Discovery of Network Properties with All-Shortest-Paths Queries

Discovery of Network Properties with All-Shortest-Paths Queries*

Technical Report Nr. 591, Department of Computer Science, ETH Zurich

Davide Bilò[†], Thomas Erlebach[‡], Matúš Mihalák[†], and Peter Widmayer[†]

April 2008

Abstract

We consider the problem of discovering properties (such as the diameter) of an unknown network $G(V, E)$ with a minimum number of queries. Initially, only the vertex set V of the network is known. Information about the edges and non-edges of the network can be obtained by querying nodes of the network. A query at a node $q \in V$ returns the union of all shortest paths from q to all other nodes in V. We study the problem as an online problem – an algorithm does not initially know the edge set of the network, and has to decide where to make the next query based on the information that was gathered by previous queries. We study how many queries are needed to discover the diameter, a minimal dominating set, a maximal independent set, the minimum degree, and the maximum degree of the network. We also study the problem of deciding with a minimum number of queries whether the network is 2-edge or 2-vertex connected. We use the usual competitive analysis to evaluate the quality of online algorithms, i.e., we compare online algorithms with optimum offline algorithms. For all properties except maximal independent set, 2-vertex connectivity and minimum/maximum degree, we present and analyze online algorithms. Furthermore we show, for all the aforementioned properties, that "many" queries are needed in the worst case. As our query model delivers more information about the network than the measurement heuristics that are currently used in practise, these negative results suggest that a similar behavior can be expected in realistic settings, or in more realistic models derived from the all-shortest-paths query model.

1 The Problem and the Model

Dynamic large-scale networks arise in our everyday life naturally, and it is no surprise that they are the subject of current research interest. Both the natural sciences and the humanities have their own stance on that topic. A basic prerequisite is the network itself, and thus, before any study can even begin, the actual representation (a map) of a network has to be obtained. This can be a very difficult task, as the network is typically dynamic, large, and the access to it may be limited. For example, a map of the Internet is difficult to obtain, as the network consists of many autonomous nodes, who organize the physical connections locally, and thus the network lacks any central authority or access point.

There are several attempts to obtain an (approximate) map of the Internet. A common approach, on the level of Autonomous Systems (ASs), is to inspect routing tables and paths stored in each router (passive measurement) or directly ask the network with a traffic-sending probe (active

*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.

[†]Institute of Theoretical Computer Science, ETH Zurich, CH-8092 Zurich, Switzerland. {dbilo, mmihalak, widmayer}@inf.ethz.ch

[‡]Department of Computer Science, University of Leicester, University Road, Leicester LE1 7RH, United Kingdom. t.erlebach@mcs.le.ac.uk

measurement). All these methods are commonly called *traceroute*-like measurements. For example, the Oregon Route-Views (RV) project [11] is based on the analysis of the Border Gateway Protocol (BGP) routing tables on the level of ASs. Essentially, for each BGP router its list of paths (to all other AS nodes in the network) is retrieved. More recently, and due to good publicity very successfully, the Distributed Internet Measurements and Simulations (DIMES) project [6] has started collecting data with the help of a volunteer community. Users can download a client which collects paths in the Internet by executing successive traceroute commands. A central server can direct each client individually by specifying which routes to investigate. Data obtained by these or similar projects has been used in heuristics to obtain (approximate) maps of the Internet, basically by simply overlaying possible paths found by the respective project, see e.g. [4, 7, 6, 11].

As performing such measurements at a node is usually very costly (in terms of time, energy consumption or money), the question of minimizing the number of such measurements arises naturally. This problem was formalized as a combinatorial optimization problem and studied in [2]. The map of a network (and the network itself) is modeled as an undirected graph $G = (V, E)$. The nodes V represent the communication entities (such as ASs in the Internet) and the edges represent physical or logical communication links. A measurement at a node $v \in V$ of the network is called a *query at* v, or simply a *query* v. Each query q gives some information about the network. The *network discovery* problem asks for the minimum number of queries that discover the whole network. In [2] the *layered-graph query model* (LG for short) is defined: a query q returns the union of all shortest paths from q to every other node. In this paper we refer to the LG query model as the *all-shortest-paths* query model. Network discovery is an online problem, where the edges and non-edges (a pair $\{u, v\}$ is a non-edge, if it is not an edge) are initially not known and an algorithm queries vertices of V one by one, until all edges and non-edges are discovered.

Having a map of a network G at our disposal, various aspects of G can be studied. For example, the routing aspects of G are influenced by the diameter, average degree, or connectivity of G. Other graph properties that are studied in the networking community include, for example, a maximal/maximum independent set, minimal/minimum dominating set, shell index, the decision whether the graph is bipartite, power-law, etc. All these properties can be computed from the map of G.

If only a single parameter of a network is desired to be known, obtaining the whole map of the network may be too costly. In this work we address the problem of computing (an approximation of) network properties (such as the diameter of G) in an online way: given an unknown network (only the nodes are known in the beginning), discover a *property* (or an approximation of a property) of the network (graph) with a minimum number of queries. The properties that we address in this paper are the diameter of the graph, a minimal dominating set, a maximal independent set, minimum degree, maximum degree, edge connectivity and vertex connectivity. We use standard graph-theoretic terminology and notation, as it is described for example in [5].

We assume the all-shortest-paths query model, i.e., a query q returns the union of all shortest paths from q to every other node. The result of the query q can be viewed as a layered graph: all the vertices at distance i from q form a layer $L_i(q)$, and the query returns all information between any two layers, i.e., if u and v are from different layers, then the query returns whether $\{u, v\}$ is an edge or a non-edge. We depict the result of a query graphically as in Figure 1. For simplicity we sometimes write L_i instead of $L_i(q)$, if it is clear from the context which node is queried. We denote by E_q and $\overline{E}_q$ the set of edges and non-edges, respectively, that are discovered by query q. In the all-shortest-paths query model, E_q is the set of edges whose endpoints have different distance from q, and $\overline{E}_q$ is the set of non-edges whose endpoints have different distance from q. By E_Q and $\overline{E}_Q$ we denote the set of edges and non-edges that are discovered by queries Q, i.e., $E_Q = \bigcup_{q \in Q} E_q$ and $\overline{E}_Q = \bigcup_{q \in Q} \overline{E}_q$. The graph G_Q is the graph on V with the edge set E_Q. Finally, we denote by $\texttt{comp}(G, Q)$, the set of all graphs G' with vertex set V containing all the edges in E_Q and all non-edges in $\overline{E}_Q$.

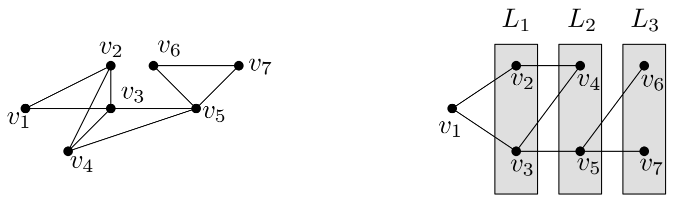

Figure 1: A graph G (left) and the result of a query at node v_1 as a layered graph (right)

It is easy to observe that querying all vertices of G discovers all the edges and non-edges of G and thus any property of the graph can be derived from this information. We are interested in algorithms that deliver minimum-sized query sets that reveal the necessary information about the sought network property. An online algorithm for the (approximate) discovery of a network property is called *c-competitive*, if the algorithm delivers, for any input graph G, a query set Q of size at most $c \cdot \mathrm{Opt}$, where Opt is the optimum number of queries that discover the (approximation of the) property. By an approximate discovery of a property we understand a computation of a value A which is "close" to the actual value O of the property. We require $A \geq O$, if we want to approach O from above (we call the property a *minimization property*), or $A \leq O$, if we want to approach O from below (we call the property a *maximization property*). We will treat the diameter as a minimization property. We call an online algorithm a *ρ-approximation* algorithm for the problem of discovering a minimization property if for any input graph G it discovers a ρ-approximation of the property, i.e., if for the numerical value A returned by the algorithm, and the actual value O of the property, we have $O \leq A \leq \rho \cdot O$. For example, a ρ-approximation, c-competitive algorithm for the diameter discovery problem is an algorithm that discovers a graph G_Q for which the diameter diam_{G_Q} is at most $\rho \cdot \mathrm{diam}_G$, and queries at most c times more queries than an optimal offline ρ-approximation algorithm.

Related Work. Deciding exactly (and deterministically) a graph-theoretic property of a given graph where the measure of quality is the number of accessed entries in the adjacency matrix of the graph is a well understood area. Rivest and Vuillemin [9] show that any deterministic procedure for deciding any non-trivial monotonous n-vertex graph property must examine $\Omega(n^2)$ entries in the adjacency matrix representing the graph. Each such examination of an entry can be seen as a query. Our approach introduces a general concept where other types of queries can be considered. We study the case where the query at a vertex returns all shortest paths from that vertex. This is, however, not the only possible query model to study, and we expect that other interesting query models will be studied following this concept. Moreover, in contrast to the previous work, we study the problem as an online problem, and thus evaluate the quality of algorithms using the competitive ratio.

An active and related field of research is the well-established area of *property testing*, in which a graph property is asked to be probabilistically examined with possibly few edge-queries on the edges of the graph. The aim of such property-testing algorithms is to spend time that is sub-linear or even independent of the size of the graph. In property testing, a graph possessing an examined property $\mathcal{P}$ shall be declared by the algorithm to have property $\mathcal{P}$ with probability at least 3/4, and a graph that is "far" from having property $\mathcal{P}$ should be declared by the algorithm not to have property $\mathcal{P}$ with probability at least 3/4. A survey on property testing can be found for example in [10]. Our work differs from property testing in the type of query we make, and in that we consider deterministic strategies.

The all-shortest-paths query model was introduced by Beerliová et al. for studying the mapping process of large-scale networks [2]. The authors studied the problem of discovering all edges and all non-edges of an unknown network with as few queries as possible. They presented, among other results, a randomized $O(\sqrt{n \log n})$-competitive algorithm, and lower bounds 3 and 4/3 on the competitive ratio of any deterministic and randomized algorithm, respectively. A query set that discovers the edges and non-edges of the network is also called a *resolving set* and the minimum-size resolving set is called a *basis* of the underlying graph, and the size of the basis is the *dimension* of the graph. A graph-theoretic and algorithmic overview of this topic can be found in [3] and [1], respectively.

Our Contribution. We consider several graph properties in the property discovery setting with the all-shortest-paths query model. We first study the discovery of the diameter of an unknown graph G. We present and use a new technique of querying an "interface" between two parts of a graph G. Using k "interfaces" leads to a $(1+\frac{1}{k+1})$-approximation algorithm for the discovery of the diameter of G. The "interface" is in our case a layer of vertices which are at the same distance from an initial query q_0. Considering the competitive ratio as well, and setting $k=1$, we can present a $(\frac{3}{2}+\frac{2p-1}{\ell})$-approximation, $(\frac{n}{2p})$-competitive algorithm, where ℓ is the maximum distance from q_0 (which is at least half of the diameter of G), and p is a parameter, $p<\ell/4$. We present a lower bound $\sqrt{n}-1/2$ for the competitive ratio of any algorithm computing a minimal dominating set. We also present an algorithm which queries at most $O(\sqrt{d\cdot n})$ vertices, where d is the size of a minimum dominating set of G. For the problem of finding a maximal independent set we show a lower bound $\sqrt{n}$ on the competitive ratio of any algorithm. We further study the discovery of 2-edge and 2-vertex connectivity of G, and show a lower bound $n/2$ on the competitive ratio of any algorithm for discovering a bridge or an articulation vertex of G. We also present an $n/2$-competitive algorithm which discovers whether G is 2-edge connected. For the problem of discovering the maximum and the minimum degree of G, we present lower bounds $n/2$ and $n/2$, respectively, for the competitive ratios of any algorithm.

2 Discovering the Properties

In the following we use a common approach to the (approximate) discovery of a graph property of a given graph G: select a query set Q such that the resulting graph G_Q has the same (or approximately similar) graph property.

2.1 Discovering the Diameter

Following the general approach, we want to find a (possibly) small query set Q, such that the resulting graph $G_Q=(V,E_Q)$ has a diameter which is a good approximation of the diameter of G.

It has been previously observed [8] that a single query $q\in V$ yields a 2-approximation of the diameter of G. To see this, let q be a vertex of G. Let v be the vertex with the maximum distance from q. Let ℓ denote this distance, i.e., $d(q,v)=\ell$. Clearly, diam $\geq \ell$. Also, for any two nodes $u,v\in V$, $d(u,v)\leq d(u,q)+d(v,q)\leq 2\ell$. Thus, the diameter of G_q is at most 2ℓ, and therefore it is at most twice the diameter of G.

The following example shows that in general, unless we discover the whole network, we cannot hope for a better approximation than 2. Consider two graphs: $G_1=K_n$, the complete graph, and $G_2=K_n\setminus\{u,v\}$, the complete graph minus one edge $\{u,v\}$. The diameter of G_1 is 1, and the diameter of G_2 is 2. For any query q, but u or v, the result looks all the same, a star graph centered at q. Thus, we know that the diameter is at most 2, but cannot obtain a better approximation until all the vertices (but one) are queried. As any deterministic algorithm can be forced to query $V\setminus\{u,v\}$ first, the example shows that there is no deterministic $(2-\epsilon)$-approximation algorithm with less than $n-1$ queries.

If the diameter of the graph is larger than two (e.g. a growing function in n, such as $\log n$), the following strategy guarantees a better approximation ratio. We first make an arbitrary query $q\in V$. This splits the vertices of V into layers L_i, $i=1,2,\ldots,\ell$, where L_i contains the vertices at distance

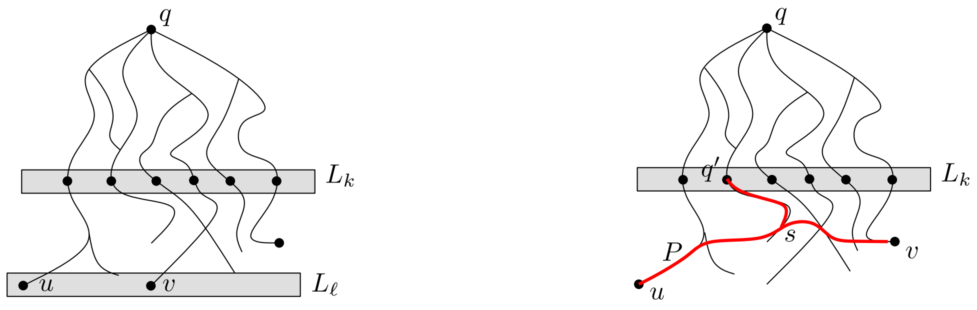

Figure 2: The initial query q splits the vertices of G into ℓ layers $L_1, L_2, \dots L_\ell$. The distance $d(u,v)$ between any two nodes $u, v \in V$ is at most $d(u,q) + d(q,v) \le 2\ell$, but can be shorter if edges within the same layer are present

i from q. As a next step we query all vertices at layer L_k (we will show that $k = \frac{3}{4}\ell$ is a good choice). See Figure 2 for an illustration of the upcoming discussion. From the information that we gain after querying all vertices in L_k we want to improve the upper bound or the lower bound for the diameter, and thus the approximation ratio of our algorithm. Thus, the algorithm computes the diameter of $G' := G_{\{q\} \cup L_k}$ (the discovered part of G), and reports it as the approximate solution. In the following we discuss the quality of such an approximation. Let u and v be the vertices whose distance is the diameter of G'.

If a shortest path between u and v in G goes via vertices of the queried layer L_k, the actual distance (in G) between u and v will be discovered in G' (and the approximation ratio will be 1). Thus, we concentrate on the cases where the shortest path between u and v does not go via L_k.

Case 1. If u and v lie both within layers $L_1, \dots, L_{k-1}$, then clearly $d_{G'}(u,v) \le 2(k-1)$. This type of nodes guarantees an approximation ratio of $2(k-1)/\ell$ (as the diameter of G is at least ℓ).

Case 2. If both u and v lie within layers $L_{k+1}, \dots, L_\ell$, and every shortest path in G between u and v goes via vertices of layers $L_{k+1}, \dots, L_\ell$, we can obtain the following bounds on $d_G(u,v)$. Trivially, $d_G(u,v) \le d_G(u,q') + d_G(q',v) = d_{G'}(u,q') + d_{G'}(q',v)$, for any $q' \in L_k \cup \{q\}$. Let P be a shortest path in G between u and v. Let $s \in V$ be a vertex on P that is closest to L_k and let q' be a vertex in L_k which is closest to s. We obtain $d_G(q',u) \le d_G(q',s) + d_G(s,u) \le (\ell - k) + d_G(s,u)$, and similarly $d_G(q',v) \le d_G(q',s) + d_G(s,v) \le (\ell - k) + d_G(s,v)$. Thus, $d_G(q',u) + d_G(q',v) \le 2(\ell-k) + d_G(s,u) + d_G(s,v) = 2(\ell-k) + d_G(u,v)$. As $d_G(q',u) = d_{G'}(q',u)$ and $d_G(q',v) = d_{G'}(q',v)$, we obtain $d_{G'}(q',u) + d_{G'}(q',v) - 2(\ell-k) \le d_G(u,v) \le d_{G'}(q',u) + d_{G'}(q',v)$, and the approximation ratio obtained for this type of vertices is at most $\frac{d_{G'}(q',u)+d_{G'}(q',v)}{\max\{\ell, d_{G'}(q',u)+d_{G'}(q',v)-2(\ell-k)\}}$. We now distinguish two cases. First, if $d_{G'}(q',u) + d_{G'}(q',v) - 2(\ell-k) \le \ell$, then the approximation ratio is at most $\frac{\ell+2(\ell-k)}{\ell} = \frac{3\ell-2k}{\ell}$. Second, if $d_{G'}(q',u) + d_{G'}(q',v) - 2(\ell-k) > \ell$, then the approximation ratio is of the form $\frac{x}{x-2(\ell-k)}$, which is maximized (under the condition that $x - 2(\ell-k) \ge \ell$) for $x = \ell + 2(\ell-k)$. Thus the approximation ratio is at most $\frac{3\ell-2k}{\ell}$.

Taking all cases into account, the approximation ratio of the algorithm is $\max\{1, \frac{2(k-1)}{\ell}, \frac{3\ell-2k}{\ell}\}$. To minimize the approximation ratio, we need to set $2(k-1) = 3\ell - 2k$, i.e., $k = \frac{3\ell+2}{4}$, which leads to $\text{diam}_{G_Q}/\text{diam}_G \le \frac{3}{2} - \frac{1}{\ell}$.

We assume, for simplicity of presentation, that every fractional computation results in an integral number (such as the query level $k = \frac{3\ell+2}{4}$). In reality one has to round the numbers, which can "shift" the queried layer by half, i.e., $|[k] - k| \le 0.5$ (by $[k]$ we denote the rounding of k). This results in a small additive error of order $\frac{1}{\ell}$ in the approximation ratio of the diameter. Observe that this error approaches zero, as ℓ (and the diameter) grows with n. For instance, we obtain that $\text{diam}_{G_Q}/\text{diam}_G \le \frac{2([k]-1)}{\ell} = \frac{2([\frac{3\ell+2}{4}]-1)}{\ell} \le \frac{2(\frac{3\ell+2}{4}+0.5-1)}{\ell} = \frac{3}{2}$. Therefore $\text{diam}_{G_Q} \le \frac{3}{2}\text{diam}_G$ (compare with the original bound $\text{diam}_{G_Q} \le (\frac{3}{2} - \frac{1}{\ell})\text{diam}_G$). Thus, for simplicity, we sometimes omit these small rounding errors in the statements about approximation ratios.

Theorem 1 *Let G be any graph. A query set $Q = \{q\}$ results in a graph $G_Q = (V, E_Q)$ such that* $\mathrm{diam}_{G_Q} \leq 2 \cdot \mathrm{diam}_G$. *Let ℓ be the maximum distance from q to a vertex of G. Setting $Q = \{q\} \cup L_{\alpha\ell}(q)$, $\alpha < 1$, then the approximation ratio ρ of the algorithm (which computes* diam_{G_Q} *as the approximation of* diam_G*) is $\max\{2\alpha, 3 - 2\alpha\}$. For $\alpha = \frac{3}{4} + \frac{1}{2\ell}$, the approximation ratio is $\frac{3}{2}$.*

It is not difficult to imagine that querying more layers leads to a better approximation of the diameter. This is indeed the case. For example, if we query two layers L_k and L_s, $k < s$, we obtain the following bounds on the approximation ratio. The query q, layer L_k and layer L_s divide the nodes naturally into three parts P_1, P_2 and P_3 (where P_1 consists of nodes with distance less than k from q, part P_2 consists of nodes with distance from q between k and s, and part P_3 consists of nodes with distance from q greater than s). For nodes u and v that lie in different parts or in the queried layers, the upper bound on their distance is the actual distance and hence they are not critical for the approximation ratio. If the nodes u and v are from P_1, we get a bound $\frac{2(k-1)}{\ell}$, if they are from P_3 we get a bound $\frac{\ell+2(\ell-s)}{\ell} = \frac{3\ell-2s}{\ell}$, and if they are from P_2 we get a bound $\frac{\ell+2(s-k-1)}{\ell}$ on the approximation ratio. The first two bounds can be obtained analogously as in the case where only one layer was queried. The bound for vertices within P_2 is derived similarly as the bound for vertices in P_3. Observe that if the shortest path between u and v lies completely within P_2, then there is a query q' from L_k which is at distance at most $(s - k - 1)$ from the path. Thus, similarly as in the case where we queried a single layer L_k, $d_{G'}(q', u) + d_{G'}(q', v) - 2(s - k - 1) \leq d_G(u, v) \leq d_{G'}(q', u) + d_{G'}(q', v)$. Setting $k = \frac{4}{6}\ell + \frac{1}{3}$, $s = \frac{5}{6}\ell + \frac{2}{3}$, the graph G_Q has a diameter $\mathrm{diam}_{G_Q} \leq \frac{4}{3}\mathrm{diam}_G$.

We can generalize the approach to s layers. The previous discussion of case 1 shows that querying a layer $k \leq \ell/2$ does not bring any improvement in the analysis of the approximation of the diameter. Hence, all the s queried layers shall lie within layers L_j, $j > \ell/2$. To obtain the best approximation ratio, the queried layers $L_{k_1}, L_{k_2}, \ldots, L_{k_s}$ have to be chosen evenly from the layers $L_{\ell/2}, \ldots, L_\ell$, so that the queried layers and the layers $L_{\ell/2}$ and L_ℓ are uniformly spaced. It is not difficult to verify that such a choice of s queries leads to a $(1 + \frac{1}{s+1})$-approximation.

Theorem 2 *Let ℓ be the maximum distance from an initial query q to a vertex of G. Let $Q = \{q\} \cup L_{k_1} \cup L_{k_2} \cup \ldots L_{k_s}$, $s \geq 1$, $k_i < k_{i+1}$, $i = 1, \ldots, s-1$, where $k_i = \ell/2 + i \cdot \frac{\ell}{2(s+1)}$. Then the query set Q leads to a graph G_Q for which the diameter* diam_{G_Q} *is a $1 + \frac{1}{s+1}$ approximation of the diameter of G.*

So far we have been mainly concerned with the quality of the approximation but we did not consider the number of queries we make. A problem of the previous algorithm is that the right choice of layer L_k where we make the queries may result in many queries (say, $n-\ell$ in the worst case, if the layer L_k contains almost all vertices of G). If we want to maintain a bounded competitive ratio, we have to be careful about the choice of L_k, which leads to a bi-criteria optimization problem.

Bi-criteria Optimization. To keep some control over the number of queries, a natural idea is to allow some freedom in the choice of the layer L_k. Thus, we do not set $k = \frac{3}{4}\ell + 0.5$, but parametrize the choice of k and allow k to be in the range $\{\frac{3}{4}\ell + 0.5 - p, \ldots, \frac{3}{4}\ell + 0.5 + p\}$, where p is a parameter. The algorithm now picks the layer L_k with the minimum number of vertices among all layers L_i, $i \in \{\frac{3}{4}\ell + 0.5 - p, \ldots, \frac{3}{4}\ell + 0.5 + p\}$. Thus, the size of L_k is at most $n/2p$, which is also the upper bound on the competitive ratio of the algorithm. Relaxing p allows to keep the number of queries small, but can harm the approximation quality, while setting p very small improves the approximation but leaves no control over the number of queries. Clearly, a meaningful choice of p is in the range $\{0, 1, 2, \ldots, \frac{1}{4}\ell - 0.5\}$.

Repeating the previous case analysis, the upper bounds on the approximation ratio for the different cases are 1, $2(k-1)/\ell$, and $\frac{3\ell-2k}{\ell}$. As $3\ell - 2k \leq 3\ell - 2(\frac{3}{4}\ell + 0.5 - p) = \frac{3}{2}\ell - 1 + 2p$ and

$2(k-1) \leq 2(\frac{3}{4}\ell + 0.5 + p - 1) = \frac{3}{2}\ell + 2p - 1$ we obtain that the approximation ratio is $\max\{1, 3/2 + \frac{2p-1}{\ell}, 3/2 + \frac{2p-1}{\ell}\} = 3/2 + \frac{2p-1}{\ell}$. The parameter p can be used to tweak the approximation ratio and the competitive ratio of the algorithm, which are $3/2 + \frac{2p-1}{\ell}$ and $n/2p$, respectively.

Theorem 3 *Let G be any graph and q a query which results in ℓ layers. Then there is an algorithm, parametrized by $p \in \{0, 1, 2, \ldots, \lfloor \frac{1}{4}\ell - 0.5 \rfloor\}$, which delivers a $(3/2 + \frac{2p-1}{\ell})$ approximation of the diameter of an unknown graph, and is $n/2p$ competitive.*

2.2 Discovering a Minimal Dominating Set

In this section we consider the problem of discovering a minimal dominating set in G. We provide an algorithm that discovers a minimal dominating set of G with $O(\sqrt{d \cdot n})$ queries, where d is the size of a minimum dominating set of G. The algorithm, which we simply call *Alg*, works as follows. It starts from an empty set D and grows it by adding vertices step by step so that D will eventually be a minimal dominating set. At each step, *Alg* queries two vertices x and y (an *x-vertex* and a *y-vertex*, respectively). The first vertex x is chosen arbitrarily among the vertices that are not yet dominated by D. The algorithm queries x and the information of the query decides the next choice of vertex y; y is chosen among the set of neighbors of x in such a way that it maximizes the set of *newly dominated* nodes by y (i.e., the subset of neighbors $N(y)$ of y which are at distance 2 from x and which are not neighbors of any vertex belonging to our partial solution D). Both x and y are put into D. It can happen that the query x has only one layer, and hence y does not dominate any new vertex, and thus D is not minimal (y can be removed from D). Similarly, if y dominates all neighbors of x and some vertices from $L_2(x)$, x is obsolete, and D is not minimal. Thus, at the end, we modify D to make it minimal. The procedure is described in Algorithm 1.

Algorithm 1 The algorithm for discovering a minimal dominating set in G

```
Input: The vertex set of a graph $G = (V, E)$.
Output: A minimal dominating set $D \subseteq V$ of $G$.
 1: $D \leftarrow \emptyset$                     // dominating set
 2: $X \leftarrow \emptyset$
 3: $Y \leftarrow \emptyset$
 4: $U \leftarrow V$                     // set of uncovered nodes
 5: while $U \neq \emptyset$ do
 6:     Query any node $x \in U$
 7:     $X \leftarrow X \cup \{x\}$
 8:     Let $y \in L_1(x)$ be a node that maximizes $|N(y) \cap U \cap L_2(x)|$
 9:     Query $y$
10:     $Y \leftarrow Y \cup \{y\}$
11:     $D \leftarrow D \cup \{x, y\}$
12:     $U \leftarrow U \setminus \big(\{x\} \cup L_1(x) \cup \{y\} \cup L_1(y)\big)$
13: end while
14: Make $D$ minimal
15: return $D$
```

Theorem 4 *The set D returned by Alg is a minimal dominating set in G. Moreover, in order to discover D, the algorithm makes $O(\sqrt{d \cdot n})$ queries, where d denotes the size of a minimum dominating set in G.*

Proof. It is clear that the returned set D is a minimal dominating set. It remains to show the bound on the number of queries. Let $\{z_1, \ldots, z_d\} \subseteq V$ be a minimum dominating set in G. We partition the set V into subsets C_i, $i = 1, \ldots, d$: The set $C_i \subseteq V$ contains z_i and all the neighbors of z_i that are not in $\{z_1, \ldots, z_d\}$ and that are not in any of the previous sets C_j, $j < i$.

Let X and Y denote the x-vertices and y-vertices, respectively, produced by the algorithm. Every x-vertex belongs to a single set C_i. Let X_i, $i = 1, \ldots, d$, denote the vertices of X that belong to C_i. We consider the vertices of X_i in the reverse order in which they have been queried by the algorithm. Let k_i denote the size of X_i and let $x^i_1, \ldots, x^i_{k_i}$ denote the reverse order. For each vertex x^i_j we denote by y^i_j the corresponding y-vertex (which was chosen in the same step as x^i_j). Now observe that (i) there are at least ℓ uncovered vertices in X_i (and thus in C_i, too) before querying x^i_ℓ, (i.e., at least the vertices $x^i_1, \ldots, x^i_\ell$); and (ii) at least ℓ uncovered vertices are covered during the iteration of the while loop in which x^i_ℓ and y^i_ℓ are queried (as z_i, a neighbor of x^i_ℓ, has at least ℓ undominated neighbors in C_i at that time, and y^i_ℓ is chosen to maximize the number of newly dominated vertices).

Consequently, we have that all vertices of the graph are covered when $\sum_{i=1}^{d} \sum_{\ell=1}^{k_i} \ell = n$, i.e., when $\sum_{i=1}^{d} k_i(k_i + 1) = 2n$. The algorithm queries at most $|X| + |Y| = 2|X| = 2\sum_{i=1}^{d} k_i$ vertices. We are thus interested in how big the sum $\sum_{i=1}^{d} k_i$ can be. We consider the following maximization problem:

$$\begin{aligned} \max \sum_{i=1}^{d} k_i & \\ \text{s.t.} \sum_{i=1}^{d} k_i(k_i+1) &= 2n \\ k_i \geq 0 \qquad & \forall i = 1, \ldots, d. \end{aligned}$$

An optimal solution of this maximization problem is $k_1, \ldots, k_d = \frac{\sqrt{\frac{8n}{d}+1}-1}{2}$ which is at most $\sqrt{\frac{2n}{d}}$. This implies that $|X| = \sum_{i=1}^{d} k_i \leq d\sqrt{\frac{2n}{d}} = \sqrt{2dn}$. $\square$

Now we construct an example in which it is possible to compute a minimal dominating set of size $d \geq \sqrt{n} - 1/2$ after querying one specific vertex, but any algorithm needs at least d queries before being able to compute a minimal dominating set.

The graph G has the following structure (see Figure 3 for an illustration). The vertices in V are partitioned into three sets $L_0 = \{q\}, L_1 = \{q^*, x_1, \ldots, x_{d-1}\} \cup \{v_1, \ldots, v_{d-1}\}$ and $L_2 = Y_1 \cup \cdots \cup Y_{d-1}$, where all the sets $Y_1, \ldots, Y_{d-1}$ have cardinality d. All vertices but those in L_2 are connected to q. Moreover, for all $i = 1, \ldots, d-1$, vertex x_i is also connected to the vertex v_i and all vertices in Y_i. It is easy to see that both $\{q, x_1, \ldots, x_{d-1}\}$ and $\{q^*, x_1, \ldots, x_{d-1}\}$ are minimum dominating sets of G.

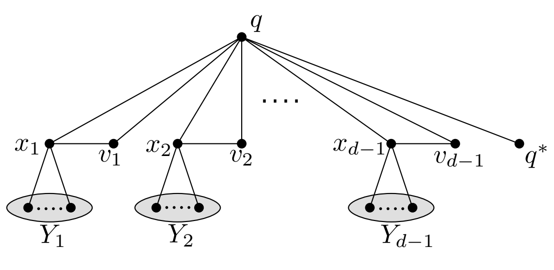

Figure 3: The lower bound construction for a minimal dominating set

First we prove it is enough to query q^* to find a minimal dominating set of G. Indeed, after querying q^*, we discover all edges of G except the ones linking x_i with the vertex v_i. The layers of q^* are $\{q^*\}, \{q\}, L_1 \setminus \{q^*\}, L_2$ (ordered according to the distance from q^*). The query q^* also discovers that q^* is connected to q only, and that, considering only the edges between the layers, vertices of Y_i are adjacent with x_i only. It is now an easy observation that from the information of query q^* the algorithm can infer that $\{q, x_1 \dots, x_{d-1}\}$ is a minimal dominating set in G.

Now let Alg be any deterministic algorithm and let us assume that it has queried any set $Q \subseteq V \setminus \{q^*\}$ with $|Q| < d$ and such that Q contains q (notice that we can always ensure that q is the first vertex queried by the algorithm). We will show that the algorithm cannot guarantee the minimality of any dominating set of G_Q; moreover, it can be proved that the set of vertices that are indistinguishable to the algorithm and that contains q^* has size at least $d - |Q| + 1$. Finally, we prove that there are at least $d - |Q| + 1$ indistinguishable vertices in every Y_i. As a consequence, we can claim that Alg needs at least d queries for discovering a minimal dominating set of G, as we can force the algorithm to make the next query not equal to q^*. Expressing d in terms of n, we obtain a lower bound of $d = \sqrt{n + \frac{1}{4}} - \frac{1}{2} \geq \sqrt{n} - 1/2$.

Let D be any minimal dominating set the algorithm can compute in G_Q, and, without loss of generality, let us assume that the algorithm has not queried any vertex in $\{q^*\} \cup \bigcup_{i=1}^{(d-1)-(|Q|-1)} (\{x_i, v_i\} \cup Y_i)$. Thus, there is at least one index i ($i = 1$) for which there is no query in $\{x_i, v_i\} \cup Y_i$. Observe that if D does not contain $\{x_1, \dots, x_{d-1}\}$, then there is a set Y_j, $j \in \{1, \dots, d-1\}$ which is not dominated by the corresponding vertex x_j. Thus, all vertices of Y_j should be in D in order to be dominated. However, there can be at most $|Q| - 1$ queries within Y_j, and thus there are at least $d - |Q| + 1$ vertices in Y_j which are indistinguishable to the algorithm. Among these indistinguishable vertices ($Y_j \setminus Q$) the algorithm does not know about possible edges, and thus it cannot claim D is a minimal dominating set as if there is such an edge, removing one of its endpoints from D results in a smaller dominating set.

In the case in which D contains $\{x_1, \dots, x_{d-1}\}$, observe first that D cannot contain any vertex from Y_i, $i = 1, \dots, d-1$, otherwise D cannot be a minimal dominating set D. We now argue that there has to be at least one more vertex x in D (not equal to a vertex in Y_i, $i = 1, \dots, d-1$), as $\{x_1, \dots, x_{d-1}\}$ is not a dominating set on its own. At the same time, the algorithm cannot claim the minimality of D: Among the vertices $\{x_1, x_2, \dots, x_{d-1}\}$ there is certainly at least one vertex x_i not in Q. Thus, the algorithm does not know whether $\{q^*, x_i\}$ is an edge or not, and hence cannot know whether x is necessary to dominate all vertices of G.

Theorem 5 *There is a graph for which any algorithm needs to query at least $\sqrt{n} - 1/2$ vertices before it discovers a minimal dominating set, while an optimum offline algorithm needs only one query. Thus no algorithm can achieve a better competitive ratio than $\sqrt{n} - 1/2$ for the problem of discovering a minimal dominating set.*

2.3 Discovering a Maximal Independent Set

In this section we consider the problem of discovering a maximal independent set in G. We construct an example where Opt needs one query, and any algorithm can be forced to make at least $\sqrt{n}$ queries before it discovers any maximal independent set.

Let Alg be any deterministic algorithm. Let us assume that its first query is at node q_1 (out of n nodes $v_1, \dots, v_n$). The graph G has the following structure (see Figure 4 for an illustration). There exists a *central* node c which is connected to every node in V, and forms a maximal independent set on its own. Thus, Opt can make a query at this node and discover that c is a maximal independent set. We add other edges to G to make it impossible for any algorithm to find a maximal independent set with less than $\sqrt{n}$ queries. First, we split the vertices of V into three groups: $L_0 = \{q_1\}$, L_1, and L_2. Vertex q_1 is in L_0, $\sqrt{n}$ vertices are in L_2, and the rest of the vertices is in L_1. The central vertex c is in L_1. Vertex q_1 is connected to every vertex in L_1, and all vertices in L_1 are also connected to L_2, and c is connected to every vertex in L_1 (hence, c is indeed connected to every vertex). There is no edge within vertices in L_2. The query at q_1 splits the vertices into two layers L_1 and L_2. Observe first that $X_1 := \{q_1\} \cup L_2$ is a maximal independent set and there is no other

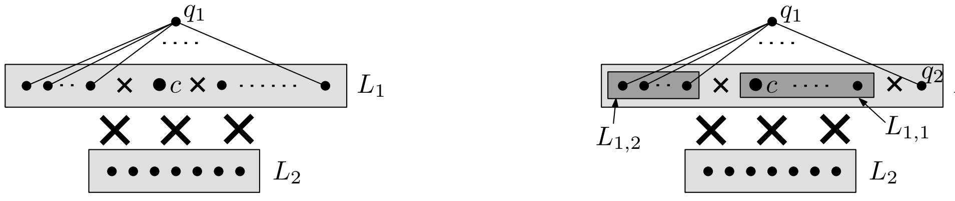

Figure 4: Construction of a graph G for which any algorithm needs $\sqrt{n}$ queries to discover a maximal independent set

one containing a vertex from X_1. Any algorithm discovering X_1 as an independent set needs to query all but one nodes in L_2, which is $\sqrt{n} - 1$ (no query in L_1 can discover any information on non-edges within L_2). Observe that any such query does not discover any information about edges and non-edges within L_1. If Alg does not query only in L_2 (and thus cannot discover X_1 with less than $\sqrt{n}$ queries), let q_2 be the first node that is queried in L_1. Because all the nodes in this layer look the same to the algorithm, the algorithm can be forced to query q_2 at any node of L_2. The edge construction within L_2 is a recursive construction: the query q_2 splits L_1 into two layers: $L_{1,1}$ and $L_{1,2}$, where, $L_{1,2}$ has $\sqrt{n} - 1$ nodes, c is in $L_{1,1}$, q_2 is connected to every node in $L_{1,1}$, and $L_{1,1}$ is connected to every node $L_{1,2}$. There is no edge in $L_{1,2}$. Again, $X_2 := \{q_2\} \cup L_{1,2}$ is the only maximal independent set containing a vertex from X_2, and any algorithm needs $|L_{1,2}| - 1 = \sqrt{n} - 2$ nodes to discover X_2. If Alg queries also in $L_{1,1}$, the nodes within $L_{1,1}$ are split recursively into three parts $\{q_3\}$, $L_{1,1,1}$, and $L_{1,1,2}$, with the obvious size and edge-set. This recursive splitting can obviously run for at least $\sqrt{n}$ times, which shows that no deterministic algorithm can guarantee to find a maximal independent set of a graph with less than $\sqrt{n}$ queries.

Theorem 6 *There is a graph for which any algorithm needs to query at least $\sqrt{n}$ vertices before it discovers a maximal independent set, while an optimum offline algorithm needs only one query. Thus there is no $o(\sqrt{n})$-competitive algorithm for the problem of discovering a maximal independent set.*

2.4 Discovering a Bridge or an Articulation Node of G

In this section we discuss two related properties of G. We want to discover whether the graph G has an articulation node or a bridge. An articulation node of G is a vertex v such that the induced graph on $V \setminus \{v\}$ is not connected. A bridge is an edge e for which the graph $G \setminus e$ is not connected. We show that if the graph contains an articulation node, no algorithm is better than $n/2$-competitive, and if the graph contains a bridge, similarly, no algorithm can achieve a competitive ratio better than $n/2$. We also present an $n/2$-competitive algorithm for the bridge discovery problem.

We begin with the bridge discovery problem. Consider the graph G from Figure 5. G has an even number of vertices, and consists of one node q_0 connected to all remaining $n - 1$ vertices $v_1, \ldots, v_{n-1}$. The vertices v_{2i-1} and v_{2i}, $i = 1, \ldots, (n - 2)/2$, form an edge. The graph contains exactly one bridge – the edge $\{q_0, v_{n-1}\}$. Any algorithm can be forced to make the first query at q_0. Thus, all the remaining vertices lie within the same layer L_1, and look indistinguishable to the algorithm. We can force the next query to be at v_1. This query keeps the vertices $v_3, v_4, \ldots, v_{n-1}$ indistinguishable to the algorithm, and does not give any information on the bridge $\{q_0, v_{n-1}\}$. Hence, next time the algorithm queries a vertex in this group of vertices, we can force it to query v_3. Thus, using the recursive approach, any algorithm can be forced to query at least vertices $v_1, v_3, v_5, \ldots, v_{n-3}$, which then together discover the bridge $\{q_0, v_{n-1}\}$. Observe that an optimum algorithm can query v_{n-1} to discover the bridge. This shows the lower bound $n/2$.

For the problem of discovering an articulation node we prove a lower bound of $n/2$ by modifying the input graph G according to the vertices queried by the algorithm (i.e., we assume that the

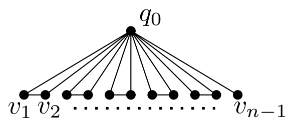

Figure 5: Bridge discovery

adversary is adaptive to the algorithm). The graph G will be a super-graph of a star centered at q such that a node $q^* \neq q$ is incident with q only. In this case, by querying q^* we can claim that q is an articulation node as we discover that q^* has degree 1. Before explaining how the idea behind the proof of the lower bound works, we provide some new definitions. First, given a set of queries Q, we define a *Q-block* as a maximal set of vertices in $V \setminus \{q\}$ that are connected in the graph $G_Q \setminus \{q\}$. Clearly, if $Q = V$, we discover the whole graph, and thus G has an articulation node if and only if there are at least two Q-blocks in the original graph G. The idea of the lower bound is to prevent any algorithm from learning this information soon. In every Q-block B of G_Q we consider a special vertex – an *anchor*. An anchor is a vertex for which the query set Q does not reveal whether the anchor is connected to another anchor in the original graph, i.e., Q is not enough to distinguish G from another $G' \in \texttt{comp}(G,Q)$ (recall that $\texttt{comp}(G,Q)$ is the set of all graphs G' which give the same query results for queries in Q), i.e., we do not know whether all Q-blocks are connected to one another after querying Q, hence we cannot claim that G is (is not) 2-vertex connected. Clearly, in order to claim that G is 2-vertex connected, the algorithm has to prove that $V \setminus \{q\}$ is a Q-block, i.e., all the graphs in $\texttt{comp}(G,Q)$ are 2-vertex connected. Conversely, in order to claim that G is not 2-vertex connected, the algorithm has to prove that all the graphs in $\texttt{comp}(G,Q)$ are not 2-vertex connected.

Now, let us consider any deterministic algorithm. As all vertices are indistinguishable, we may assume that the algorithm starts by querying $Q = \{q_0 = q\}$. Clearly, for each vertex x in $V \setminus \{q\}$, we have that $\{x\}$ is a Q-block whose anchor vertex is x. As all vertices $V \setminus \{q\}$ are indistinguishable, we can assume that the algorithm queries $q_1 \neq q^*, q$. In this case we grow the Q-block $B = \{q_1\}$ by merging it with two other Q-blocks $B' = \{x'\}$ and $B'' = \{x''\}$, with $x', x'' \neq q^*$. Basically, we add the edges $\{q_1, x'\}$ and $\{x', x''\}$ to G. Notice that there are 2-vertex connected graphs in $\texttt{comp}(G, \{q_0, q_1\})$ as we do not know whether there are edges connecting two anchor vertices to each other. Finally we let x'' be the new anchor vertex of the Q'-block B, where $Q' = Q \cup \{q_1\}$. At a generic step, let us assume that the algorithm queried all the vertices in Q, and let us assume that $\texttt{comp}(G,Q)$ contains a 2-vertex connected graph and a graph with an articulation node. The algorithm can either choose to query a vertex q' in the Q-block B we grew so far or not. In the first case, notice that the new information discovered is maximized when q' is exactly the anchor vertex of the Q-block B. In the case where q' is from B and is the anchor vertex a of B, we merge B with two other blocks $B' = \{x'\}$ and $B'' = \{x''\}$, where $x', x'' \neq q^*$ (it is worth noticing that all vertices but q and those in B are indistinguishable in G_Q) by simply adding edges $\{a, x'\}$, and $\{x', x''\}$ to G. Let $Q' = Q \cup \{q'\}$. In the new graph, x'' is the new anchor of the enlarged Q'-block B' containing the old block B. In the case where query q' is outside B, we merge two singleton Q-blocks $\{q'\}$ and $\{x'\}$ to B by adding edges $\{q', x'\}$, and $\{x', a\}$ to G, where a is the anchor vertex of B, and x' is any vertex outside B and not equal to q'. Notice that in this new construction, a remains the anchor of the new Q'-block B' that contains the original Q-block B (where $Q' = Q \cup \{q'\}$). The lower bound of $n/2$ follows from the fact that the algorithm queries at least $(|B| - 1)/2$ vertices of B.

Theorem 7 *For the problem of discovering a bridge or an articulation node there is no better deterministic algorithm than $n/2$-competitive.*

We now present a simple algorithm for determining whether a graph G is 2-edge connected. The algorithm needs at most $\lceil \frac{n}{2} \rceil$ queries. The algorithm makes an arbitrary initial query q_0. The

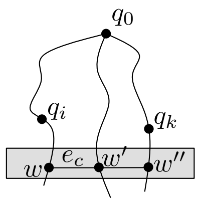

Figure 6: Assigning vertex w' to the query q_i

resulting layered graph $G_{\{q_0\}}$ is used by the algorithm to choose the next queries. We denote by q_i the query that is made by the algorithm in the i-th step, and by Q_i all the queries (including q_i) made so far. Observe that if there is i such that there is only one edge e between L_i and L_{i+1}, the edge e is a bridge of G. Observe also that if G has a bridge $e \in E$, it has to appear as an edge in the result of the query q_0. Thus, choosing query q_{i+1}, we can concentrate on those edges of G_{q_0}, which are not part of any cycle of G_{Q_i}. While there are such edges (and thus candidates for a bridge), the algorithm picks among all such edges the farthest endpoint from q_0, and queries it. We claim that this algorithm terminates, and that the algorithm knows at the end whether the graph has a bridge or not, and that it makes at most $\lfloor (n-1)/2 \rfloor$ queries on top of q_0 (and is thus $\lceil n/2 \rceil$-competitive).

Let q_i be the query of the algorithm in step i, and let $e_i = \{u_i, q_i\}$ be the bridge of $G_{Q_{i-1}}$ with q_i the farthest endpoint from q_0 among all bridges of $G_{Q_{i-1}}$. Let ℓ_i denote the distance of q_i from q_0. Let $R(q_i)$ be the set of vertices from layers L_j, $j \geq \ell_i$, which can be reached from q_i by a path which uses at most one vertex from each L_j, $j \geq \ell_i$. (i.e., if we orient the edges according to the increasing distance from q_0, the set $R(q_i)$ is the set of all vertices for which there exists a directed path from q_i). Thus, $R(q_i)$ forms a component of $G_{Q_{i-1}} \setminus \{e_i\}$, as there cannot be any edge with endpoints in the same layer leaving $R(q_i)$ (otherwise e_i would no longer be a bridge in $G_{Q_{i-1}}$). Let us assume that e_i is not a bridge in G. Then there exists a cycle C in G which contains the edge e_i. The cycle C has to contain a not yet discovered edge $e_c = \{w, w'\}$ which is adjacent to a vertex w in $R(q_i)$, and to a vertex $w' \notin R(q_i)$. The vertices w and w' have to be from the same layer L_j, $j \geq \ell_i$ (as the edge $\{w, w'\}$ was not discovered by q_0). Clearly, q_i discovers this edge $\{w, w'\}$, as the distance from q_i to w is $j - \ell_i$ (as $w \in R(q_i)$), and the distance from q_i to w' is bigger than $j - \ell_i$ (as $w \notin R(q_i)$). As $\{w, w'\}$ is a newly discovered edge, it follows that w' was not queried before. To show that at most $\lfloor (n-1)/2 \rfloor$ queries are made by the algorithm after the query q_0, we want to assign one unqueried vertex to one queried vertex. In our case we assign w' to q_i (notice that w could possibly be equal to q_i, and thus cannot be assigned to q_i). We now show that w' is not already assigned to a previously queried vertex q_k, $k < i$, with $\ell_k \geq \ell_i$. Figure 6 depicts the situation. If this is the case, w' is assigned to query q_k because w' is an endpoint of an edge $\{w', w''\}$ which was discovered by query q_k, and which is a part of a cycle that shows that q_k is not an endpoint of a bridge in G. Thus, $w'' \in R(q_k)$ and $w' \notin R(q_k)$. Clearly, the distance between q_k and w' is $j - \ell_k + 1$. The distance between q_k and w has to be $j - \ell_k + 1$ as well, as the edge $\{w, w'\}$ is not discovered by q_k. But this is not possible. The shortest path from q_k to w cannot go via a vertex from layer L_s, $s < \ell_k$ (the distance would be bigger than $j - \ell_k + 1$). Thus, the shortest path between q_k and w goes only via vertices of layers L_s, $s \geq \ell_k$. But then e_i cannot be a bridge in $G_{Q_{i-1}}$: The shortest path from q_k to w, the shortest path from w to q_i, and the path from q_i to q_k via q_0 induce a cycle with e_i, using edges known after query q_k. This is a contradiction, and thus w' is not assigned to q_k and can be assigned to q_i.

Thus, if e_i is not a bridge, we will discover at least one new edge e_c that includes e_i into a cycle of G, and one of the endpoints of e_c can be assigned to q_i. If we do not discover any such edge, the edge e_i is a bridge of G. The assignment argument shows that after q_0 we query at most $\lfloor (n-1)/2 \rfloor$ vertices. The termination of the algorithm follows from the fact that we can query at most n vertices, and from the fact that if G_{Q_i} contains a bridge, then its endpoint further from q_0

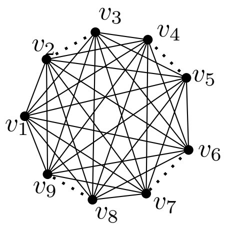

Figure 7: A graph for the lower bound of maximum-degree discovery. The dotted lines depict the missing (deleted) edges in the graph

was not queried yet, and we still have a vertex to query in step $i + 1$.

Theorem 8 *There is an $\lceil n/2 \rceil$-competitive algorithm for the problem of discovering a bridge of a graph.*

2.5 Discovering the Min/Max Degree of G

We investigate how many queries are needed in order to discover the minimum degree of G, and the maximum degree of G. The lower bound construction for the problem of finding an articulation node (Section 2.4) shows an example where any deterministic algorithm needs at least $n/2$ queries to discover the minimum degree of G, whereas an optimum algorithm needs only one query, yielding a lower bound $n/2$ on the competitive ratio of deterministic algorithms. For the problem of discovering the maximum degree we similarly present a lower bound $n/2$ on the competitive ratio of deterministic algorithms. Consider a graph G with $n = 2k + 1$ vertices, which is constructed from a complete graph K_n by deleting the "even" edges $\{v_{2i}, v_{2i+1}\}$, $i = 1, \ldots, k$ from the cycle $v_1, v_2, v_3, \ldots, v_n$. An example of such a graph for $n = 9$ is in Figure 7. Observe that v_1 is the only vertex of the graph which has degree $n - 1$, and thus the maximum degree of G can be discovered by one query at v_1. On the other hand, any other vertex v_i has exactly $n - 2$ neighbors, which are indistinguishable with the query. Thus, every deterministic algorithm can be forced to query k vertices before it can distinguish v_1 from other vertices, and therefore the algorithm makes at least $k + 1$ queries before it reveals the maximum degree of G.

3 Conclusions

We have introduced the online problem of discovering graph properties with all-shortest-paths queries, and considered in more detail the discovery of the diameter, a minimal dominating set, a maximal independent set, the 2-edge connectivity, the 2-vertex connectivity, the maximum degree, and the minimum degree of an unknown graph. We have presented lower bounds for the problems, and also an $O(\sqrt{d \cdot n})$-competitive algorithm for the minimal dominating set discovery, and an optimal $\frac{n}{2}$-competitive algorithm for the bridge discovery problem. We have also introduced a technique of querying an interface of a graph G_Q, which may prove to be helpful in other discovery settings. Furthermore we have shown an adaptive-adversary lower bound construction, which is the first adaptive construction in the discovery setting as introduced in [2].

Our work was motivated by the current intensive activities in the area of mapping the Internet. The all-shortest-path queries model the information that is obtained from routing tables of BGP routers. Of course, our assumption of getting *all* shortest paths is not reflected fully in reality – it certainly is a simplification which helps to analyze the problem. In reality, we would assume to get much less information. The lower bounds presented in this paper suggest, however, that in any realistic situation we cannot hope for better results.

References

[1] A. Barrat, T. Erlebach, M. Mihalák, and A. Vespignani. A (short) survey on network discovery. Technical report, DELIS – Dynamically Evolving, Large-Scale Information Systems, 2008.

[2] Z. Beerliová, F. Eberhard, T. Erlebach, A. Hall, M. Hoffmann, and M. Mihalák. Network discovery and verification. *IEEE Journal on Selected Areas in Communications (JSAC)*, 24(12):2168–2181, December 2006.

[3] G. Chartrand and P. Zhang. The theory and applications of resolvability in graphs: A survey. *Congressus Numerantium*, 160:47–68, 2003.

[4] B. Cheswick. Internet mapping project. http://www.cheswick.com/ches/map/.

[5] R. Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics.* Springer-Verlag, Heidelberg, 3rd edition, July 2005.

[6] DIMES. Mapping the internet. http://www.netdimes.org/.

[7] R. Govindan and H. Tangmunarunkit. Heuristics for Internet map discovery. In *Proceedings of the 19th Conference on Computer Communications (IEEE INFOCOM)*, pages 1371–1380, Tel Aviv, Israel, March 2000.

[8] L. S. Ram. Tree-based graph reconstruction. Research Report of the European Graduate Program Combinatorics-Computation-Geometry (CGC), March 2003.

[9] R. L. Rivest and J. Vuillemin. On recognizing graph properties from adjacency matrices. *Theoretical Computer Science*, 3:371–384, 1976.

[10] D. Ron. Property testing. In P. M. Pardalos, S. Rajasekaran, J. Reif, and J. D. P. Rolim, editors, *Handbook of Randomized Computing*, volume II, pages 597–649. Kluwer Academic Publishers, 2001.

[11] Route Views Project. University of Oregon. http://www.routeviews.org.