

Network Discovery and Verification

Zuzana Beerliova, Felix Eberhard, Thomas Erlebach, Alexander Hall,
Michael Hoffmann, Matúš Mihalák, and L. Shankar Ram

Abstract—Due to its fast, dynamic, and distributed growth process, it is hard to obtain an accurate map of the Internet. In many cases such a map—representing the structure of the Internet as a graph with nodes and links—is a prerequisite when investigating properties of the Internet. A common way to obtain such maps is to make certain local measurements at a small subset of the nodes and then to combine these in order to “discover” (an approximation of) the actual graph. Each of these measurements is potentially quite costly. It is thus a natural objective to minimize the number of measurements which still discover the whole graph. We formalize this problem as a combinatorial optimization problem and consider it for two different models characterized by different types of measurements. We give several upper and lower bounds on the competitive ratio (for the on-line network discovery problem) and the approximation ratio (for the off-line network verification problem) in both models. Furthermore, for one of the two models we compare four simple greedy strategies in an experimental analysis.

Index Terms—Internet discovery, complex networks, approximation algorithms, on-line algorithms, random graphs.

I. INTRODUCTION

In recent years, there has been an increasing interest in the study of networks whose structure has not been imposed by a central authority but has arisen from local and distributed processes. Prime examples of such networks are the Internet and unstructured peer-to-peer networks such as Gnutella. For these networks, it is very difficult and costly to obtain a “map” providing an accurate representation of all nodes and the links between them. Such maps would be useful for many purposes, e.g., for studying routing aspects or robustness properties.

In order to create maps of the Internet, a commonly used technique is to obtain local views of the network from various locations (vantage points) and combine them into a map that is hopefully a good approximation of the real network. There is an extensive body of related work studying various aspects of this approach, see e.g. [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12].

More generally, one can view this technique as an approach for discovering the topology of an unknown network by using a certain type of queries—a query corresponds to asking for the local view of the network from one specific vantage point. Such a query at a node is usually very costly (in terms of

time, energy consumption or money). Hence, in this paper, we formalize *network discovery* as a combinatorial optimization problem whose goal is to minimize the number of queries required to discover all edges and non-edges (absent edges) of the network. We study the problem as an on-line problem using competitive analysis. Initially, the network is unknown to the algorithm. To decide the next query to ask, the algorithm can only use the knowledge about the network it has gained from the answers of previously asked queries. Thus, the difficulty in selecting good queries arises from the fact that the amount of information discovered by a query may depend on the parts of the network structure that are still unknown. In the end, the number of queries asked by the algorithm is compared to the optimal number of queries sufficient to discover the network.

In the off-line version of network discovery, the network is known to the algorithm from the beginning. The goal is to compute a minimum number of queries that suffice to discover the network. Although an algorithm for this off-line problem would not be useful for network discovery (if the network is known in advance, there is no need to discover it), it could be employed for network verification, i.e., for checking whether a given map is accurate. Thus, we refer to the off-line version of network discovery as *network verification*. Here, we seek polynomial-time optimal or approximation algorithms.

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. This observation could be helpful e.g. when investigating the quality of published maps of the Internet.

We consider two different query models. In the *layered-graph query model* the answer to a query at a vertex v consists of all edges and non-edges whose endpoints have different distance (number of hops) from v . We refer to the on-line problem in this model as LG-ALL-DISCOVERY and to the off-line problem as LG-ALL-VERIFICATION. In the considerably weaker *distance query model* a query $q \in V$ gives all distances from q to any other node of the investigated graph G . We refer to the corresponding problems as DIST-ALL-DISCOVERY and DIST-ALL-VERIFICATION.

Motivation for Layered-Graph Query Model. The layered-graph query model can be interpreted in the following way: A query at v yields the shortest-path subgraph rooted at v , i.e., the set of all edges on shortest paths between v and any other vertex. To see that this is equivalent to our definition (where

Research partially supported by the EU within the 6th Framework Programme under contract 001907 (DELIS), for which funding in Switzerland is provided by SBF grant 03.0378-1. Preliminary versions of different parts of this work were presented at WG 2005, ECCS 2005, and CIAC 2006.

Zuzana Beerliova, Alexander Hall, and L. Shankar Ram are at the Department of Computer Science, ETH Zürich {bzuzana, alex.hall, lshankar}@inf.ethz.ch

Felix Eberhard is with Syssec Informatik, Lucerne

Thomas Erlebach, Michael Hoffmann, and Matúš Mihalák are at the Department of Computer Science, University of Leicester {te17, mh55, mm215}@mcs.le.ac.uk

a query yields all edges and non-edges between vertices of different distance from v), note that an edge connects two vertices of different distance from v if and only if it lies on a shortest path between v and one of these two vertices. Furthermore, the shortest-path subgraph rooted at v implicitly confirms the absence of all edges between vertices of different distance from v that are not part of the shortest-path subgraph. A real-life scenario where the shortest-path subgraph rooted at a node of the network can be determined arises as follows. With traceroute tools, one can determine the path that packets take in the Internet if they are sent from a node to some destination. If each traceroute experiment returns a random shortest path to the destination, this path would be part of the shortest-path subgraph. One could use repeated traceroute experiments to all destinations to discover all edges of the shortest-path subgraph. Making a query at v would mean getting access to node v and running repeated traceroute experiments from v to all other nodes. If we assume that the cost of getting access to a node is much higher than that of running the traceroute-experiments, minimizing the number of queries is a meaningful goal.

Motivation for the Distance Query Model. The distance query model is much weaker than the layered-graph query model, in the sense that typically a query reveals much less information about the network. There are several reasons that motivate us to study this 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 roughly correspond to reading out the routing table. Another scenario is the discovery of the topology of peer-to-peer networks such as Gnutella [13]. 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 [14]. Repeated Pings could be used to determine the distances to all other nodes.

Without doubt, both of our query models are simplifications of reality. In real networks, routing does not necessarily use shortest paths, and traceroute experiments will often reveal only a single path (or at most a few different paths) to each destination, but not the whole shortest-path subgraph. Real peer-to-peer networks are often so large that it becomes prohibitive to send Pings for larger TTL values, and there are also many other aspects that make the actual discovery of the topology of a Gnutella network very difficult [14]. Nevertheless, we believe that our models constitute a good starting point for a theoretical investigation of fundamental issues arising in network discovery.

A. 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 [3] 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 on the level of autonomous systems—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 [2] 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 in heuristics to obtain maps of the Internet, basically by simply overlaying possible paths found by the respective project, see e.g. [5], [3], [2], [1].

Bejerano and Rastogi [15] propose a two-phase approach for monitoring link delays and faults in IP networks. The first phase corresponds to the problem of verifying all edges of a graph with as few queries as possible in a model which is related to our layered-graph query model, but where query results are trees. They give a SETCOVER [16] based $O(\log n)$ -approximation algorithm and show that the problem is \mathcal{NP} -hard. They state that their reduction can be strengthened to give a lower bound of $\Omega(\log n)$ on the approximation ratio. In contrast to Bejerano and Rastogi, we are interested in verifying (or discovering) both the edges and the non-edges of a graph.

It turns out that the network verification problem in the layered-graph model has previously been considered as the problem of placing landmarks in graphs [17]. Here, the motivation is to place landmarks in as few vertices of the graph as possible in such a way that each vertex of the graph is uniquely identified by the vector of its distances to the landmarks. The smallest number of landmarks that are required for a given graph G is also called the *metric dimension* of G [18]. For a survey of known results, we refer to [19].

The problem of determining whether k landmarks suffice (i.e., of determining if the metric dimension is at most k) is \mathcal{NP} -complete [20]; see [17] for an explicit proof by reduction from 3-SAT. In [17] it is also shown that the problem of minimizing the number of landmarks admits an $O(\log n)$ -approximation algorithm for graphs with n vertices, based on SETCOVER as the algorithm of Bejerano and Rastogi [15] that verifies all edges of a graph. For trees, they show that the problem can be solved optimally in polynomial time. Furthermore, they prove that one landmark is sufficient if and only if G is a path, and discuss properties of graphs for which 2 landmarks suffice. They also show that if k landmarks suffice for a graph with n vertices and diameter D , we must have $n \leq D^k + k$. For d -dimensional hypercubes, it was shown in [21] (using an earlier result from [22] on a coin weighing problem) that the metric dimension is asymptotically equal to $2d/\log_2 d$. See also [23] for further results on the metric dimension of Cartesian products of graphs.

B. Our Results and Outline

In Section II we present basic definitions and preliminaries for the two new query models. For all online algorithms we use

the competitive ratio measure as described in Section II. For LG-ALL-DISCOVERY, we give a lower bound showing that no deterministic on-line algorithm can have competitive ratio better than 3, and we present a randomized on-line algorithm with competitive ratio $O(\sqrt{n \log n})$ for networks with n nodes in Section III-A. For LG-ALL-VERIFICATION, we prove in Section III-B that it cannot be approximated within a factor of $o(\log n)$ unless $\mathcal{P} = \mathcal{NP}$, thus showing that the approximation algorithm from [17] is best possible (up to constant factors). We also give a useful lower bound formula for the optimal number of queries of a given graph. Section III-C presents simulation experiments for four different heuristic discovery strategies on various types of graphs.

In Section IV, we consider the distance query model. Section IV-A gives some lower bounds on the number of queries needed to discover or verify a graph. For DIST-ALL-VERIFICATION we present polynomial time algorithms for several basic graph classes and show that for general graphs, the problem is \mathcal{NP} -hard and admits an $O(\log n)$ -approximation algorithm, see Section IV-B. For DIST-ALL-DISCOVERY, Section IV-C shows 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. We also present a randomized on-line algorithm with competitive ratio $O(\sqrt{n \log n})$.

In Section V, we conclude and point to possible directions for further research.

II. PRELIMINARIES AND PROBLEM DEFINITIONS

Throughout this paper, the term *network* refers to a connected, undirected, unweighted graph. 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 of G is denoted by \bar{E} . We assume that the set V of nodes is known in advance and that it is the presence or absence of edges that needs to be discovered or verified. A *query* is specified by a node $v \in V$ and is called a query *at* v or simply the query v .

A. Layered-Graph Query Model

The answer of a query at v consists of a set E_v of edges and a set \bar{E}_v of non-edges. These sets are determined as follows. Label every vertex $u \in V$ with its distance (number of edges on a shortest path) from v . We refer to sets of vertices with the same distance from v as *layers*. Then E_v is the set of all edges connecting vertices in different layers, and \bar{E}_v is the set of all non-edges whose endpoints are in different layers. Because the query result can be seen as a layered graph, we refer to this query model as the *layered-graph query model*.

A set $Q \subseteq V$ of queries discovers (all edges and non-edges of) a graph $G = (V, E)$ if $\bigcup_{q \in Q} E_q = E$ and $\bigcup_{q \in Q} \bar{E}_q = \bar{E}$. In the off-line case, we also say “verifies” instead of “discovers”. The network verification problem is to compute, for a given network G , a smallest set of queries that verifies G . The network discovery problem is the on-line version of the network verification problem. Its goal is to compute a smallest

set of queries that discovers G . Here, the edges and non-edges of G are initially unknown to the algorithm, the queries are made sequentially, and the next query must always be determined based only on the answers of previous queries.

B. Distance Query Model

In the *distance query model* the answer to a query at v consists of the distances from v to every node of G . By $d_G(u, v)$ we denote the distance from u to v in G . We may omit the subscript G if the graph is clear from the context. Let $\mathbf{D}_G(Q)$, for $Q \subseteq V$, be a collection of distance vectors, one vector $\mathbf{d}_G(Q, v)$ for each $v \in V$. The vector $\mathbf{d}_G(Q, v)$ 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. 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 $d_G(q, v) \neq d_{G'}(q, v)$, and $\mathbf{D}_G(Q) = \mathbf{D}_{G'}(Q)$, 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, 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 the off-line setting). 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 $\bar{e} \in \bar{E}$), if for all graphs $G' = (V, E')$ with $\mathbf{D}_G(Q) = \mathbf{D}_{G'}(Q)$ it must hold that $e \in E'$ ($\bar{e} \in \bar{E}'$). $Q \subseteq V$ discovers the graph G , if it discovers all edges and non-edges of G . This means that 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).

Characterizing the Queries Discovering a Non-Edge. If we look at a particular non-edge $e \in \bar{E}$, 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 $\{u, v\} \in \bar{E}$ 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: 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 a 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 $\{u, v\}$ into 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* $\{u, v\}$.

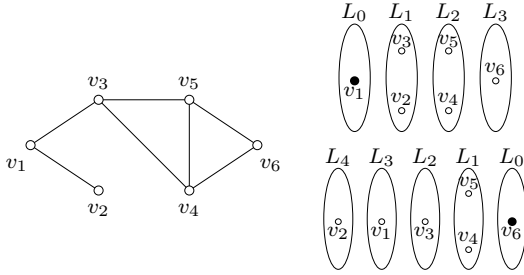


Fig. 1. Edge $\{v_3, v_4\}$ of a graph (left) is discovered by the combination of queries at nodes v_1 and v_6 ; the distances to the query node v_1 (top right) and v_6 (bottom right) are depicted via layers of the graph

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, 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 different 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 for any query $q \in Q$, 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.

C. Approximation and Online Algorithms

We denote by $OPT(G)$, for a given graph G , the cardinality of an optimal query set verifying G , and by $A(G)$ the cardinality of the query set produced by an algorithm A . The quality of an algorithm is measured by the worst possible ratio $A(G)/OPT(G)$ over all networks G . In the off-line case, an algorithm is a ρ -approximation algorithm (and achieves approximation ratio ρ) if it runs in polynomial time and satisfies $A(G)/OPT(G) \leq \rho$ for all networks G . In the on-line case, an algorithm is ρ -competitive (and achieves competitive ratio ρ) if $A(G)/OPT(G) \leq \rho$ for all networks G . It is weakly ρ -competitive if $A(G) \leq \rho \cdot OPT(G) + c$ for some constant c . If the on-line algorithm is randomized, $A(G)$ is replaced by $\mathbb{E}[A(G)]$ in these definitions. As is common in competitive analysis, we do not require on-line algorithms to run in polynomial time.

III. LAYERED-GRAPH QUERY MODEL

A. Network Discovery

We consider the on-line scenario. Clearly, any algorithm that does not repeat queries has competitive ratio at most $n-1$, since $n-1$ queries are always sufficient to discover a network. Furthermore, the inapproximability result that we will derive in Section III-B (Theorem 3) shows that we cannot hope for a polynomial-time on-line algorithm with competitive ratio $o(\log n)$; it may still be possible to obtain such a ratio using exponential-time on-line algorithms, however.

Theorem 1: No deterministic on-line algorithm for LG-ALL-DISCOVERY can have weak competitive ratio $3 - \varepsilon$ for any $\varepsilon > 0$.

Proof: Let A be any deterministic algorithm for LG-ALL-DISCOVERY. We first give a simpler proof that A cannot be better than 2-competitive. Consider Fig. 2(a). We refer to the subgraph induced by the vertices labeled r, x, y , and z as a *2-gadget*. Assume that the given graph G consists of a global root g and $k, k \geq 2$, disjoint copies of the 2-gadget, with the r -vertex of each 2-gadget connected to the global root g . One can easily verify that $OPT(G) = k$ for this graph, and

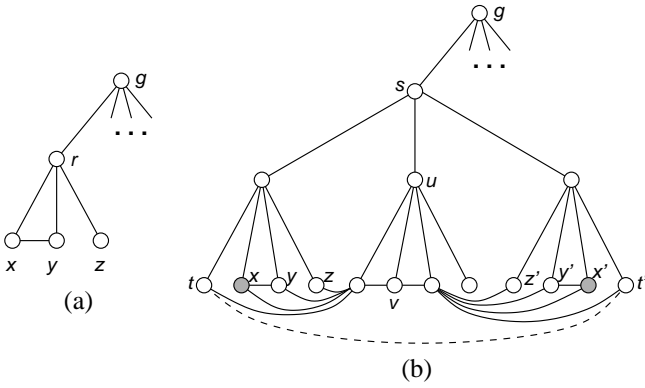


Fig. 2. Lower bound constructions

that the set of all x -vertices (or y -vertices) of the 2-gadgets constitutes an optimal query set. On the other hand, algorithm A can be forced to make the first query at g (as, initially, the vertices are indistinguishable to the algorithm). This will not discover any information about edges or non-edges between vertices x , y and z of each 2-gadget. The only queries that can discover this information are queries at x , y and z . In fact, a query at x or y suffices to discover the edge between x and y and the non-edges between x and z and between y and z . When A makes the first query among the vertices in $\{x, y, z\}$ of a 2-gadget, we can force it to make that query at z , since the three vertices are indistinguishable to the algorithm. The query at z does not discover the edge between x and y . The algorithm must make a second query in the 2-gadget to discover that edge. In total, the algorithm must make at least $2k + 1$ queries. As the construction works for arbitrary values of k , this shows that no deterministic on-line algorithm can guarantee weak competitive ratio $2 - \varepsilon$ for any constant $\varepsilon > 0$.

To get a stronger lower bound of 3, we create a new gadget, called the 3-gadget, as shown in Fig. 2(b). The 3-gadget is the subgraph induced by all vertices except g in the figure. We claim that A can be forced to make 6 queries in each 3-gadget, whereas the optimum query set consists of only 2 vertices in each 3-gadget (drawn shaded in the figure). If we construct a graph with k , $k \geq 2$, disjoint copies of the 3-gadget, the s -vertex in each of them connected to the global root g as indicated in the figure, we get a graph G for which we claim that $OPT(G) = 2k$ and the algorithm A can be forced to make at least $6k + 1$ queries. This shows that no deterministic on-line algorithm can guarantee weak competitive ratio $3 - \varepsilon$ for any constant $\varepsilon > 0$.

To see that $OPT(G) = 2k$, let Q be the set of queries consisting of the two shaded vertices from each copy of the 3-gadget as shown in Fig. 2(b). We claim that Q discovers G . This can be verified manually as follows: For every vertex in a 3-gadget Π , consider the 3-tuple whose components are the distances from that vertex to the two query vertices in Π and the distance to an arbitrary query vertex from Q outside Π . One finds that each vertex in Π has a unique 3-tuple, showing that all edges and non-edges of Π are discovered by Q . Each non-edge between two different 3-gadgets is discovered by one of the queries inside these two 3-gadgets. The edges and non-edges between g and each 3-gadget are also discovered.

Hence, $OPT(G) \leq 2k$. We have $OPT(G) \geq 2k$, because each of the edges $\{x, y\}$ and $\{x', y'\}$ (see Fig. 2(b)) of a 3-gadget requires a separate query.

To show that $A(G) \geq 6k + 1$, we argue as follows. First, we can force A to make the first query at g . This will not reveal any information about edges within the same layer of any of the 3-gadgets. We view each 3-gadget as consisting of s and a left part, a middle part, and a right part. The left part consists of the left child of s and its four adjacent vertices below (these four vertices are called *bottom vertices*, and the left child of s is called the *root* of that part); the middle and right part are defined analogously. The three parts of a 3-gadget Π are indistinguishable to A until it makes its first query inside Π . A query at s would not discover any new information about Π , so we can ignore queries that A might make at s in the following arguments. When A makes its first query inside Π , we can force this query to be in the middle part, and we can force it to be at u or v . In both cases, the query does not discover any information about the edges and non-edges between the bottom vertices of the left part, nor does it discover any information about the edges and non-edges between the bottom vertices of the right part, nor does it discover the edge drawn dashed. When A chooses its second query in Π , it could be in the left part, in the middle part, or in the right part. Assume that A chooses the left part; since the bottom vertices of the left part are still indistinguishable to A , we can force A to make the query either at the root of the left part or at the bottom vertex t . Similarly, in the right part we can force A to make the query at its root or at t' . In the middle part, A can make the query anywhere. In any case, the second query made by A does not discover any information about edges and non-edges between vertices in the set $\{x, y, z\}$ and in the set $\{x', y', z'\}$. Similarly as in the case of Fig. 2(a), for each of these sets we can force A to make the first query at z (at z') and thus require a second query at x or y (at x' or y') to discover everything about these groups. In total, A must make at least 6 queries in each 3-gadget. ■

With the gadget of Fig. 2(a) one can prove easily that no randomized on-line algorithm for LG-ALL-DISCOVERY can have weak competitive ratio $4/3 - \varepsilon$ for any $\varepsilon > 0$; just observe that we can force a randomized algorithm to make the first query at z with probability at least $1/3$. All lower bounds on the weak competitive ratio also hold for the (standard) competitive ratio where no additive constant c is allowed.

Theorem 2: There is a randomized on-line algorithm with competitive ratio $O(\sqrt{n \log n})$ for LG-ALL-DISCOVERY.

Proof: The algorithm consists of 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 node pairs with unknown status exist, it picks an arbitrary such pair $\{u, v\}$ and proceeds as follows. First, it queries u and v in order to determine the distance of all nodes to u and v . From this it can deduce the set S of nodes from which the edge or non-edge between u and v can be discovered; these are simply the nodes for which the distance to u differs from the distance to v . Then, it queries all remaining nodes in S .

To analyze the algorithm, it is helpful to view LG-ALL-DISCOVERY as a HITTINGSET problem [20]. For every edge

or non-edge $\{u, v\}$, let S_{uv} be the set of nodes from which a query discovers $\{u, v\}$. The task of the LG-ALL-DISCOVERY problem translates into the task of computing a subset of V that hits all sets S_{uv} . The goal of the first phase is to hit all sets that have size at least $\sqrt{n \ln n}$ with high probability. If this succeeds, the problem remaining for the second phase is a HITTINGSET problem where all sets have size at most $\sqrt{n \ln n}$. The algorithm of the second phase repeatedly picks an arbitrary set that is not yet hit, and includes all its elements in the solution. As the sets have size at most $\sqrt{n \ln n}$, the number of queries made in the second phase is at most a factor of $\sqrt{n \ln n}$ away from the optimum.

Let us make this analysis precise. Consider a node 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 $\left(1 - \frac{\sqrt{\ln n}}{\sqrt{n}}\right)^{3\sqrt{n \ln n}} \leq e^{-3 \ln n} = \frac{1}{n^3}$. 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 $\binom{n}{2} \cdot \frac{1}{n^3} \leq \frac{1}{n}$.

Consider the second phase, conditioned on the event that the first phase has hit all sets S_{uv} of size at least $\sqrt{n \ln n}$. In each iteration the algorithm asks at most $\sqrt{n \ln n}$ queries. Let ℓ be the number of iterations. It is clear that the optimum must make at least ℓ queries, because no two unknown pairs $\{u, v\}$ considered in different iterations of the second phase can be resolved by the same query.

Since $OPT(G) \geq 1$ and $OPT(G) \geq \ell$, the number of queries made by the algorithm is at most $3\sqrt{n \ln n} + \ell\sqrt{n \ln n} = O(\sqrt{n \log n}) \cdot OPT(G)$.

With probability at least $1 - \frac{1}{n}$, the first phase succeeds and the algorithm makes $O(\sqrt{n \log n}) \cdot OPT(G)$ queries. If the first phase fails, the algorithm makes at most n queries. This case increases the expected number of queries made by the algorithm by at most $\frac{1}{n} \cdot n = 1$. Thus, the expected number of queries is at most $O(\sqrt{n \log n}) \cdot OPT(G) + \frac{1}{n} \cdot n = O(\sqrt{n \log n}) \cdot OPT(G)$. ■

B. Network Verification

As mentioned in the introduction, the problem LG-ALL-VERIFICATION is known to be \mathcal{NP} -hard, and there is a SETCOVER-based $O(\log n)$ -approximation algorithm for it [17]. In this section, we first show that it is impossible to achieve approximation ratio $o(\log n)$ unless $\mathcal{P} = \mathcal{NP}$.

Theorem 3: It is \mathcal{NP} -hard to approximate LG-ALL-VERIFICATION within ratio $o(\log n)$.

Proof: We prove the inapproximability result using an approximation-preserving reduction from the *test collection problem (TCP)*: For a given ground set S and collection \mathcal{C} of subsets of S , find a minimum cardinality subset $\mathcal{C}' \subseteq \mathcal{C}$ such that for each two distinct elements x and y of S , there exists a set $C \in \mathcal{C}'$ such that exactly one of x and y is in C .

In the original application for TCP, S is a set of diseases and \mathcal{C} is a collection of tests. A test $C \in \mathcal{C}$, applied to a patient, will give a positive result if the patient is infected by

a disease in C . If a patient is known to be infected by exactly one of the diseases in S , the goal of TCP is to compute a minimum number of tests that together can uniquely identify that disease.

Without loss of generality, we can restrict ourselves to instances of TCP in which any two elements of the ground set can be separated by at least one of the sets in \mathcal{C} ; instances without this property do not have any feasible solutions.

Halldórsson et al. [24] prove that TCP cannot be approximated with ratio $o(\log |S|)$ unless $\mathcal{P} = \mathcal{NP}$. Their proof uses an approximation-preserving reduction from SETCOVER [20]; the latter problem was shown \mathcal{NP} -hard to approximate within $o(\log n)$, where n is the cardinality of the ground set, by Arora and Sudan [25]. The proof by Arora and Sudan establishes the inapproximability result for SETCOVER even for instances in which the size of the ground set and the number of sets are polynomially related. The reduction from SETCOVER to TCP maintains this property. Hence, we know that it is \mathcal{NP} -hard to approximate TCP with ratio $o(\log |S|)$ even for instances satisfying $|\mathcal{C}| \leq |S|^g$ for some positive constant g .

Let an instance (S, \mathcal{C}) of TCP be given. Let $n_{\text{TCP}} = |S|$ and $m_{\text{TCP}} = |\mathcal{C}|$. By the remark above, we can assume that $m_{\text{TCP}} = n_{\text{TCP}}^{O(1)}$. We construct an instance $G = (V, E)$ of LG-ALL-VERIFICATION as follows. First, we add $n_{\text{TCP}} + m_{\text{TCP}}$ vertices to V : an *element vertex* v_s for every element $s \in S$ and a *test vertex* u_C for every $C \in \mathcal{C}$. We initially add the following edges to E : Any two element vertices are joined by an edge, and every test vertex u_C is joined to all element vertices v_s with $s \in C$. So the element vertices form a complete graph (clique). The idea behind this construction is that queries at test vertices verify all edges in the clique of element vertices if and only if the corresponding tests form a test cover. We have to extend the construction slightly since, in LG-ALL-VERIFICATION, the edges and non-edges incident to the test vertices need to be verified as well. We add $h = 2(\lceil \log m_{\text{TCP}} \rceil + 2)$ auxiliary vertices w_1, \dots, w_h to take care of this. For each i , $1 \leq i \leq h/2$, the auxiliary vertices w_{2i-1} and w_{2i} are said to form a *pair*. In addition, we add one extra node z . We add the following edges:

- The two auxiliary vertices in each pair are joined by an edge.
- Number the m_{TCP} test vertices arbitrarily from 0 to $m_{\text{TCP}} - 1$. Both auxiliary vertices in the i -th pair, $1 \leq i \leq h/2 - 2$, are joined to those of the m_{TCP} test vertices whose number has a 1 in the i -th position of its binary representation.
- Both auxiliary vertices in the last two pairs are joined to all test vertices.
- The extra node z is joined to all other vertices of the graph.

The graph constructed in this way is denoted by $G = (V, E)$. See Fig. 3 for an illustration. We prove two claims:

Claim 1: Given a solution \mathcal{C}' to the TCP instance (S, \mathcal{C}) , there is a solution Q of the constructed instance $G = (V, E)$ of LG-ALL-VERIFICATION satisfying $|Q| = |\mathcal{C}'| + \lceil \log m_{\text{TCP}} \rceil + 2$.

Proof (of Claim 1). Let a solution \mathcal{C}' to the TCP instance (S, \mathcal{C}) be given. Let Q contain all test vertices corresponding to sets

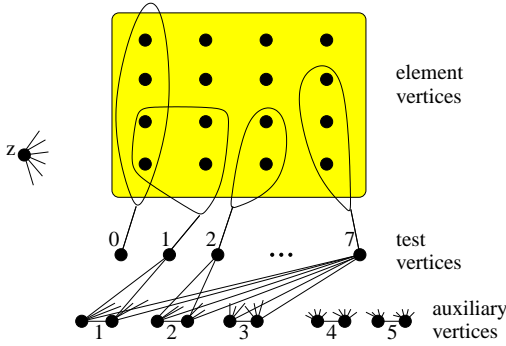


Fig. 3. Illustration of the construction of the graph $G = (V, E)$ that is an instance of LG-ALL-VERIFICATION. The auxiliary vertices in pairs 4 and 5 are adjacent to all test vertices. The auxiliary vertices in pair i , $1 \leq i \leq 3$, are adjacent to the test vertices whose number has a 1 in position i of the binary representation. For example, the auxiliary vertices in pair 2 are adjacent to test vertices 2, 3, 6 and 7

$C \in \mathcal{C}'$ as well as the first vertex of every pair of auxiliary vertices. Obviously, we have $|Q| = |\mathcal{C}'| + \lceil \log m_{\text{TCP}} \rceil + 2$. One can verify that Q discovers all edges and non-edges of G . ■

Claim 2: Given a solution Q to the constructed instance $G = (V, E)$ of LG-ALL-VERIFICATION, one can construct in polynomial time a solution \mathcal{C}' of the original TCP instance (S, \mathcal{C}) satisfying $|\mathcal{C}'| \leq |Q| - \lceil \log m_{\text{TCP}} \rceil - 2$.

Proof (of Claim 2). Observe that Q must contain at least one vertex from each pair of auxiliary vertices; otherwise, the edge joining this pair would not be discovered. The queries at these vertices do not discover any edges between element vertices (all element vertices are at distance 2 from any auxiliary vertex because of the extra vertex z). Let Q' be the vertices in Q that are not auxiliary vertices. We have $|Q'| \leq |Q| - \lceil \log m_{\text{TCP}} \rceil - 2$. Now, Q' is a set of element vertices and test vertices that, in particular, discovers all edges between element vertices.

Let Q_S be the set of element vertices in Q' and let Q_C be the set of test vertices in Q' . If Q_S is empty, the queries at the vertices in Q_C discover all edges of the clique of element vertices. In particular, this means that for any two distinct element vertices v_s and v_t in V , there must be a query at a vertex adjacent to one of v_s, v_t but not to the other. This shows that the set $\mathcal{C}' = \{C \in \mathcal{C} \mid u_C \in Q'\}$ is a solution of the original TCP instance of the required size.

Now assume Q_S is nonempty. The set of edges between element vertices that are not discovered by Q_C is a disjoint union of cliques. The queries in Q_S must discover all edges in these cliques. As the only edges between element vertices that a query at an element vertex discovers are the edges incident to that vertex, a clique of size k requires $k-1$ queries. Assume that there are p cliques and denote the number of vertices in these cliques by k_1, \dots, k_p . Then Q_S contains at least $\sum_{i=1}^p (k_i - 1)$ vertices. All edges in a clique of size k can always be discovered by $k-1$ queries at test vertices: simply select these queries greedily by choosing, as long as there is an edge $\{u, v\}$ in the clique that has not yet been discovered, any test vertex that is adjacent to one of u, v but not the other. Hence, we can replace the queries in Q_S by at most $\sum_{i=1}^p (k_i - 1)$ queries at test vertices and add these to Q_C , obtaining a set of queries at test vertices that discovers all

edges between element vertices. As in the previous paragraph, this set of test vertices gives a solution to the original TCP instance of cardinality at most $|Q'|$. ■

The claims imply that an approximation algorithm with ratio $o(\log n)$ for LG-ALL-VERIFICATION would allow approximating TCP with ratio $o(\log n_{\text{TCP}})$. To show this, we can argue as follows. Assume there is an approximation algorithm A for LG-ALL-VERIFICATION that achieves ratio $o(\log n)$, where $n = |V|$. Consider the algorithm B for TCP that, given an instance of TCP, constructs an instance of LG-ALL-VERIFICATION as described above, applies A to this instance, and transforms the result into a solution to the TCP instance following Claim 2. Recall that $m_{\text{TCP}} = n_{\text{TCP}}^{O(1)}$. We claim that B achieves ratio $o(\log n_{\text{TCP}})$ for TCP. Let OPT_{TCP} be the optimum objective value for the given TCP instance and OPT_{LG} be the optimum objective value for the constructed instance of LG-ALL-VERIFICATION. Let B_{TCP} and A_{LG} denote the objective values of the solutions computed by B and A , respectively. Note that $OPT_{\text{TCP}} \geq \log n_{\text{TCP}}$ always holds, since n_{TCP} elements cannot be separated by fewer than $\log n_{\text{TCP}}$ test sets.

Claims 1 and 2 imply that $OPT_{\text{TCP}} = OPT_{\text{LG}} - \lceil \log m_{\text{TCP}} \rceil - 2$. We have $OPT_{\text{LG}} = OPT_{\text{TCP}} + \lceil \log m_{\text{TCP}} \rceil + 2 \leq OPT_{\text{TCP}} + O(\log n_{\text{TCP}}) = O(OPT_{\text{TCP}})$. Claim 2 implies $B_{\text{TCP}} \leq A_{\text{LG}}$ and thus we get $B_{\text{TCP}} \leq o(\log n) \cdot OPT_{\text{LG}} = o(\log n) \cdot O(OPT_{\text{TCP}}) = o(\log n_{\text{TCP}}) \cdot O(OPT_{\text{TCP}})$, where the last equality follows from $n = n_{\text{TCP}} + m_{\text{TCP}} + 2(\lceil \log m_{\text{TCP}} \rceil + 2) + 1 = n_{\text{TCP}}^{O(1)}$. This shows $B_{\text{TCP}} \leq o(\log n_{\text{TCP}}) \cdot OPT_{\text{TCP}}$ and completes the proof of Theorem 3. ■

We have seen that no approximation algorithm can have approximation ratio $o(\log n)$ for the problem LG-ALL-VERIFICATION unless $\mathcal{P} = \mathcal{NP}$. Now we investigate the optimal number of queries for specific graphs. First, it is easy to see that $OPT(G) = 1$ if and only if G is a path. Cycles have $OPT(G) = 2$, and cliques on n vertices have $OPT(G) = n - 1$. Finally, we give a useful lower bound on $OPT(G)$ for specific graphs G .

Theorem 4: If a graph $G = (V, E)$ contains a subgraph H of diameter D_H with n_H vertices, then $OPT(G) \geq \log_{D_H+1} n_H$.

Proof: Imagine the queries being performed sequentially. At any instant, the unknown edges and non-edges induce disjoint cliques, which we call *unknown groups*. Two vertices are in the same unknown group if and only if they were in the same layer of all queries made so far. Consider the n_H vertices of subgraph H . Initially, all vertices form an unknown group. For each query, the n_H vertices of H will be in at most D_H+1 consecutive layers of the layered graph returned by the query. Therefore, after the first query, at least $n_H/(D_H+1)$ vertices of H will still be in the same unknown group. Similarly, after k queries, at least $n_H/(D_H+1)^k$ vertices of H will be in an unknown group together. If k queries suffice to verify all edges and non-edges, the unknown groups must be singletons in the end. So we must have $n_H/(D_H+1)^k \leq 1$. ■

This theorem implies that a graph containing a clique on k vertices requires at least $\log_2 k$ queries, and a graph with maximum degree Δ at least $\log_3(\Delta+1)$ queries. For the

former, take H to be the clique on k vertices, and for the latter, take H to be the subgraph induced by a vertex of degree Δ and its neighbors.

C. Experiments

1) *Heuristic Strategies and Simulation Setup:* We have implemented several strategies for selecting the next query vertex: Strategy CURRENT selects the next query so as to maximize the number of newly discovered non-edges under the assumption that all edges have already been discovered. Strategies OPTIMISTIC and PESSIMISTIC compute upper and lower bounds on the number of newly discovered edges and non-edges at each potential query vertex and select the query that maximizes the upper or lower bound, respectively. For comparison, we have also implemented a strategy OFFLINE that knows the network and computes, using a greedy SETCOVER heuristic, a small set of queries to discover the whole network. Since OFFLINE has full information, it can be expected to perform better than the three other strategies, and we thus employ it as a benchmark strategy.

In our simulation experiments, we use various types of generated network graphs (Erdős-Rényi random graphs [26], Barabási-Albert scale-free random graphs [27], Dorogovtsev-Mendes-Samukhin scale-free random graphs [28], and grid graphs) and also snapshots of the Internet graph at seven different dates. They are obtained by querying up to 23 BGP routers. The data is from the RouteViews project [3] and is compiled on the web pages [29]. On all graphs we run the implemented discovery strategies.

2) *Results:* First, we compare the different strategies with each other concerning the number of queries required to discover the whole graph in each of the different random graph models. We generate graphs with $n = 1,000$ nodes and varying average degree, and we record the average number of queries that each of the strategies requires. The results for Erdős-Rényi random graphs are shown in Fig. 4. The chart on the left-hand side covers smaller degrees and uses a logarithmic scale on the vertical axis, whilst the chart on the right-hand side covers a larger range of degrees and uses a linear scale. We find that the number of queries required by PESSIMISTIC, CURRENT and OFFLINE is roughly the same and slightly smaller than that of OPTIMISTIC. This indicates that a good selection of queries can be made in spite of the initial lack of information about the graph structure. We also find that the absolute number of queries required to discover a network is often surprisingly small. For networks with 1,000 nodes and average degree ranging from 10 to 250, only 10–60 queries are usually sufficient to discover the whole network. The results are similar for the Barabási-Albert and Dorogovtsev-Mendes-Samukhin scale-free random graphs, with slightly worse performance of OPTIMISTIC. The number of queries increases sharply as the average degree approaches the border cases of 0 and $n - 1$. This is to be expected, since for both the empty and the complete graph $n - 1$ queries are needed. Perhaps more surprising is the peak at a low average degree of about 70. Investigating this phenomenon may be an interesting research question.

Grid graphs were the only graphs for which OPTIMISTIC outperformed PESSIMISTIC (see Fig. 5; here, we have varied the number of nodes). This is due to the special property of grids that two queries in adjacent corners are sufficient to discover the graph, independent of the number of nodes. Note that in the experiments for each grid size we ran the strategies on 8 different random permutations of the nodes. This is to make sure that ties (if a strategy can choose from several different nodes for the next query) are broken randomly.

In the charts of Fig. 4 we also show the number of queries required by CURRENT in order to discover 95% of the edges and 95% of the non-edges (labeled “95%, CURR.”). It is a striking observation that more than 10 queries are rarely needed for this purpose; this also holds for the Barabási-Albert and Dorogovtsev-Mendes-Samukhin scale-free random graphs. In practice, being able to discover such a large fraction of the graph may already be satisfactory, especially if this comes with apparently large savings. Actually, in networks with a high frequency of change, such as the Internet, it is perhaps unrealistic to hope for more than the discovery of a large portion of the network.

We have also studied how the number of queries required to discover scale-free random graphs changes if the size of the network grows. The results for strategy OFFLINE and Barabási-Albert graphs are shown in Fig. 6. The chart shows the number of queries needed for networks with increasing number of nodes and fixed average degree. As expected, one finds that when the number of nodes is increased, the number of queries required to discover the network increases as well. However, when checking how many queries suffice to discover 95% of the edges and 95% of the non-edges of a graph (labeled “95%, OFFLINE”), we notice that for Barabási-Albert graphs this number appears to be a constant smaller than 10 as the number of nodes varies from 100 to 12,800. This indicates that extremely few queries can be sufficient to discover a huge portion of an unknown network, independent of the size of the network. For Dorogovtsev-Mendes-Samukhin random graphs with average degree 4, however, the number of queries for 95% discovery grows linearly with the number of nodes (not shown as a chart). Only when the average degree is larger (achieved by varying the generation process so that each new node is made adjacent to the endpoints of several randomly chosen edges), the number of queries required for 95% discovery appears to be a constant for this model as well.

Table I shows how the OFFLINE strategy performs on snapshots of the Internet graph. The numbers are surprisingly high, but it is quite promising that only 4 or 5 queries suffice to verify a large portion of the graph. The “vantage points” entries are the number of routers that were actually queried to obtain the snapshot. The entries “unknown” give the number of pairs that were actually not yet discovered by these queries, under the assumption of the layered-graph query model.

IV. DISTANCE QUERY MODEL

A. Lower bounds

In this section we show lower bounds on the number of queries needed to discover G in the distance query model. We

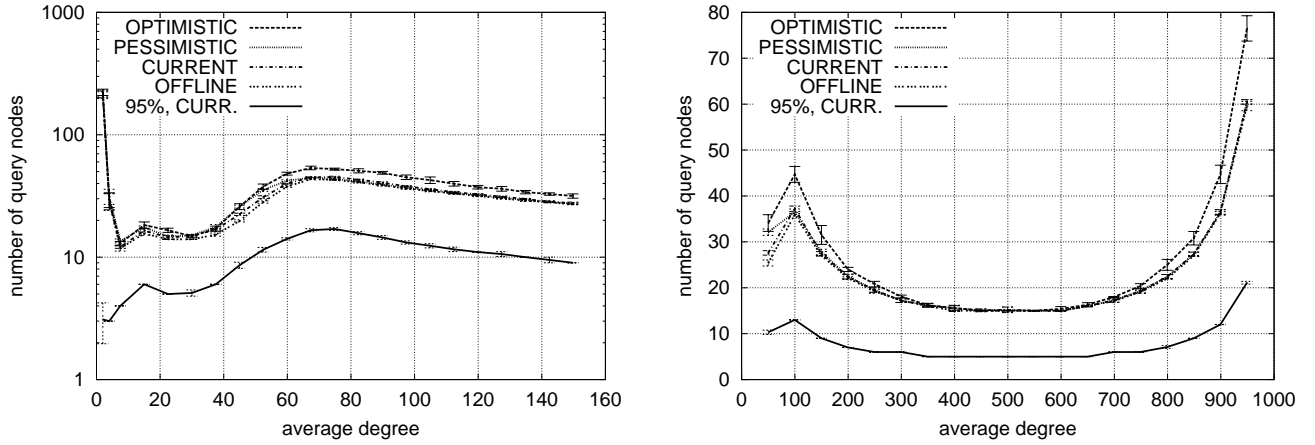


Fig. 4. $G_{n,p}$: Erdős-Rényi random graphs on $n = 1,000$ nodes. Between a pair of nodes an edge is present with probability p . This parameter is varied. In the charts the expected degree $p \cdot (n - 1)$ of a node is given on the abscissa.

TABLE I

SNAPSHOTS OF THE INTERNET GRAPH AND THE NUMBER OF QUERIES OFFLINE NEEDS TO VERIFY THE ENTIRE GRAPH OR 95% OF IT.

	2001.04.	2002.02.	2002.04.	2002.07.	2003.01.	2003.06.	2004.02.
vertices / edge	10923 / 23935	12788 / 27936	13164 / 28510	13409 / 24804	14710 / 30992	15806 / 36534	16932 / 37534
vantage pts. / unknown	10 / $2.02 \cdot 10^6$	12 / $3.18 \cdot 10^6$	19 / $1.04 \cdot 10^6$	12 / $1.91 \cdot 10^6$	10 / $2.17 \cdot 10^6$	23 / $1.83 \cdot 10^6$	23 / $1.08 \cdot 10^6$
OFFLINE	5674	6730	6962	7467	7782	7857	8691
95%, OFFLINE	4	4	4	4	4	5	4

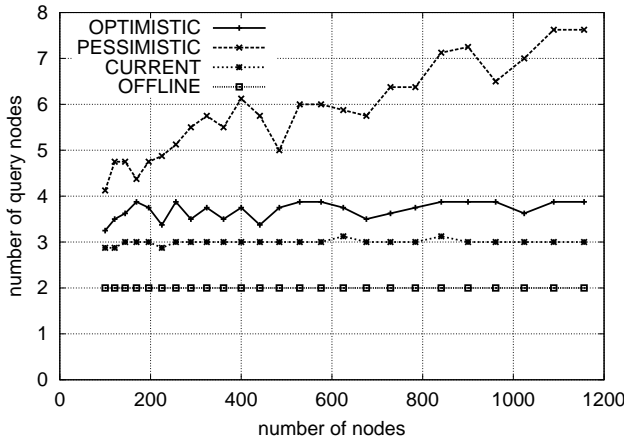


Fig. 5. Grids of increasing sizes from 10×10 to 34×34 . The number of nodes is given on the abscissa.

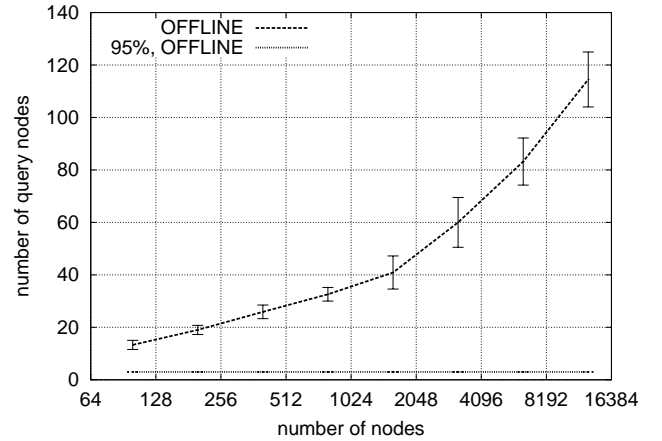


Fig. 6. Barabási-Albert random graphs. The average degree is fixed to 4. The number of nodes is increased in exponential steps.

relate this number to the clique number ω of the graph and to the number of edges m .

Lemma 1: 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 2: 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. ■

B. Network Verification

1) Polynomially Solvable Cases:

Lemma 3: 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 , note that q cannot discover an edge or non-edge between two vertices at the same distance from q . Thus, the vertices of G have unique distances from q , and so G is a chain.

The second part of the statement follows from Lemma 1 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. Interestingly, it is not very difficult to show that two queries suffice for cycles on at least 7 nodes.

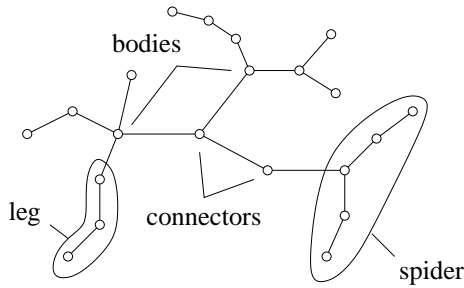


Fig. 7. Legs, bodies, spiders and connectors in a tree

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. 7. 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 4: 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.
- 3) 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, \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 discovers 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 5: 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., $u = 000 \dots 0$ and $v = 100 \dots 0$ and $q = q_1 q_2 \dots q_d$, where $q_1 = 0$. There must exist $i > 1$ such that $q_i = 1$. Then $w = \underbrace{10 \dots 0}_{i-1} 10 \dots 0$ 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., $u = 00\dots 0$ and $v = \underbrace{1\dots 1}_k 0\dots 0$, $k \geq 2$. If

$k = d$, i.e., v is antipodal to u then $10\dots 0$ is a neighbor of u and therefore in the vertex cover. $10\dots 0$ 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 $0\dots 01$ (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 the size of any vertex cover is 2^{d-1} . One can easily check that all vertices with even Hamming distance to the origin $00\dots 0$ form a vertex cover and the number of such vertices is 2^{d-1} . ■

2) \mathcal{NP} -Hardness: We consider the complexity of the DIST-ALL-VERIFICATION problem and show that it is \mathcal{NP} -hard. The start with an easy lemma.

Lemma 6: 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 and only if the distances from q to u and v differ by at least 2. Since the graph has diameter 2, 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 5: DIST-ALL-VERIFICATION is \mathcal{NP} -hard.

Proof: We present a polynomial-time reduction from the VERTEXCOVER 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 6 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 VERTEXCOVER, 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' . 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 . Therefore, 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 6. Furthermore, 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 $|Q| - 2n + 2$.

This 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 VERTEXCOVER is \mathcal{NP} -hard to approximate within a factor of $7/6 - \varepsilon$ by [30], the problem DIST-ALL-VERIFICATION is \mathcal{NP} -hard. ■

3) *Approximation Algorithm:* We present an $O(\log n)$ -approximation algorithm based on the well-known greedy algorithm for the SETCOVER problem.

Theorem 6: 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 SETCOVER 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 SETCOVER approximation algorithm (see, e.g., [16]) gives an approximation ratio of $O(\log |E \cup \overline{E}|) = O(\log n)$. ■

C. Network Discovery

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.

Consider the graph G_k from Fig. 8. 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

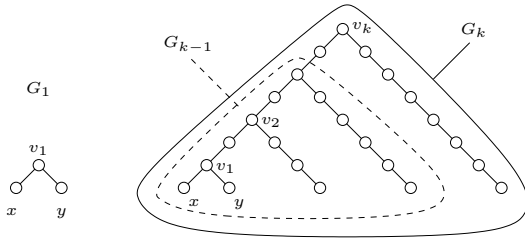


Fig. 8. Graph used in the proof of the lower bound $\Omega(\sqrt{n})$ for on-line algorithms

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 4, 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 Fig. 8) 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 each 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 7: There is no $o(\sqrt{n})$ -competitive deterministic on-line algorithm for DIST-ALL-DISCOVERY.

The following theorem states our lower bound for randomized on-line algorithms.

Theorem 8: 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 [31]: 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 iso-

morphic to G_k (as shown in Fig. 8). 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 $i \in \{1, 2, \dots, k-1\}$. (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 $OPT = 2$ and applying Yao's principle, we obtain the theorem. ■

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.

Theorem 9: There is a randomized on-line algorithm with competitive ratio $O(\sqrt{n \log n})$ for DIST-ALL-DISCOVERY.

Proof: The algorithm is based on similar ideas as the one of Theorem 2. We view the problem as a HITTINGSET problem. For every non-edge $\{u, v\}$ let S_{uv} be the set of vertices that discover $\{u, v\}$. For every edge $\{u, v\}$ let S_{uv} be the set of all partial witnesses for $\{u, v\}$. The algorithm discovers the whole graph G if it hits all sets S_{uv} , for $\{u, v\} \in E \cup \bar{E}$.

The algorithm runs in two phases. In the first phase it makes $3\sqrt{n \ln n}$ queries at nodes chosen uniformly at random. As shown in the proof of Theorem 2, the probability that at least one of the sets S_{uv} of cardinality at least $\sqrt{n \ln n}$ is not hit in the first phase is at most $\frac{1}{n}$.

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 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 $\frac{\sqrt{n}}{\sqrt{\ln n}}$, 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 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 and proceeds with choosing another undiscovered pair $\{u', v'\}$ (if one exists).

We analyze the algorithm as follows. Let OPT be the optimal number of queries. 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 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 $\frac{\sqrt{n}}{\sqrt{\ln n}}$, 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} has not 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 2, $OPT \geq \frac{|E|}{n} \geq \frac{k\sqrt{n}}{2n\sqrt{\ln n}} = \frac{k}{2\sqrt{n \ln n}}$ and therefore $k \leq 2\sqrt{n \ln n} \cdot OPT$.

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). So the cost of the algorithm is at most $3\sqrt{n \ln n} + \ell\sqrt{n \ln n} + 2\ell \frac{\sqrt{n}}{\sqrt{\ln n}} + 2k = O(\sqrt{n \log n}) \cdot OPT$. As in the proof of Theorem 2, it follows

that the algorithm is $O(\sqrt{n \log n})$ -competitive. ■

V. CONCLUSIONS AND FUTURE WORK

In this paper, we have considered network discovery and network verification problems in the layered-graph query model and in the distance query model. In the layered-graph query model, the major problem left open by our work is to close the gap between our randomized upper bound of $O(\sqrt{n \log n})$ and the small constant lower bounds. In the distance query model, the existence of an $o(\log n)$ -approximation algorithm remains as an interesting open problem.

Our simulation results show that, under the query model considered, it is possible to discover accurate information about the structure of large and complex networks using a moderately small number of queries. If the goal is to discover 95% of the edges and of the non-edges, it even appears that a constant number of queries is often sufficient, independent of the network size. In future work, we would like to investigate this effect from a more theoretical perspective.

It would also be interesting to study further query models. For example, a query could be given by nodes u and v and return all shortest paths between u and v (or just one shortest path); 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 discover a fixed percentage of edges and non-edges or to determine the diameter or some other property of the network.

REFERENCES

- [1] B. Cheswick and H. Burch, "Internet mapping project." [Online]. Available: <http://www.cs.bell-labs.com/who/ches/map/>
- [2] DIMES, "Mapping the Internet with the help of a volunteer community." [Online]. Available: <http://www.netdimes.org/>
- [3] Oregon RouteViews, "University of Oregon RouteViews project." [Online]. Available: <http://www.routeviews.org>
- [4] R. Govindan and A. Reddy, "An analysis of internet inter-domain topology and route stability," in *Proc. IEEE INFOCOM*, April 1997.
- [5] R. Govindan and H. Tangmunarunkit, "Heuristics for Internet map discovery," in *Proc. IEEE INFOCOM*, March 2000, pp. 1371–1380.
- [6] L. Gao, "On inferring autonomous system relationships in the internet," *IEEE/ACM Trans. Networking*, vol. 9, no. 6, pp. 733–745, Dec 2001.
- [7] 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.
- [8] L. Subramanian, S. Agarwal, J. Rexford, and R. Katz, "Characterizing the internet hierarchy from multiple vantage points," in *Proc. IEEE INFOCOM*, 2002.
- [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 Trans. Networking*, 2006, to appear.
- [10] 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 STOC*, 2005.
- [11] L. Dall'Asta, I. Alvarez-Hamelin, A. Barrat, A. Vázquez, and A. Vespignani, "Statistical theory of internet exploration," *Phys. Rev. E*, vol. 71, 2005.
- [12] L. Dall'Asta, I. Alvarez-Hamelin, A. Barrat, A. Vázquez, and A. Vespignani, "Exploring networks with traceroute-like probes: theory and simulations," *Theoret. Comput. Sci.*, 2005, to appear.
- [13] Clip2, "The Gnutella protocol specification v0.4," 2001. [Online]. Available: http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf

- [14] 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 at INFORMATIK 2004*, 2004.
- [15] Y. Bejerano and R. Rastogi, "Robust monitoring of link delays and faults in IP networks," in *Proc. IEEE INFOCOM*, 2003.
- [16] V. V. Vazirani, *Approximation Algorithms*. Springer, 2001.
- [17] S. Khuller, B. Raghavachari, and A. Rosenfeld, "Landmarks in graphs," *Discrete Appl. Math.*, vol. 70, pp. 217–229, 1996.
- [18] F. Harary and R. Meller, "The metric dimension of a graph," *Ars Combin.*, pp. 191–195, 1976.
- [19] G. Chartrand and P. Zhang, "The theory and applications of resolvability in graphs: A survey," *Congr. Numer.*, vol. 160, pp. 47–68, 2003.
- [20] M. R. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*. Freeman, 1979.
- [21] A. Sebő and E. Tannier, "On metric generators of graphs," *Math. Oper. Res.*, vol. 29, no. 2, pp. 383–393, 2004.
- [22] B. Lindström, "On a combinatorial detection problem I," *Magyar Tud. Akad. Mat. Kutató Int. Közl.*, vol. 9, pp. 195–207, 1964.
- [23] 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," 2005, manuscript.
- [24] B. V. Halldórsson, M. M. Halldórsson, and R. Ravi, "On the approximability of the minimum test collection problem," in *Proc. 9th ESA*, ser. LNCS 2161. Springer-Verlag, 2001, pp. 158–169.
- [25] S. Arora and M. Sudan, "Improved low-degree testing and its applications," in *Proc. 29th STOC*, 1997, pp. 485–495.
- [26] P. Erdős and A. Rényi, "On random graphs I," *Publ. Math. Debrecen*, vol. 6, pp. 290–297, 1959.
- [27] A. L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, pp. 509–512, 1999.
- [28] S. N. Dorogovtsev, J. F. F. Mendes, and A. N. Samukhin, "Size-dependent degree distribution of a scale-free growing network," *Phys. Rev. E*, vol. 63, p. 062101, 2001.
- [29] S. Agarwal, "Collection of internet snapshots," <http://www.cs.berkeley.edu/~sagarwal/research/BGP-hierarchy/>.
- [30] J. Håstad, "Some optimal inapproximability results," in *Proc. 29th STOC*, 1997, pp. 1–10.
- [31] A. Yao, "Probabilistic computations: Towards a unified measure of complexity," in *Proc. 17th FOCS*, 1977, pp. 222–227.



Thomas Erlebach received a Ph.D. in Computer Science from Technische Universität München in 1999. From 2000 to 2004, he was an assistant professor in Theory of Communication Networks at ETH Zürich. In September 2004 he joined the Department of Computer Science at the University of Leicester as a Reader in Algorithms. His research interests lie in the design and analysis of efficient algorithms for optimization problems arising in communication networks and other application areas.



Alexander Hall received a Master's degree ("Diplom") in Computer Science at the Technische Universität München in 1998. In December 2003 he completed his doctoral studies in the group of Thomas Erlebach at the ETH Zürich and received a Ph.D. for the thesis "Scheduling and Flow-Related Problems in Networks". He currently is a post-doc at ETH Zürich.



Michael Hoffmann obtained a M.Sc. degree in Mathematics from Sussex University in 1997. He obtained his Ph.D. in Computer Science from the the University of Leicester in 2001; his thesis contained the solution of several open problems in the theory of automatic monoids. In November 2001 he became a lecturer in the Computer Science Department at Loughborough University, and, in September 2002, he was appointed to a lectureship in Computer Science at the University of Leicester. His interest lies in computational group and semigroup theory as well as in the wider area of online algorithms.



Zuzana Beerliova received a Master's degree ("Diplom") in Mathematics at ETH Zürich in 2004. She currently is a Ph.D. student in the Information Security and Cryptography Research Group of Ueli Maurer at ETH Zürich.



Matúš Mihalák received his Master Degree (Magister) from Comenius University Bratislava in 2002. He then started his Ph.D. at ETH Zurich and later transferred to University of Leicester, where he is currently finishing his thesis on various algorithmic problems in communication networks.



Felix Eberhard received a Master's degree ("Diplom") in Computer Science at the Swiss Federal Institute of Technology in Zurich (ETHZ) in 2004. He is currently employed as a software engineer at Syseca Informatik in Lucerne.



L. Shankar Ram received a Masters' degree (M.Sc. (Engg)) in Computer Science at the Indian Institute of Science, Bangalore in 2002. He currently is a Ph.D. student at the Institute of Theoretical Computer Science, ETH Zurich. His research interests lie in the area of combinatorial optimization, approximation algorithms and game theory.