Network Discovery and Verification with Distance Queries*

Thomas Erlebach[1], Alexander Hall[2], Michael Hoffmann[1], and Matúš Mihaľák[1]

[1] Department of Computer Science, University of Leicester
{te17, mh55, mm215}@mcs.le.ac.uk
[2] Institute for Theoretical Computer Science, ETH Zürich
alex.hall@inf.ethz.ch

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. In the distance query model, a query at node q returns the distances from q to all other nodes in the graph. 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 previous queries. We give a randomized on-line algorithm with competitive ratio $O(\sqrt{n \log n})$ for graphs on n nodes. We also show lower bounds of $\Omega(\sqrt{n})$ and $\Omega(\log n)$ on the competitive ratio of deterministic and randomized on-line algorithms, respectively. In the off-line network verification problem, the graph is known in advance and the problem is to compute a minimum number of queries that 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 growing interest in decentralized networks such as the Internet or peer-to-peer networks has introduced many new algorithmic challenges. A key property of these networks is that there is no central authority that maintains a map of the network. Obtaining an accurate map, usually represented as a graph, is not easy due to the dynamic growth of the network. 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, and combine these in an appropriate manner. There is an extensive body of related work studying various aspects of this approach, see e.g. [14, 9, 15, 12, 13, 11, 3, 16, 8, 1, 6, 7].

As making measurements at a node is usually costly, the problem of minimizing the number of such measurements arises naturally. Nevertheless, it was proposed only recently to study this problem from a combinatorial optimization point of view: Beerliova et al. [4] introduce the network discovery and verification

* 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.

T. Calamoneri, I. Finocchi, G.F. Italiano (Eds.): CIAC 2006, LNCS 3998, pp. 69–80, 2006.

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, 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 are known. 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 Q of queries that verify all edges and non-edges. One motivation for the off-line version is checking with as few measurements as possible whether a given map is still correct.

In order to discover a graph, it may seem sufficient to discover only its edges. However, especially in view of the on-line setting, it is also necessary to have a proof (i.e., discover) for each unconnected node pair that indeed there 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. Considering both also makes it possible to quantify how much knowledge about the network is revealed by a given set of queries. This could also be helpful e.g. when 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 different distances from v. This model was motivated by the consideration that in certain scenarios one can identify all edges on shortest paths between the query node and all other nodes. In this paper, we study network discovery and network verification 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 [5]. 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 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. The most prominent one is probably the RouteViews project [15] by the University of Oregon. It collects data (in the form of lists of paths) from a large number of so-called border gateway protocol routers. More recently, and due to good publicity very successfully, the DIMES project [9] has started collecting data with the help of a volunteer community. Users can download a client that 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 with heuristics to obtain maps of the Internet, basically by simply overlaying the paths found by the respective project, see e.g. [13, 15, 9, 14]. Another line of research aims at inferring from such local views the types of the economic relationships between nodes in the Internet graph [11, 16, 8].

Beerliova et al. [4] propose the 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 distance from q. They give an $o(\log n)$ inapproximability result for the off-line version and a randomized on-line algorithm with competitive ratio $O(\sqrt{n \log n})$. The on-line algorithm we present in this paper is based on a similar approach, but requires new ideas.

Our Results. In Sect. 2 we give basic definitions concerning network discovery and verification in the distance query model. We then characterize the queries that discover an individual non-edge and the sets of queries that together discover an individual edge. (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, however, that more intricate deductions are possible and edges at a larger distance from the query nodes can be discovered.) In Sect. 3 we show lower bounds on the number of queries needed to discover or verify a graph, based on the independence number α, clique number ω, and size of the edge set of the graph. For Dist–All–Verification we present in Sect. 4 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. For Dist–All–Discovery we show in Sect. 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 $O(\sqrt{n \log n})$. Proofs omitted due to space restrictions can be found in [10].

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 the 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 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. First we give a formal notion of what we mean by "discovering" a graph in this model. 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 the off-line setting). The following definitions hold for both terms but for simplicity are stated only for the network discovery setting.

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 holds 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.

If Q discovers G, this implies that any graph G' with $\mathbf{D}_G(Q) = \mathbf{D}_{G'}(Q)$ must have the same edges and non-edges as G, i.e., $G' = G$. 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. 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 that discover a graph G can distinguish it from any other graph G' (sufficient and necessary condition).

Observation 1. *For $G = (V, E)$ the query set $Q \subseteq V$ discovers 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 obvious. To see the second implication "$\Rightarrow$", assume that $\{u, v\}$ is a non-edge and that (for a contradiction) every query node q gives $|d(q, u) - d(q, v)| \leq 1$. Then, if $\{u, v\}$ was an edge, the distances returned by Q would not change, as u and v are either in the same layer or in consecutive layers of each query $q \in Q$. □

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* $\{u, v\}$.

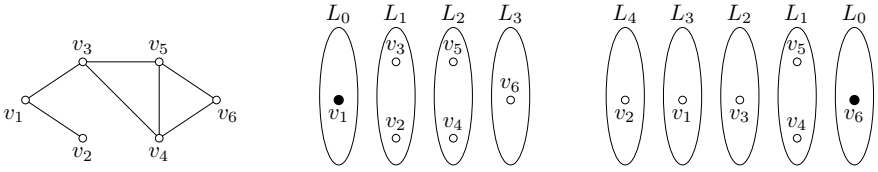

Fig. 1. Edge $\{v_3, v_4\}$ of a graph (left) is discovered by the combination of queries at nodes v_1 and v_6; the distances to the query node v_1 (middle) and v_6 (right) are depicted as layers of the graph

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 it is not necessarily 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 (or both). 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 [10].

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 $L_i \setminus \{u\}$.

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 (or verify) a graph G, based on the independence number α, the clique number ω, and 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.*

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

Proof. Consider a clique K of size ω in G. Let q be the first query. The nodes of K 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 if and only if there is exactly one node v from K in layer i and the remaining nodes of K are in layer $i+1$. Moreover, q is a partial witness only for edges incident with v. After query q, there is still a clique K' of size $\omega - 1$ 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 clique of size 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. □

4 Network Verification

Polynomially Solvable Cases. We discuss some classes of graphs for which the optimal number of queries for network verification can be determined in polynomial time.

Lemma 4. *G can be verified with 1 query if and only if G is a chain. A clique K_n on n vertices needs $n-1$ queries to be verified.*

The example of the cycle with 4 nodes C_4 shows that there is a graph that needs $n-1$ queries to be verified 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, $n > 6$, can be verified optimally with 2 queries.*

Now we characterize the optimal query sets for trees. 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. 2. If the tree is not a chain, there must 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).

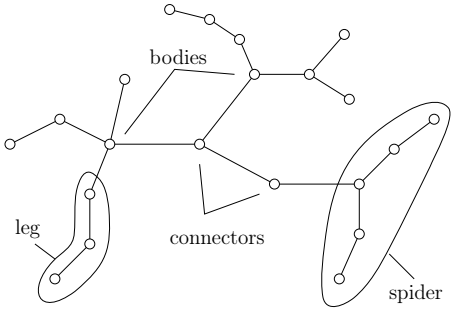

Fig. 2. Legs, bodies, spiders and connectors in a tree

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 verify T is $\sum_{b \in B}(l_b - 1) + |VC(T[B])|$.*

Proof. We show first that we indeed need at least this many queries. Observe that if there is no query in two legs adjacent to a body, then we cannot verify the non-edges formed by vertices of the two legs at the same distance from the body. So, for each body, there must be at least one query in every leg except one. Moreover, if there are two legs of two different bodies which are connected by an edge then there must be at least one query in one of these legs. Otherwise we cannot verify 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. The bodies all of whose legs contain 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 verifies 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. If v and w are from the same leg, the non-edge is clearly verified by any query. If v and w are from different legs, and there is a query q in the leg where v or w is, then this query verifies 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 bodies are adjacent.) Now assume that v and w are from different spiders with bodies 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 $u, x, \ldots, y, u'$,

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 verify 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, possibly $b' = u'$, satisfies $|d(q', v) - d(q', w)| = |(d_v + d) - d_w| \geq 3$ and thus q' verifies 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 verifies all non-edges. We claim now that Q verifies 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. So, Q verifies T. □

Lemma 7. *A query set that verifies 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 verifies H_3. Therefore, the optimal number of queries is 2^{d-1} (size of a minimum vertex cover in H_d) for $d \geq 3$.*

Complexity and Approximability. We can show that Dist–All–Verification is $\mathcal{NP}$-hard by a reduction from the Vertex–Cover problem (see [10]).

Theorem 1. *The problem* Dist–All–Verification *is $\mathcal{NP}$-hard.*

An $O(\log n)$-approximation algorithm for Dist–All–Verification can be obtained using the well-known greedy algorithm for the set cover problem: Each vertex v corresponds to a set containing the non-edges a query at v verifies and the edges for which a query at v is a partial witness, and the goal is to cover all edges and non-edges.

Theorem 2. *For the problem* Dist-All-Verification*, there is an $O(\log n)$-approximation algorithm.*

5 Network Discovery

Lower Bounds for On-line Algorithms. We present lower bounds on the competitive ratio of on-line algorithms for Dist–All–Discovery. Consider the graph G_k from Fig. 3. 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

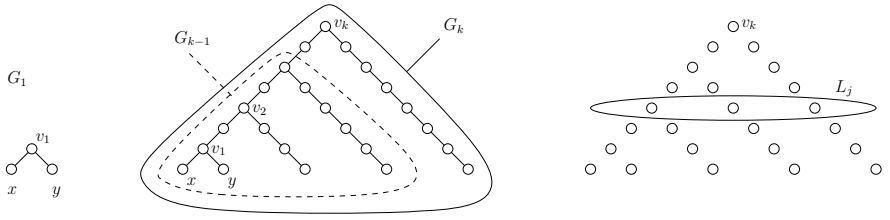

Fig. 3. Graph used in the proof of the lower bound $\Omega(\sqrt{n})$ for on-line algorithms (left and middle); layers after query at vertex v_k (right)

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. No non-edge is discovered within the layers. In particular, the non-edge $\{x, y\}$ in G_1 (see Fig. 3) 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. We can continue recursively in this manner and therefore we can force A to query in every arm before it discovers $\{x, y\}$. Hence, 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 a lower bound of $\Omega(\sqrt{n})$ for deterministic algorithms. Furthermore, from the same construction we can also derive a lower bound for randomized on-line algorithms, see [10] for details.

Theorem 3. *For* Dist–All–Discovery*, there is no $o(\sqrt{n})$-competitive deterministic and no $o(\log n)$-competitive randomized on-line algorithm.*

Randomized On-line Algorithm. We present a randomized algorithm for Dist–All–Discovery. Its competitive ratio $O(\sqrt{n \log n})$ 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.

Theorem 4. *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 whether $\{u, v\}$ 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 had not 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 $\sqrt{n/\ln n}$, 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 $i+2$. The algorithm then queries all vertices in S. Third, if the number of neighbors of u or the number of neighbors of v is larger than $\sqrt{n/\ln n}$, 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 attempts to query 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 $1 - \sqrt{n \ln n}/n = 1 - \sqrt{(\ln n)/n}$. Thus, standard calculations show that the probability that S_{uv} is not hit throughout the first phase is at most $1/n^3$. 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 undiscovered 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 has not 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 $\sqrt{n \ln n} + 2$ queries in total for this iteration of the second phase. Otherwise, if the number of neighbors of u and of v is bounded by $\sqrt{n/\ln n}$, we query also all neighbors of u and v to determine the set S_{uv}, amounting to at most $2\sqrt{n/\ln n}$

queries, and then the set S_{uv}, giving another $\sqrt{n \ln n}$ queries (since S_{uv} has not been hit in the first phase). In total, we make at most $\sqrt{n \ln n}+2\sqrt{n/\ln n}$ queries in this iteration of the second phase. Consider the remaining case, i.e., the case where the undiscovered pair $\{u, v\}$ is an edge, no partial witness for the edge has been queried before, and u or v has degree larger than $\sqrt{n/\ln n}$. Assume that there are k iterations of the second phase in which the undiscovered pair falls into this case. Note that no node can be part of an undiscovered pair in two such iterations. Hence, we get that $2|E| \geq k\sqrt{n/\ln n}$ and, by Lemma 3, $\mathrm{OPT} \geq \frac{|E|}{n} \geq \frac{k\sqrt{n}}{2n\sqrt{\ln n}} = \frac{k}{2\sqrt{n \ln n}}$ and therefore $k \leq 2\sqrt{n \ln n} \cdot \mathrm{OPT}$.

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, $\mathrm{OPT} \geq \ell$, because no two undiscovered pairs $\{u, v\}$ considered in different good iterations can be discovered by the same query (or have the same partial witness). So the cost of the algorithm is at most $3\sqrt{n \ln n} + \ell\sqrt{n \ln n} + 2\ell\frac{\sqrt{n}}{\sqrt{\ln n}} + 2k = O(\sqrt{n \log n}) \cdot \mathrm{OPT}$.

We have that with probability at least $1 - \frac{1}{n}$, the first phase succeeds and $O(\sqrt{n \log n}) \cdot \mathrm{OPT}$ 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 $O(\sqrt{n \log n}) \cdot \mathrm{OPT} + \frac{1}{n}n = O(\sqrt{n \log n}) \cdot \mathrm{OPT}$.

□

6 Conclusions and Future Work

In this paper, we have studied network discovery and network verification in the distance query model. The network verification problem is $\mathcal{NP}$-hard and admits 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 the 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.

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. 37th Ann. ACM Symp. Theory of Computing (STOC'05)*, pages 694–703, 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. Mihaľák, and L. S. Ram. Network discovery and verification. In *Proc. 31st International Workshop on Graph-Theoretic Concepts in Computer Science (WG'05)*, LNCS 3787, pages 127–138. Springer, 2005.
5. Clip2. The Gnutella protocol specification v0.4, 2001. http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf.
6. L. Dall'Asta, I. Alvarez-Hamelin, A. Barrat, A. Vázquez, and A. Vespignani. Statistical theory of internet exploration. *Physical Review E*, 71, 2005.
7. 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.
8. G. Di Battista, T. Erlebach, A. Hall, M. Patrignani, M. Pizzonia, and T. Schank. Computing the types of the relationships between autonomous systems. Submitted to IEEE/ACM Transactions on Networking, 2005.
9. DIMES. Mapping the Internet with the help of a volunteer community. http://www.netdimes.org/.
10. T. Erlebach, A. Hall, M. Mihal'ák, and M. Hoffmann. Network discovery and verification with distance queries. Research Report CS-06-002, Department of Computer Science, University of Leicester, March 2006.
11. L. Gao. On inferring autonomous system relationships in the internet. *IEEE/ACM Transactions on Networking*, 9(6):733–745, December 2001.
12. R. Govindan and A. Reddy. An analysis of internet inter-domain topology and route stability. In *Proc. IEEE INFOCOM 1997*, April 1997.
13. R. Govindan and H. Tangmunarunkit. Heuristics for Internet map discovery. In *Proc. IEEE INFOCOM 2000*, pages 1371–1380, Tel Aviv, Israel, March 2000.
14. Internet Mapping Project. Lucent Bell Labs. http://www.cs.bell-labs.com/who/ches/map/.
15. Oregon RouteViews Project. University of Oregon. http://www.routeviews.org.
16. L. Subramanian, S. Agarwal, J. Rexford, and R. Katz. Characterizing the internet hierarchy from multiple vantage points. In *Proc. IEEE INFOCOM 2002*, 2002.