde{L}_i$ with at least $(\alpha - 1)/\beta$ nodes from A_0. Let A_1 denote the set of these nodes. If $(\alpha - 1)/\beta > 1$, then we need at least one more query to discover the non-edges within A_1. After the second query, there is a new layer containing at least $(|A_1| - 1)/\beta \geq ((\alpha - 1)/\beta - 1)/\beta$ nodes from A_1, and the argument can be repeated. Let α_k, for $k \geq 1$, denote the size of the biggest subset of A_0 for which the queries $q_1, \ldots, q_k$ do not discover any non-edge. By the arguments above, we have $\alpha_k \geq a_k$, where $a_0 = \alpha$ and $a_k = \frac{a_{k-1}-1}{\beta}$ for $k \geq 1$. We get $a_k = \frac{\alpha}{\beta^k} - \frac{1}{\beta^k} - \frac{1}{\beta^{k-1}} - \ldots - \frac{1}{\beta}$, i.e. $a_k = \frac{1}{\beta^k}(\alpha - \frac{\beta^k - 1}{\beta - 1})$ if $\beta > 1$ and $a_k = \alpha - k$ if $\beta = 1$. If k queries discover G, we must have that $a_k \leq 1$. For $\beta = 1$ we get $k \geq \alpha - 1$. For $\beta > 1$ we get $\beta^{k+1} \geq 1 + \alpha(\beta - 1)$, i.e. $k \geq \log_\beta \alpha + \log_\beta (\beta - 1 + \frac{1}{\alpha}) - 1 \geq \log_\beta \alpha - 1$. □

Lemma 2 *For any graph G with clique number ω we need at least $\omega - 1$ queries to discover G.*

Proof. Consider a clique $K_\omega \subseteq G$ of size ω. Let q be the first query. The nodes of K_ω appear in at most two consecutive layers i and $i + 1$ of query q. Observe that q is a partial witness of an edge from K_ω if and only if there is exactly one node v from K_ω in layer i and the rest is in layer $i + 1$. Moreover, q is a partial witness only for edges incident on v. After query q, there is still a $K_{\omega-1}$ for which no query has been made that is a partial witness of any of its edges. Therefore, by induction (using the fact that one query is necessary for a K_2 as the base case), it follows that we need at least $\omega - 1$ queries to discover G. □

Lemma 3 *Any graph G with n nodes and m edges needs at least $m/(n - 1)$ queries to be discovered.*

Proof. Consider the layers of an arbitrary query $q \in V$. For each node v on layer i, q can be a partial witness for at most one edge $\{u, v\}$ with u in layer $i - 1$. Therefore, q can be a partial witness for at most $n - 1$ edges. Since a set of queries that discovers G must contain a partial witness for each of the m edges of G, the bound follows. □

This lower bound shows that graphs with a super-linear number of edges need a non-constant number of queries to be discovered.

4 Network Verification

4.1 Polynomially Solvable Cases

Lemma 4 *G needs 1 query to be discovered if and only if G is a chain. A clique K_n needs $n - 1$ queries to be discovered.*

Proof. If G is a chain, then clearly a vertex of degree 1 discovers the chain. On the other hand, if one query q discovers the whole graph G, observe that q cannot discover an edge or non-edge between two vertices at the same distance from q. Therefore, the vertices of G have unique distance from q and therefore G is a chain.

The second part of the statement follows from Lemma 2 and since with $n - 1$ queries each edge has at least one incident query and therefore will be discovered. □

The example of the cycle with 4 nodes C_4 shows that there is a graph that needs $n - 1$ queries to be discovered and is not a clique. (The same holds for graphs that are obtained from K_n by deleting one edge, for $n \geq 4$.) In general, for cycles the following lemma holds.

Lemma 5 *A cycle C_n, $n > 6$, needs 2 queries to be discovered.*

Proof. By Lemma 4 we have that 1 query does not discover a cycle. We show now that 2 queries are enough. We argue for n being odd, i.e., $n = 2k + 1$. Similar arguments can be given for even n.

Let $V = \{v_0, \ldots, v_{n-1}\}$ be ordered according to their appearance on the cycle. Let $q_1 = v_0$ and $q_2 = v_2$ be two queries at C_n. Query v_0 divides the vertices into layers according to the distance. In every layer $i \geq 1$ there are 2 vertices v_i and v_{n-i} (see Figure 2). Observe that q_1 is a partial witness for all edges except $\{v_k, v_{k+1}\}$, and q_2 is a partial witness for $\{v_k, v_{k+1}\}$ (cf. Figure 2).

Query q_1 discovers all non-edges between vertices from non-neighboring layers. We show that q_2 discovers all the remaining undiscovered non-edges of type $\{v_i, v_{n-i}\}$, $\{v_i, v_{n-i-1}\}$ and $\{v_{i+1}, v_{n-i}\}$, for $i = 1, 2, \ldots, k - 1$. Notice that $\{v_1, v_{n-1}\}$ and $\{v_1, v_{n-2}\}$ are the only unknown non-edges incident on v_1 after query q_1. Observe that if $n > 6$, query q_2 discovers these non-edges. Hence we consider an unknown non-edge $\{v_a, v_b\}$ where $a > 2$ and $a \leq k$ and $b \geq k+1$, $b \in \{n-a+1, n-a, n-a-1\}$. The distance d_a from v_2 to v_a can be used to bound the distance to v_b as follows: $d(v_2, v_b) \geq \min\{4 + (d_a - 1), d_a + 2\} \geq d_a + 2$

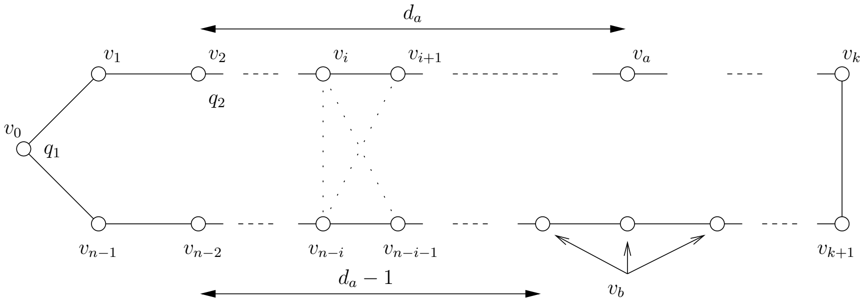

Figure 2: Cycle C_n can be discovered by queries at v_0 and v_2

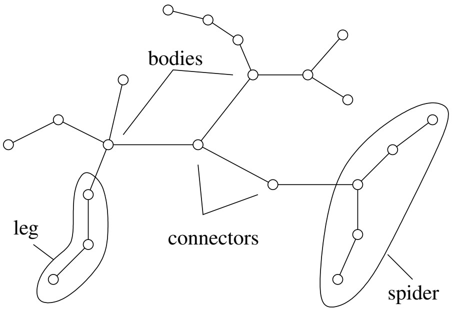

Figure 3: Legs, bodies, spiders and connectors in a tree

(by considering the lengths of the two paths from v_2 to v_b via v_0 or via v_k). Thus the distances $d(q_2, v_a)$ and $d(q_2, v_b)$ differ by at least two and therefore q_2 discovers the non-edge $\{v_a, v_b\}$.

We showed that q_1 and q_2 discover all non-edges and are partial witness for all edges. Therefore q_1 and q_2 discover the cycle C_n. □

Now we characterize the optimal query set for a tree T. For this, we define a *leg* to be a maximal path in the tree starting at a leaf and containing only vertices of degree at most 2, see Fig. 3. Therefore, if T is not a chain, there has to be a node u of degree greater than 2 adjacent to the last vertex of the leg. We call u a *body* and we say that the leg is *adjacent* to its body u. The body u with all its adjacent legs is called a *spider*. Nodes that are not part of a spider are called *connectors* (i.e., nodes that are not in a leg and have no adjacent leg).

Lemma 6 *Let $T = (V, E)$ be a tree that is not a chain. Denote by $B \subset V$ the set of bodies of the graph. Let l_b, for $b \in B$, be the number of legs adjacent to b. Let $T[B]$ be the induced subgraph of T on vertex set B. Let $VC(T[B])$ denote a minimum vertex cover of $T[B]$. Then the minimum number of queries to discover T is $\sum_{b \in B}(l_b - 1) + |VC(T[B])|$.*

Proof. We show first that we indeed need at least that many queries. For this observe that if there is no query in two legs adjacent to a body, then we cannot discover the non-edges formed by vertices of the two legs at the same distance from the body. Therefore there has to be at least one query in every leg but one of any body. Moreover, if there are two legs of two different bodies which are connected by an edge then there has to be at least one query in one of the legs. Otherwise we cannot discover the non-edge between vertices of the legs at the same distance from their bodies. Therefore for any two bodies connected by an edge at least one of them has a query in every leg. Observe that the bodies with all legs containing a query form a vertex cover of $T[B]$ and therefore a minimum vertex cover gives a lower bound on the number of spiders that have a query in every leg.

To prove that the claimed number of queries is sufficient, we construct a query set Q in the following way. We compute a minimum vertex cover of $T[B]$ (which can be done in polynomial time on trees). Let u be a body. We add the leaves of $l_u - 1$ of its legs to Q. If u is in the vertex cover, we add also the leaf of the last (the l_u-th) leg to Q.

We show now that Q discovers T. We start with non-edges. Let $\{v, w\}$ be a non-edge. We distinguish several cases. First, consider the case that both v and w are from legs. Consider the following subcases.

1. v and w are from the same leg. Clearly, the non-edge is discovered by any query.

2. v and w are from different legs, and there is a query q in the leg where v or w is. This query discovers the non-edge. (Note that there must be a query in the leg of v or w if they are in different legs of the same spider, or in legs of spiders whose centers are adjacent.)

3. v and w are from different spiders centered at u and u', which are not neighbors, and there is no query in the legs containing v and w. Let the path from u to u' be $u, x, \dots, y, u'$, where $x = y$ is possible. Let q be a query from a leg adjacent to a body b such that the path from b to u does not contain x, possibly $b = u$. Let d_v be the distance from u to v, d_w be the distance from u' to w and let $d \geq 2$ be the distance between u and u'. If q does not discover the non-edge $\{v, w\}$ then $|d(q, v) - d(q, w)| = |d_v - (d + d_w)| \leq 1$. Then a query q' from a leg adjacent to a body b' such that the path from b' to u' does not contain y satisfies $|d(q', v) - d(q', w)| = |(d_v + d) - d_w| \geq 3$ and thus q' discovers the non-edge.

Now, consider the case that at least one of the two nodes, say, the node v, is not from a leg. Then any query in a tree of the forest $T \setminus \{v\}$ that does not contain w verifies the non-edge. Observe that such a query always exists.

Therefore Q discovers all non-edges. We claim now that Q discovers all edges. For this observe that for a tree T any query is a partial witness for every edge. To see this imagine the tree rooted at the query node. Therefore, Q discovers T, which concludes the proof. □

Lemma 7 *A query set discovering a d-dimensional hypercube H_d is a vertex cover and any vertex cover verifies a d-dimensional hypercube H_d for $d \geq 4$. A minimum vertex cover discovers H_3. Therefore we need 2^{d-1} queries (size of a minimum vertex cover in H_d) for $d \geq 3$.*

Proof. First we show that a query set Q that discovers the given hypercube H_d is a vertex cover. Let $\{u, v\}$ be an arbitrary edge. Recall that we can label the nodes of the hypercube by d-dimensional vectors such that there is an edge between two vertices if and only if their labels have Hamming distance 1. Now, suppose that neither u nor v is in Q. We show that no other query is a partial witness for the edge $\{u, v\}$. Let q be a query. W.l.o.g. u is closer to q than v is. Therefore, w.l.o.g., $u = 000\dots0$ and $v = 100\dots0$ and $q = q_1q_2\dots q_d$, where $q_1 = 0$. There must exist an $i > 1$ such that $q_i = 1$. Then $w = \underbrace{10\dots0}_{i-1}10\dots0$ is a neighbor of v and is at the same distance from q as u, and therefore q cannot be a partial witness for the edge $\{u, v\}$. Thus, Q does not discover H_d.

Now we show that an arbitrary vertex cover discovers H_d when $d \geq 4$. Clearly, a vertex cover discovers all edges. We show that it discovers also all non-edges. Let $\{u, v\}$ be a non-edge in H_d. If u or v are in the vertex cover, the non-edge is discovered. We assume now that neither u nor v is in the vertex cover. W.l.o.g., $u = 00\dots0$ and $v = \underbrace{1\dots1}_{k}0\dots0$, $k \geq 2$. If $k = d$, i.e., v is antipodal to u then $10\dots0$ is a neighbor of u and therefore in the vertex cover. $10\dots0$ has a distance $d-1$ to v and distance 1 to u and since $(d-1)-1 = d-2 \geq 2$ the query at this node discovers the non-edge $\{u, v\}$. If $k < d$ then vertex $0\dots01$ (neighbor of u and therefore in the vertex cover) has distance $k+1$ to v and distance 1 to u and therefore the distance difference is $k \geq 2$ and therefore $\{u, v\}$ is discovered.

For $d = 3$, observe that $V \setminus \{000, 111\}$ is a vertex cover, but does not discover the non-edge $\{000, 111\}$. On the other hand this is not a minimum vertex cover for H_3 and therefore a minimum vertex cover for H_3 has to contain a vertex from every antipodal pair (and therefore discovers every such non-edge). To discover a non-edge $\{u, v\}$ of vertices at distance 2 from each other, i.e., w.l.o.g., $u = 000$ and $v = 110$, note that 111 has to be in the vertex cover if none of u, v is in it, and 111 discovers the non-edge $\{u, v\}$.

Finally, we note that the size of a minimum vertex cover for H_d, $d \geq 1$, is 2^{d-1}. To see this, observe that every vertex can cover at most d edges. The hypercube has $\frac{1}{2}2^d d$ edges and therefore a lower bound on

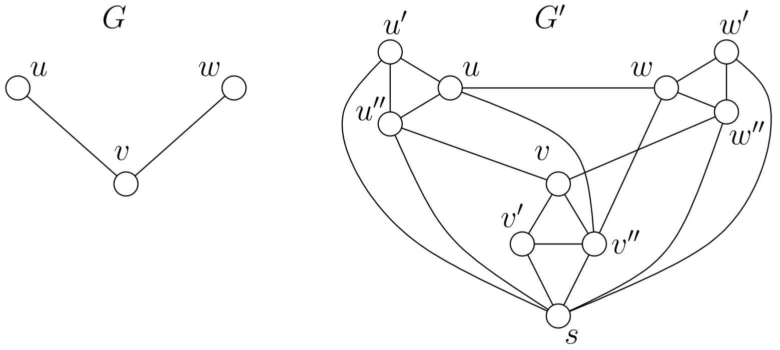

Figure 4: Instance of VERTEX–COVER (left), constructed instance of DIST–ALL–VERIFICATION (right)

the size of any vertex cover is 2^{d-1}. One can easily check that all vertices with even Hamming distance to the origin $00\dots0$ form a vertex cover and the number of such vertices is 2^{d-1}. □

4.2 $\mathcal{NP}$-Hardness

We consider the complexity of the DIST–ALL–VERIFICATION problem and show that it is $\mathcal{NP}$-hard. First we prove a useful lemma.

Lemma 8 *To discover a non-edge in a graph of diameter 2, one of its endpoints has to be a query.*

Proof. A non-edge $\{u, v\}$ is discovered by a query q, if the distances from q to u and v differ by at least 2. Since $diam = 2$, any node other than q is at distance 1 or 2 and therefore a query $q \notin \{u, v\}$ cannot discover the non-edge $\{u, v\}$. □

Theorem 1 *The problem* DIST–ALL–VERIFICATION *is $\mathcal{NP}$-hard.*

Proof. We present a polynomial-time reduction from the VERTEX–COVER problem to our problem. Let $G = (V, E)$ be a given graph for which a vertex cover is to be found. Let $n = |V|$. The basic idea is to create the complement $\overline{G}$ of G and add a new node s to the graph and connect it to all other nodes. The resulting graph G' has diameter 2. According to Lemma 8 a query set Q verifying G' contains an endpoint of every non-edge. Thus, discovering the non-edges in G' corresponds to finding a vertex cover in G. To verify also the edges of G' we may need more queries, however, and the number of these additional queries may vary. Therefore we modify the construction of G' in order to force an additional fixed (or more precisely: tightly bounded) number of queries which discover all edges. Given an instance $G = (V, E)$ of VERTEX–COVER, we start by constructing $\overline{G}$ and then extend it as follows: For each node $v \in V$, we add two new nodes v' and v'' and the edges $\{v, v'\}$, $\{v, v''\}$ and $\{v', v''\}$. In addition, we connect v'' to all nodes $w \in V$ that are not adjacent to v in $\overline{G}$. Finally, we add an extra node s and make it adjacent to all nodes of type v' and v''. Call the resulting graph G'. An example of the construction is shown in Figure 4. Denote the set of all nodes of type v' by V', and the set of all nodes of type v'' by V''. We observe that G' has diameter 2. Furthermore, both V' and V'' are independent sets in G'.

Let $C \subseteq V$ be an optimal vertex cover for G. We claim that $Q_C = \{s\} \cup V' \cup V'' \cup C$ is a query set that verifies G'. First, note that Q_C contains partial witnesses for all edges of G'; in particular, the query at v' is a partial witness for all edges in G' that connect v to other nodes from V. Furthermore, Q_C verifies all non-edges. For non-edges incident to a node from $\{s\} \cup V' \cup V''$, this is obvious. For non-edges between nodes in V this follows because C, being a vertex cover in G, contains at least one endpoint of every edge in G, and therefore at least one endpoint of every non-edge in G between nodes in V. Hence, there is a query set of size $2n + 1 + |C|$ that verifies G'.

Let Q be any query set that verifies G'. As V' and V'' are independent sets and G' has diameter 2, Q must contain at least $n - 1$ nodes from V' and at least $n - 1$ nodes from V'' by Lemma 8. Furthermore,

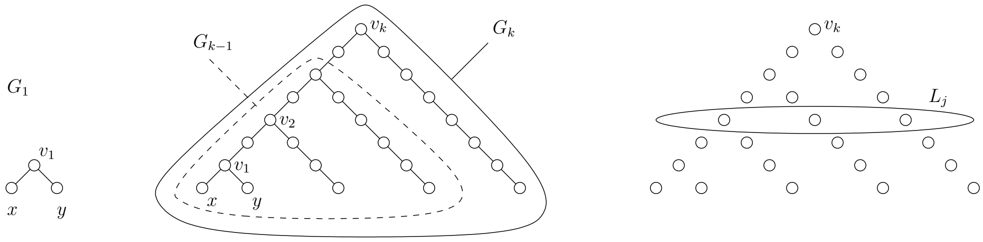

Figure 5: Graph used in the proof of the lower bound $\Omega(\sqrt{n})$ for on-line algorithms (left and middle); layers after query at vertex v_k (right)

the set $C' = Q \cap V$ must be a vertex cover of G, since it must contain an endpoint of every non-edge in G' between nodes in V. Hence, a query set Q that verifies G' yields a vertex cover of G of size at most $|Q| - 2n + 2$.

The discussion above shows that a polynomial-time algorithm computing an optimal query set for G' would give a vertex cover of size at most $(2n + 1 + |C|) - 2n + 2 = |C| + 3$. As VERTEX–COVER is $\mathcal{NP}$-hard to approximate within a factor of $7/6 - \varepsilon$ by [15], the problem DIST–ALL–VERIFICATION is $\mathcal{NP}$-hard. □

4.3 Approximation Algorithm

We present an $O(\log n)$-approximation algorithm for DIST–ALL–VERIFICATION that is based on the well-known greedy algorithm for the set cover problem. This technique was also used to derive the $O(\log n)$-approximation algorithm for metric dimension (equivalent to network verification in the layered graph query model) in [17].

Theorem 2 *There is an $O(\log n)$-approximation algorithm for* DIST–ALL–VERIFICATION.

Proof. We transform an instance $G = (V, E)$ of DIST–ALL–VERIFICATION into an instance of the set cover problem as follows. The edges and non-edges form the ground set $E \cup \overline{E}$ for the set cover problem. For each query $q \in V$, we introduce a subset $S_q = U_q \cup W_q$ of the ground set, formed by the set U_q of non-edges it verifies and the set W_q of edges for which it is a partial witness. By Observations 1 and 2, we can compute U_q and W_q. As a set of queries verifies G if and only if it discovers all non-edges and contains a partial witness for every edge, there is a direct correspondence between set covers and query sets that discover G. The standard greedy set cover approximation algorithm gives an approximation ratio of $O(\log |E \cup \overline{E}|) = O(\log \binom{n}{2}) = O(\log n)$. □

5 Network Discovery

5.1 Lower Bounds for Online Algorithms

We present a lower bound of $\Omega(\sqrt{n})$ on the competitive ratio of any deterministic on-line algorithm for the problem DIST–ALL–DISCOVERY. We also obtain an $\Omega(\log n)$ lower bound on the competitive ratio of randomized on-line algorithms.

Theorem 3 *There is no $o(\sqrt{n})$-competitive deterministic on-line algorithm for* DIST–ALL–DISCOVERY.

Proof. Consider the graph G_k from Figure 5. It is a tree built recursively from a smaller tree G_{k-1} as depicted in the figure. Alternatively, G_k can be described as follows. Start with a chain of length $2k - 1$ from x to v_k. For $1 \leq i \leq k$, the node on the chain at distance $2i - 1$ from x is labeled as v_i. To each such node v_i, $1 \leq i \leq k$, we attach another chain (which we call *arm*) of length $2i - 1$, starting at v_i. The number n_k of nodes of G_k satisfies $n_k = n_{k-1} + 1 + 2k$ for $k > 1$ and $n_1 = 3$. Hence, $n_k = k^2 + 2k$.

G_k is a non-trivial tree and, by Lemma 6, the optimum number of queries is 2. Now consider any deterministic algorithm A. As all vertices are indistinguishable to A, we may assume that the initial query q_0 made by A is at v_k. This sorts the vertices into layers according to their distance from v_k. There is no non-edge discovered within the layers. In particular, the non-edge $\{x, y\}$ in G_1 (see Figure 5) is not discovered. We now show that A needs at least k additional queries to discover $\{x, y\}$.

Observe that in the rightmost arm (attached to v_k) we have vertices from every layer. A picks a vertex from some layer j and, because all the vertices in this layer are indistinguishable for A, we may force A to pick the vertex from the rightmost arm. Such a query in the rightmost arm does not reveal any new information within G_{k-1}. The vertices within one layer of G_{k-1} remain indistinguishable for A. Thus, when A places its first query in G_{k-1}, we can force it to be at a node from G_{k-1}'s rightmost arm. Clearly, we can continue recursively in this manner and therefore we can force A to query in every arm before it discovers $\{x, y\}$. This yields that A needs at least $1 + k$ queries to discover G_k.

Since $n_k = k^2 + 2k$, we have that $k = \Theta(\sqrt{n_k})$. Together with the fact that the optimum needs 2 queries, we get the desired lower bound. □

Theorem 4 *There is no $o(\log n)$-competitive randomized on-line algorithm for* DIST–ALL–DISCOVERY.

Proof. To show a lower bound on the competitive ratio of any randomized algorithm A against an oblivious adversary, we use Yao's principle [21]: The (worst case) expected number of queries of a randomized algorithm A (against all inputs) is at least the expected number of queries of the best deterministic algorithm for any input distribution. Thus, to show the lower bound for any randomized algorithm, we create a set of instances and a probability distribution and show that any deterministic algorithm performs badly on this input distribution in expectation.

The input set $\mathcal{G}_k$ is as follows. The graph is always isomorphic to G_k (as shown in Figure 5). Let layer L_i be the set of all nodes at distance i from v_k. The input distribution is constructed by permuting the labels (identities) of the nodes in each layer L_i, $1 \leq i \leq 2k - 1$, using a permutation chosen uniformly at random. Let A be any deterministic algorithm. Let E_k denote the expected number of queries made by A on an instance G_k from $\mathcal{G}_k$, assuming that a query at v_k (or at some node outside G_k, if the G_k is part of a larger tree) may have been made already but no other query inside G_k has been made. When the algorithm makes the first query q inside G_k, there are the following cases. If the query q is made at some v_i, at the parent of v_i, or at a node in the arm attached to the parent of the parent of v_i, then after the query there is still a G_i such that no query has been made in it (except possibly at its root v_i). In that case, we say that a G_i *remains*. The expected number of queries required to discover G_i is then E_i. If the query q is made at one of the children of v_1, no G_i remains, and the algorithm may not require any additional queries. Letting p_i denote the probability that a G_i remains after the first query, we have $E_k \geq 1 + \sum_{i=1}^{k-1} p_i E_i$.

The algorithm makes the first query inside G_k at some layer j. Since the labels of the nodes of layer j have been permuted randomly, each of the nodes in layer j is equally likely to be the query node. For each layer, the probability that a G_i remains (possibly as part of a remaining $G_{i'}$ for $i' > i$) after a query in that layer is at least $\frac{1}{k+1}$ for each $i \in \{1, 2, \ldots, k-1\}$. (The minimum is achieved at the leaf layer.) Hence, we get

$$E_k \geq 1 + \sum_{i=1}^{k-1} \frac{1}{k+1} E_i$$

for $k \geq 2$ and $E_1 = 1$. This implies $E_k \geq H_{k+1} - \frac{1}{2} = \Theta(\log k)$, where $H_h = \sum_{i=1}^{h} \frac{1}{i}$ denotes the h-th harmonic number. Noting that the optimum is 2 and applying Yao's principle, we obtain the theorem (note that $k = \Theta(\sqrt{n_k})$, where n_k is the number of nodes in G_k, and thus $\log k = \Theta(\log n_k)$). □

5.2 Randomized Online Algorithm

In this section we present a randomized algorithm for DIST–ALL–DISCOVERY. The algorithm has competitive ratio $O(\sqrt{n \log n})$, which is very close to the lower bound $\Omega(\sqrt{n})$ for deterministic algorithms but leaves a gap to the lower bound $\Omega(\log n)$ for randomized algorithms.

The algorithm is a (non-straightforward) adaptation of the randomized algorithm for network discovery in the layered graph query model given in [4].

Theorem 5 *There is a randomized on-line algorithm with competitive ratio $O(\sqrt{n \log n})$ for* DIST–ALL–DISCOVERY.

Proof. The algorithm runs in two phases. In the first phase it makes $3\sqrt{n \ln n}$ queries at nodes chosen uniformly at random. In the second phase, as long as there is still an undiscovered pair $\{u, v\}$ (i.e., the queries executed so far have not discovered whether $\{u, v\}$ is an edge or non-edge), the algorithm executes the following. First, it queries both u and v. This discovers if $\{u, v\}$ is an edge or non-edge. In case it is a non-edge, the algorithm then knows from the queries at u and v the set S of all queries that discover $\{u, v\}$: S is the set of vertices w for which $|d(u, w) - d(v, w)| \geq 2$. The algorithm then queries the whole set S. In case $\{u, v\}$ is an edge, the algorithm distinguishes three cases. First, if the queries at u and v discover a non-edge, say, $\{u, w\}$, that hadn't been discovered before, the algorithm proceeds with the pair $\{u, w\}$ instead of $\{u, v\}$ and handles it as described above. Second, if the number of neighbors of u and the number of neighbors of v is at most $\frac{\sqrt{n}}{\sqrt{\ln n}}$, then the algorithm queries also all neighbors of u and v (notice that after querying u and v we know all their neighbors). With this information we know the set S of vertices that are partial witnesses for $\{u, v\}$: a vertex w is in S if and only if the two vertices are at distances i and $i + 1$ from w and all the other neighbors of the more distant vertex are at distances $i + 1$ or $i + 2$. Third, if the number of neighbors of u or the number of neighbors of v is more than $\frac{\sqrt{n}}{\sqrt{\ln n}}$, the algorithm does not do any further processing for this pair (i.e., this iteration of the second phase is completed) and proceeds with choosing another undiscovered pair $\{u', v'\}$ (if one exists).

The algorithm can be viewed as solving a HITTINGSET problem. For every non-edge $\{u, v\}$ let S_{uv} be the set of vertices that discover $\{u, v\}$. Similarly, for every edge $\{u, v\}$ let S_{uv} denote the set of all partial witnesses for $\{u, v\}$. The algorithm discovers the whole graph G if it hits all sets S_{uv}, for $\{u, v\} \in E \cup \overline{E}$. In the first phase, the algorithm aims to hit all the sets S_{uv} of size at least $\sqrt{n \ln n}$. Then, in the second phase, as long as there is an undiscovered pair $\{u, v\}$, the algorithm queries the whole set S_{uv}; if $\{u, v\}$ is an edge, it also queries all the neighbors of u and v in order to determine S_{uv}, except in the case where the degree of u or v is too large. In the case that the undiscovered pair $\{u, v\}$ is an edge for which a partial witness has already been queried before, the query at u or v must discover a new non-edge, and the algorithm uses that non-edge instead of $\{u, v\}$ to proceed.

We analyze the algorithm as follows. Let OPT be the optimal number of queries. Consider a pair $\{u, v\}$ for which the set S_{uv} has size at least $\sqrt{n \ln n}$. In each query of the first phase, the probability that S_{uv} is not hit is at most $1 - \frac{\sqrt{n \ln n}}{n} = 1 - \frac{\sqrt{\ln n}}{\sqrt{n}}$. Thus, the probability that S_{uv} is not hit throughout the first phase is at most

$$\left(1 - \frac{\sqrt{\ln n}}{\sqrt{n}}\right)^{3\sqrt{n \ln n}} = \left(\left(1 - \frac{\sqrt{\ln n}}{\sqrt{n}}\right)^{\frac{\sqrt{n}}{\sqrt{\ln n}}}\right)^{3 \ln n} \leq e^{-3 \ln n} = \frac{1}{n^3}.$$

There are at most $\binom{n}{2}$ sets S_{uv} of cardinality at least $\sqrt{n \ln n}$. The probability that at least one of them is not hit in the first phase is at most $\binom{n}{2} \cdot \frac{1}{n^3} \leq \frac{1}{n}$.

Now consider the second phase, conditioned on the event that the first phase has indeed hit all sets S_{uv} of size at least $\sqrt{n \ln n}$. If the unknown pair $\{u, v\}$ is a non-edge, after querying u and v we know S_{uv}, and querying the whole set S_{uv} requires at most $\sqrt{n \ln n}$ queries (note that $|S_{uv}| \leq \sqrt{n \ln n}$ if $\{u, v\}$ is a non-edge that hasn't been discovered in the first phase). If the pair $\{u, v\}$ is an edge and the queries at u and v discover a new non-edge, the algorithm proceeds with that non-edge and makes at most $\sqrt{n \ln n}$ further queries (as above), hence at most $\sqrt{n \ln n} + 1$ queries in total for this iteration of the second phase. Otherwise, if the number of neighbors of u and of v is bounded by $\frac{\sqrt{n}}{\sqrt{\ln n}}$, we query also all neighbors of u and v to determine the set S_{uv}, amounting to at most $2\frac{\sqrt{n}}{\sqrt{\ln n}}$ queries, and then the set S_{uv}, giving another $\sqrt{n \ln n}$ queries (since S_{uv} hasn't been hit in the first phase). In total, we make at most $\sqrt{n \ln n} + 2\frac{\sqrt{n}}{\sqrt{\ln n}}$

queries in this iteration of the second phase. Consider the remaining case, i.e., the case where the unknown pair $\{u, v\}$ is an edge, no partial witness for the edge has been queried before, and u or v has degree larger than $\frac{\sqrt{n}}{\sqrt{\ln n}}$. Assume that there are k iterations of the second phase in which the unknown pair falls into this case. Note that no node can be part of an unknown pair in two such iterations. Hence, we get that $2|E| \geq k\frac{\sqrt{n}}{\sqrt{\ln n}}$ and, by Lemma 3, $OPT \geq \frac{|E|}{n} \geq \frac{k\sqrt{n}}{2n\sqrt{\ln n}} = \frac{k}{2\sqrt{n \ln n}}$ and therefore $k \leq 2\sqrt{n \ln n} \cdot OPT$.

Now, let ℓ denote the number of iterations of the second phase in which the set S_{uv} was determined and queried (i.e., all iterations except the k iterations discussed above). We call such iterations *good* iterations. The overall cost of the second phase is at most $\ell\sqrt{n \ln n} + 2\ell\frac{\sqrt{n}}{\sqrt{\ln n}} + 2k$. Clearly, $OPT \geq \ell$, because no two unknown pairs $\{u, v\}$ considered in different good iterations of the second phase can be discovered by the same query (or have the same partial witness). Therefore the cost of the algorithm is at most $3\sqrt{n \ln n} + \ell\sqrt{n \ln n} + 2\ell\frac{\sqrt{n}}{\sqrt{\ln n}} + 2k = O(\sqrt{n \log n}) \cdot OPT$.

So we have that with probability at least $1 - \frac{1}{n}$, the first phase succeeds and $O(\sqrt{n \log n}) \cdot OPT$ queries are made by the algorithm. If the first phase fails, the algorithm makes at most n queries (clearly, the algorithm need not repeat any query). This case increases the expected number of queries made by the algorithm by at most $\frac{1}{n}n = 1$. Thus, we have that the expected number of queries is at most $O(\sqrt{n \log n}) \cdot OPT + \frac{1}{n}n = O(\sqrt{n \log n}) \cdot OPT$. □

6 Conclusions and Future Work

In this paper, we have studied network discovery and network verification in the distance query model. We have shown that the network verification problem is $\mathcal{NP}$-hard and have given an $O(\log n)$-approximation algorithm. For certain graph classes there exist polynomial optimal algorithms or easy characterizations of optimal query sets. For the network discovery problem, we have presented lower bounds of $\Omega(\sqrt{n})$ and $\Omega(\log n)$ on the competitive ratio of deterministic and randomized on-line algorithms, respectively, and designed a randomized on-line algorithm that achieves competitive ratio $O(\sqrt{n \log n})$.

The query model studied in this paper is motivated by real-world scenarios such as discovering the topology of a network that uses a distance-vector routing protocol by analyzing selected routing tables. An interesting direction for future work would be to consider a more realistic model where queries can only be executed at certain nodes of the network; this is motivated by the fact that only a rather small subset of nodes in the Internet or in a network such as Gnutella can actually be used for queries. While our off-line results translate to such a model with forbidden query nodes in a straightforward way, it is not clear whether our on-line algorithm can be adapted to this model or a different approach needs to be employed.

Other query models may be suitable for other applications. For example, a query given by nodes u and v could returns all shortest paths between u and v (or just one shortest path); or a query at node v could return the distances to all nodes that are within distance at most k from v. Changing the objective of the problem leads to other interesting variants, e.g., one could ask for the minimum number of queries that are required to determine the diameter or the value of some other graph parameter of the network.

References

[1] D. Achlioptas, A. Clauset, D. Kempe, and C. Moore. On the bias of traceroute sampling; or, power-law degree distributions in regular graphs. In *Proc. STOC 2005*, 2005.

[2] V. Aggarwal, S. Bender, A. Feldmann, and A. Wichmann. Methodology for estimating network distances of Gnutella neighbors. In *Proceedings of the Workshop on Algorithms and Protocols for Efficient Peer-to-Peer Applications*, INFORMATIK 2004, 2004.

[3] P. Barford, A. Bestavros, J. Byers, and M. Crovella. On the marginal utility of deploying measurement infrastructure. In *Proc. ACM SIGCOMM Internet Measurement Workshop*, November 2001.

[4] Z. Beerliová, F. Eberhard, T. Erlebach, A. Hall, M. Hoffmann, M. Mihal'ák, and L. S. Ram. Network

discovery and verification. In *Proceedings of the International Workshop on Graph-Theoretic Concepts in Computer Science (WG'05)*, LNCS 3787, pages 127–138. Springer, 2005.

[5] J. Cáceres, C. Hernando, M. Mora, I. M. Pelayo, M. L. Puertas, C. Seara, and D. R. Wood. On the metric dimension of cartesian products of graphs. Manuscript, 2005.

[6] Clip2. The Gnutella protocol specification v0.4, 2001. http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf.

[7] L. Dall'Asta, I. Alvarez-Hamelin, A. Barrat, A. Vázquez, and A. Vespignani. Statistical theory of internet exploration. *Physical Review E*, 71, 2005.

[8] L. Dall'Asta, I. Alvarez-Hamelin, A. Barrat, A. Vázquez, and A. Vespignani. Exploring networks with traceroute-like probes: theory and simulations. *Theoretical Computer Science*, 2006. To appear.

[9] G. Di Battista, T. Erlebach, A. Hall, M. Patrignani, M. Pizzonia, and T. Schank. Computing the types of the relationships between autonomous systems. *IEEE/ACM Transactions on Networking*, 2006. To appear.

[10] DIMES. Mapping the Internet with the help of a volunteer community. http:// www.netdimes.org/.

[11] L. Gao. On inferring autonomous system relationships in the internet. *IEEE/ACM Transactions on Networking*, 9(6):733–745, December 2001.

[12] M. R. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. Freeman, 1979.

[13] R. Govindan and A. Reddy. An analysis of internet inter-domain topology and route stability. In *Proc. IEEE INFOCOM 1997*, April 1997.

[14] R. Govindan and H. Tangmunarunkit. Heuristics for Internet map discovery. In *Proc. IEEE INFOCOM 2000*, pages 1371–1380, Tel Aviv, Israel, March 2000.

[15] J. Håstad. Some optimal inapproximability results. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1–10, 1997.

[16] Internet mapping. Project at Lucent Bell Labs. http://www.cs.bell-labs.com/who/ches/map/.

[17] S. Khuller, B. Raghavachari, and A. Rosenfeld. Landmarks in graphs. *Discrete Applied Mathematics*, 70:217–229, 1996.

[18] Oregon RouteViews Project. University of Oregon. http://www.routeviews.org.

[19] SETI. Seti@home website, 2005. http://setiathome.ssl.berkeley.edu/.

[20] L. Subramanian, S. Agarwal, J. Rexford, and R. Katz. Characterizing the internet hierarchy from multiple vantage points. In *Proc. IEEE INFOCOM'02*, 2002.

[21] A. Yao. Probabilistic computations: Towards a unified measure of complexity. In *Proceedings of the 17th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 222–227, 1977.