Discovery of Network Properties with All-Shortest-Paths Queries

Davide Bilò\textsuperscript{a,1}, Thomas Erlebach\textsuperscript{b}, Matúš Mihalák\textsuperscript{*a}, Peter Widmayer\textsuperscript{a}

\textsuperscript{a}Institute of Theoretical Computer Science, ETH Zurich, Universitätstrasse 6, 8092 Zürich, Switzerland
\textsuperscript{b}Department of Computer Science, University of Leicester, University Road, LE1 7RH, United Kingdom

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 practice, 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.

Key words: Graph property, network, discovery, query, online algorithms, shortest path

\textsuperscript{*}An extended abstract of this work has appeared in Proceedings of the 15th International Colloquium on Structural Information and Communication Complexity (SIROCCO 2008), LNCS 5058, Springer-Verlag, April 2008, pp. 89–103.
\textsuperscript{*}Corresponding author; Tel.: +41 (44) 6320492; Fax: +41 (44) 63 21399

\textsuperscript{1}Present address: ???
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 measurement). All these methods are commonly called traceroute-like measurements (traceroute is a command/tool on UNIX-based systems that is used to trace the route of packets in IP networks such as the Internet). For example, the Oregon Route-Views (RV) project [1] 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 [2] 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. [3, 4, 2, 1].

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 [5]. The map of a network (and the network itself) is modeled as a connected, 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 [5] 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 of vertices $\{u, v\}$ 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
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 $G$. We use standard graph-theoretic terminology and notation, as it is described for example in [6]. The diameter of a graph $G = (V, E)$ is denoted by $\text{diam}_G$. An independent set $I \subseteq V$ is maximal if there is no independent set $J$ with $I \subset J$. It is maximum if there is no independent set $J$ with $|I| < |J|$. Minimal and minimum dominating sets are defined analogously. For the definition of the shell index (also known as the core index) in our context, see for example [7], and for the definition of graphs with power-law node-degree distribution, see for example [8].

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 maximal dominating set, a maximal independent set, minimum degree, maximum degree, edge connectivity and vertex connectivity. 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, respectively, that are discovered by queries $Q$. 

![Figure 1: A graph $G$ (left) and the result of a query at node $v_1$ as a layered graph (right) discovered.](image)
i.e., $E_Q = \bigcup_{q \in Q} E_q$ and $\overline{E}_Q = \bigcup_{q \in Q} \overline{E}_q$. The graph $G_q$ and graph $G_Q$ is the graph on $V$ with the edge set $E_q$ and $E_Q$, respectively. Finally, we denote by $\text{comp}(G, Q)$ the set of all graphs $G'$ with vertex set $V$ containing all the edges in $E_Q$ and all non-edges in $\overline{E}_Q$. In other words, $\text{comp}(G, Q)$ is the set of all graphs with vertex set $V$ for which the queries in $Q$ yield the same results as for $G$.

It is easy to observe that querying all vertices of $G$ discovers all the edges and non-edges of $G$ and thus any (computable) 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 $c \cdot \text{Opt}$, 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 $A$ that 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 $\rho$-approximation algorithm for the problem of discovering a minimization property if for any input graph $G$ it discovers a $\rho$-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 $\rho$-approximation, $c$-competitive algorithm for the diameter discovery problem is an algorithm that discovers a graph $G_Q$ for which the diameter $\text{diam} G_Q$ is at most $\rho \cdot \text{diam} G$, and queries at most $c$ times more queries than an optimal offline $\rho$-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 monotone $n$-vertex graph property must examine $\Omega(n^2)$ entries in the adjacency matrix representing the graph (a property is called monotone if the following implication holds: if $G = (V, E)$ has the property then every graph $G = (V, E')$, where $E \subseteq E'$, also has the property; and a property is called non-trivial if the set of $n$-vertex graphs satisfying it is neither empty nor the set of all $n$-vertex graphs). Each such examination of an entry can be seen as a query (the result of such a query would tell whether there is an edge between the two nodes specified by the 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 [5]. They 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 [11] and [12], respectively.

Among the graph properties that we study in this article, it is the diameter of a graph that gained the most attention. Albeit not considering queries (to oracles) as the main computational element, and thus having quite unrelated character to our work, we refer to [13] for a survey of the topic for further references.

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 \( (1 + \frac{1}{d+1}) \)-approximation algorithm for the discovery of the diameter of \( G \). The “interface” is in our case a layer of vertices that 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 \( (\frac{3}{2} + \frac{2}{\sqrt{d}}) \)-approximation, \( (1 + \frac{n}{2p}) \)-competitive algorithm, where \( \ell \) 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 that 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 the 2-edge and 2-vertex connectivity of \( G \), and show a lower bound \( \lceil n/2 \rceil \) on the competitive ratio of any algorithm for discovering a bridge or an articulation vertex of \( G \). We also present an \( \lceil n/2 \rceil \)-competitive algorithm that 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
### Table 1: Summary of results

<table>
<thead>
<tr>
<th>Property</th>
<th>Upper Bound</th>
<th>Lower Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diameter</td>
<td>n-1</td>
<td>n-1</td>
</tr>
<tr>
<td>((1 + \frac{1}{k+1}))-approx. of (\text{diam}_G \in \omega(1))</td>
<td>(k) layers (worst case (\Omega(n)))</td>
<td>–</td>
</tr>
<tr>
<td>((\frac{1}{2} + \frac{n}{2p+1}))-approx. of (\text{diam}_G \in \omega(1))</td>
<td>(1 + \frac{n}{2p+1})</td>
<td>–</td>
</tr>
<tr>
<td>Minimal Dom. Set</td>
<td>(O(\sqrt{d \cdot n}))</td>
<td>(\sqrt{n} - 1/2)</td>
</tr>
<tr>
<td>Maximal Ind. Set</td>
<td>–</td>
<td>(\sqrt{n})</td>
</tr>
<tr>
<td>2-edge connectivity</td>
<td>(\left\lceil \frac{2}{3} \right\rceil)</td>
<td>(\left\lceil \frac{2}{3} \right\rceil)</td>
</tr>
<tr>
<td>2-vertex connectivity</td>
<td>–</td>
<td>(\left\lceil \frac{2}{3} \right\rceil)</td>
</tr>
<tr>
<td>Minimum degree</td>
<td>–</td>
<td>(\frac{n}{2})</td>
</tr>
<tr>
<td>Maximum degree</td>
<td>–</td>
<td>(\frac{n}{2})</td>
</tr>
</tbody>
</table>

ratios of any algorithm.

We note that our lower-bound proofs for competitive ratios all share the property that for the constructed graphs an optimum offline algorithm needs only one query to discover the sought property. Therefore, by showing that any deterministic online algorithm needs to make at least \(k\) queries (i.e., has query complexity \(k\) for the constructed instance), we also obtain a lower bound of \(k\) on the competitive ratio.

Table 1 summarizes the results of this article.

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 \(G_Q = (V, E_Q)\) 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 that is a good approximation of the diameter of \(G\). We first prove that while a single query is sufficient to approximate the diameter within a factor of 2, any algorithm with approximation ratio better than 2 has competitive ratio \(\Omega(n)\), where \(n\) is the number of vertices of \(G\). The lower bound holds on graphs of diameter at most 2. Then, with graphs of larger diameter in mind, we show how approximation ratios better than 2 can be achieved by querying all vertices in one or several of the
layers determined by an initial query, although this approach does not directly yield any non-trivial bound on the competitive ratio. Finally, by allowing the algorithm to choose the layer to query by selecting a layer having fewest vertices within a certain range of layers, we obtain a trade-off between approximation ratio and competitive ratio for families of graphs whose diameter is not very small.

It has been previously observed (see e.g. [14, 15]) 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 a vertex with the maximum distance from \( q \). Let \( \ell \) denote this distance, i.e., \( d(q, v) = \ell \). Clearly, \( \text{diam}_G \geq \ell \). Also, for any two nodes \( u, v \in V \), \( d(u, v) \leq d(u, q) + d(q, v) \leq 2\ell \). Thus, the diameter of \( G_q \) is at most \( 2\ell \), 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 ratio than 2. For any \( n \geq 3 \), 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 \( q \), except \( u \) or \( v \), the result looks 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, this example shows that any deterministic algorithm needs \( n - 1 \) queries to distinguish \( G_1 \) from \( G_2 \). Hence, there is no deterministic \((2 - \epsilon)\)-approximation algorithm with less than \( n - 1 \) queries. Note also that an optimal offline algorithm can discover the exact diameter of \( G_2 \) with just one query (a query at \( u \) or \( v \) yields a complete bipartite graph with \( u \) and \( v \) on one side and the remaining vertices on the other side, showing that the diameter of the graph must be two). This implies that any deterministic \((2 - \epsilon)\)-approximation algorithm has competitive ratio at least \( n - 1 \).

If the diameter of the graph is larger than two (e.g. a growing function of \( n \), 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 \( i \) from \( q \). As a next step we query all vertices at layer \( L_k \) (we will show that \( k \approx \frac{3}{4} \ell \) 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 two 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 \( L_k \). Thus we are left with two cases, first, \( u \) and \( v \) are both from layers between \( q \) and \( L_k \), and second,
Figure 2: The initial query $q$ splits the vertices of $G$ into $\ell$ layers $L_1, L_2, \ldots, L_\ell$. The distance $d(u, v)$ between any two nodes $u, v \in V$ is at most $d(u, q) + d(q, v) \leq 2\ell$, but can be shorter if edges within the same layer are present.

$u$ and $v$ are both from layers between $L_k$ and $L_\ell$. Knowing information about distances from every vertex $w \in L_k$ (we have queried the whole layer $L_k$) allows us to further improve the bound on the distance between $u$ and $v$ with respect to $\ell$ (remember that $\ell$ is the lower bound of the diameter of $G$).

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

**Case 2.** If both $u$ and $v$ lie within layers $L_{k+1}, \ldots, L_\ell$, and every shortest path in $G$ between $u$ and $v$ goes via vertices of layers $L_{k+1}, \ldots, L_\ell$, we will use the layer $L_k$ in a similar way as $q$ was used in the previous case. Trivially, $d_G(u, v) \leq d_G(u, q') + d_G(q', v) = d_G(u, q') + d_G(q', v)$, for any $q' \in L_k \cup \{q\}$. (1)

As every shortest path between $u$ and $v$ lies “below” $L_k$, we will also improve the lower bound on the distance between $u$ and $v$. 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 $L_k$ that is closest to $s$. We obtain

$$d_G(q', u) \leq d_G(q', s) + d_G(s, u) \leq (\ell - k) + d_G(s, u),$$

and similarly

$$d_G(q', v) \leq d_G(q', s) + d_G(s, v) \leq (\ell - k) + d_G(s, v).$$

Thus,

$$d_G(q', u) + d_G(q', v) \leq 2(\ell - k) + d_G(s, u) + d_G(s, v) = 2(\ell - k) + d_G(u, v).$$

As $d_G(q', u) = d_G(q', u)$ and $d_G(q', v) = d_G(q', v)$, we obtain

$$d_G(q', u) + d_G(q', v) - 2(\ell - k) \leq d_G(u, v),$$

(2)

and the approximation ratio (for the diameter $d_G(u, v)$) obtained for this type of vertices is at most (putting together Inequality (1), Inequality (2), and the
trivial bound $\ell \leq d_G(u, v)$

\[
\frac{d_{G'}(q', u) + d_{G'}(q', v)}{\max\{\ell, d_{G'}(q', u) + d_{G'}(q', v) - 2(\ell - k)\}}
\]  (3)

We now distinguish two subcases based on the value of the denominator in (3). First, assume $\ell \geq d_{G'}(q', u) + d_{G'}(q', v) - 2(\ell - k)$, i.e., $\ell + 2(\ell - k) \geq d_{G'}(q', u) + d_{G'}(q', v)$. Then clearly, the approximation ratio is at most $\frac{\ell + 2(\ell - k)}{\ell} = \frac{3\ell - 2k}{\ell}$. Second, if $\ell \leq d_{G'}(q', u) + d_{G'}(q', v) - 2(\ell - k)$, then the upper bound on the approximation ratio is of the form $\frac{\ell}{2(\ell - k)}$, which is maximized (under the condition that $\ell \leq x - 2(\ell - k)$) for $x = \ell + 2(\ell - k)$. Putting in the value of $x$ into our bound, we get that the approximation ratio is at most $\frac{3\ell - 2k}{\ell}$ as well.

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 $\text{diam}_{G_Q}/\text{diam}_G \leq \frac{3}{2} - \frac{1}{4}$. This proves Theorem 1.

We have assumed, for simplicity of presentation, that every fractional computation results in an integral number (such as the query level $k = \frac{3\ell + 2}{4}$). To make the computations precise, one has to round the numbers, which can “shift” the queried layer by half, i.e., $[|k| - k] \leq 0.5$ (by $|k|$ we denote the rounding of $k$ to the nearest integer). This results in a small additive error of order $\frac{1}{4}$ in the approximation ratio of the diameter. Observe that this error approaches zero, as $\ell$ (and the diameter) grows with $n$. For instance, we obtain that $\text{diam}_{G_Q}/\text{diam}_G \leq \frac{2(|k| - 1)}{\ell} = \frac{2(\lfloor|k| - 1\rfloor)}{\ell} \leq \frac{2(\lfloor|k| - 1\rfloor)}{\ell} = \frac{3}{2}$. Therefore $\text{diam}_{G_Q} \leq \frac{3}{2}\text{diam}_G$ (compare with the original bound $\text{diam}_{G_Q} \leq \frac{5}{4}\text{diam}_G$).

**Theorem 1.** Let $G$ be any graph. A query set $Q = \{q\}$ results in a graph $G_Q = (V, E_Q)$ such that $\text{diam}_{G_Q} \leq 2\cdot \text{diam}_G$. Let $\ell$ be the maximum distance from $q$ to a vertex of $G$. Setting $Q = \{q\} \cup L_{\alpha}(q)$, $\alpha < 1$, the approximation ratio $\rho$ of the algorithm that computes $\text{diam}_{G_Q}$ as the approximation of $\text{diam}_G$ is $\max\{2\alpha - \frac{3}{4}, 3 - 2\alpha\}$. For $\alpha = \frac{1}{4} + \frac{1}{\ell}$, we get approximation ratio $\frac{3}{4} - \frac{1}{\ell}$ (and approximation ratio $\frac{3}{2}$ if rounding effects are taken into account).

It is natural to ask whether 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 $q$, lower 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 $\frac{\ell + 2(\ell - k)}{\ell}$, if they are from $P_2$ we get a bound $\frac{3\ell - 2k}{\ell}$, and if they are from $P_3$ we get a bound $\frac{3\ell - 2k}{\ell}$ on the approximation ratio. The first two bounds can be
obtained analogously as in the situation 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 $k = \frac{5}{6} \ell + \frac{1}{3}$, $s = \frac{5}{6} \ell + \frac{4}{3}$, the graph $G_Q$ has diameter $diam_{G_Q} \leq \left(\frac{5}{3} - \frac{2}{3}\right)diam_G$. For the rounding, we set $k = \frac{5}{6} \ell + \frac{1}{2}$ and $s = \frac{5}{6} \ell + \frac{1}{2}$. Then the graph $G_Q$ has diameter $diam_{G_Q} \leq \frac{4}{3}diam_G$ (rounding $k$ and $s$, we obtain, for example, $\ell/2(\ell + 1) - \frac{2}{3} \leq \frac{\ell+2}{3(\ell+1)}(\ell+2) - \frac{2}{3} = \frac{\ell+2}{3(\ell+1)}(\ell+2) + 0.5 = \frac{\ell+2}{3(\ell+1)}(\ell+2) - 0.5 \leq \frac{\ell+2}{3(\ell+1)}(\ell+2) - 0.5 \leq \frac{4}{3}$.)

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 $L_{j/2}$ and $L_{\ell}$ are uniformly spaced. It is then an easy adaptation of previous considerations to show that such a choice of $s$ queries leads to a $(1 + \frac{1}{s+1})$-approximation.

**Theorem 2.** Let $\ell$ be the maximum distance from an initial query $q$ to a vertex of $G$. Let $Q = \{q\} \cup L_{k_1} \cup L_{k_2} \cup \ldots \cup L_{k_s}$, $s \geq 1$, $k_i < k_{i+1}$, $i = 1, \ldots, s - 1$, where $k_i = \ell/2 + i \cdot \frac{\ell}{2(s+1)} + 0.5$. 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 $G$.

**Proof.** The queried layers $L_{k_1}, L_{k_2}, \ldots, L_{k_s}$ split the unqueried vertices into disjoint parts: vertices between $q$ and $L_{k_1}$, vertices between $L_{k_i}$ and $L_{k_{i+1}}$, for $1 \leq i \leq s - 1$, and vertices below $L_{k_s}$. As $G_Q$ is a subgraph of $G$, we have $diam_G \geq diam_{G_Q}$. Let $u$ and $v$ be the vertices that form the diameter of $G_Q$. Clearly, if $u$ or $v$ is from $Q$, then $diam_{G_Q} = diam_G$. Similarly, if $u$ and $v$ are from different parts, then $d_{G_Q}(u, v) = d_G(u, v)$, and thus again, $diam_{G_Q} = diam_G$. Thus, the interesting cases are when $u$ and $v$ are from the same part. Similarly to the case with two layers, we obtain that $diam_{G_Q}/diam_G$ is at most $\frac{2(\ell+1)}{\ell}$. If $u$ and $v$ are from the first part: at most $\frac{\ell+2(k_{i+1}-k_i-1)}{\ell}$ if $u$ and $v$ are between layer $L_{k_i}$ and layer $L_{k_{i+1}}$, $i = 1, \ldots, s - 1$; at most $\frac{\ell+2k}{\ell}$ if $u$ and $v$ are below layer $L_{k_s}$. Straightforward calculations show that, taking rounding effects into account, the approximation ratio $diam_{G_Q}/diam_G$ is in all cases at most $(1 + \frac{1}{s+1})$. □

So far we have been mainly concerned with the quality of the approximation, but we did not consider the number of queries we make. In particular, a problem of the 3/2-approximation algorithm from Theorem 1 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 $k = \lceil \frac{4}{3} \ell + 0.5 \rceil$, but parametrize the choice of $k$ and allow $k$ to be in the range $\{ \lceil \frac{1}{2} \ell + 0.5 \rceil - p, \ldots, \lceil \frac{1}{2} \ell + 0.5 \rceil + p \}$, where $p$ is a parameter. The algorithm now picks the layer $L_i$ with the minimum number of vertices among all layers $L_i$, $i \in \{ \lceil \frac{1}{2} \ell + 0.5 \rceil - p, \ldots, \lceil \frac{1}{2} \ell + 0.5 \rceil + p \}$. Thus, the size of $L_k$ is at most $n/(2p+1)$, which is also the upper bound on the competitive ratio of the algorithm. Relaxing $p$ allows us 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 $\{0, 1, 2, \ldots, \lfloor \frac{1}{4} \ell - 0.5 \rfloor \}$.

Repeating the case analysis leading to Theorem 1, the upper bounds on the approximation ratio for the different cases are $1$, $2(k-1)/\ell$, and $2\ell-2k$. As $3\ell - 2k \leq 3\ell - 2(\lceil \frac{1}{2} \ell + 0.5 \rceil - p) \leq 3\ell - 2(\lceil \frac{1}{2} \ell + 0.5 \rceil - p) = \frac{4}{3} \ell + 2p$ and $2(k-1) \leq 2(\lfloor \frac{1}{2} \ell + 0.5 \rfloor + p - 1) \leq 2(\lfloor \frac{1}{2} \ell + 1 \rfloor + p - 1) = \frac{2}{3} \ell + 2p$ we obtain that the approximation ratio is upper bounded by $\frac{3}{2} + \frac{2p}{\ell}$ and $1 + \frac{n}{2p+1}$, respectively.

**Theorem 3.** Let $G$ be any graph and $q$ a query that results in $\ell$ 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 $(3/2 + \frac{2p}{\ell})$-approximation of the diameter of $G$ and is $1+n/(2p+1)$-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 $\text{Alg}$, 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, $\text{Alg}$ 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 then vertex $y$ is chosen among the set of neighbors of $x$ such that $y$ maximizes the set of newly dominated nodes by $y$ (i.e., the subset of neighbors $N(y)$ of $y$ that are at distance 2 from $x$ and that 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.

**Theorem 4.** The set $D$ returned by $\text{Alg}$ 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 $G$. 

11
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 undominated 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: $U \leftarrow U \setminus \{(x) \cup L_1(x) \cup \{y\} \cup L_1(y)\}$
13: **end while**
14: Make $D$ minimal
15: return $D$

**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 $x_{i1}, \ldots, x_{ik_i}$ denote the reverse order of $x$-vertices in $X_i$. For each vertex $x_{ij}$ we denote by $y_{ij}$ the corresponding $y$-vertex (which was chosen in the same step as $x_{ij}$). Now observe that (i) there are $\ell$ undominated vertices in $X_i$ before querying $x_{ij}$ (the vertices $x_{i1}, \ldots, x_{ij}$) and thus there are at least $\ell$ undominated vertices in $C_i$ (i.e., at least the vertices $x_{i1}, \ldots, x_{ij}$); and (ii) at least $\ell$ undominated vertices are dominated during the iteration of the while loop in which $x_{ij}$ and $y_{ij}$ are queried (as $z_i$, a neighbor of $x_{ij}$, has at least $\ell$ undominated neighbors in $C_i$ at that time, and $y_{ij}$ is chosen to maximize the number of newly dominated vertices).

Consequently, we have that all vertices of the graph are dominated when $\sum_{i=1}^{d} k_i = n$, 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 large the sum $\sum_{i=1}^{d} k_i$ can be. We consider the following maximization problem:
\[
\max \sum_{i=1}^{d} k_i \\
\text{s.t.} \sum_{i=1}^{d} k_i(k_i + 1) = 2n \\
k_i \geq 0 \ \forall i = 1, \ldots, d.
\]

An optimal solution of this maximization problem is \(k_1, \ldots, k_d = \sqrt{\frac{2n}{d}}\). This implies that \(|X| = \sum_{i=1}^{d} k_i \leq d\sqrt{\frac{2n}{d}} = \sqrt{2dn}\). □

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.

For any given \(d\), let \(n = (d - 1)(d + 2) + 2\) and consider a family of graphs on \(n\) vertices defined as follows. The family contains graph \(G\) with the following structure (see Figure 3 for an illustration). The vertices in \(V\) are partitioned into three sets \(L_0 = \{q\}, L_1 = \{q^*, x_1, \ldots, x_{d-1}\} \cup \{v_1, \ldots, v_d\}\) 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 \(G\).

Consider a query at vertex \(q\). The family of graphs that we consider is then exactly \(\text{comp}(G, \{q\})\) (recall that \(\text{comp}(G, Q)\) is the set of all graphs on vertex set \(V(G)\) for which the query set \(Q\) delivers the same information as for graph \(G\)). The idea of the lower bound is to show that no algorithm can discover a minimal dominating set of \(G\) with less than \(d\) queries. More precisely, we will show that any algorithm can be forced to query in such a way that with less than \(d\) queries \(Q\), for any set \(D \subseteq V\) that is a minimal dominating set in \(G_Q\), there is a graph in \(\text{comp}(G, Q) \subseteq \text{comp}(G, \{q\})\) for which \(D\) is not 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
\[ x_i \text{ with the vertex } v_i. \] 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 \( \{q, x_1, \ldots, x_d-1\} \) is a minimal dominating set in \( G \).

Now let \( \text{Alg} \) 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 \( G_Q \) for all graphs in \( \text{comp}(G, Q) \); 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 \( \text{Alg} \) 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 \( d = \sqrt{n} + \frac{1}{2} - \frac{1}{2} \geq \sqrt{n} - 1/2 \).

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 \( i \) (e.g., \( i = 1 \)) for which there is no query in \( \{x_i, v_i\} \cup Y_i \). Observe that if \( D \) does not contain \( \{x_1, \ldots, x_{d-1}\} \), then there is a set \( Y_j \), \( j \in \{1, \ldots, d-1\} \) that is not dominated by the corresponding vertex \( x_j \). Thus, all vertices of \( Y_j \) should be in \( D \) in order to be dominated in \( G \). However, there can be at most \(|Q|-1 \) queries within \( Y_j \), and thus there are at least \( d - |Q| + 1 \) vertices in \( Y_j \) that 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 \( e \) (defining graph \( G' \), \( D \) is no longer minimal for \( G' \) (removing one of the endpoints of \( e \) from \( D \) results in a smaller dominating set), while it is minimal for \( G \).

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

**Theorem 5.** There are graphs 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.
Figure 4: Construction of a graph $G$ for which any algorithm needs $\sqrt{n}$ queries to discover a maximal independent 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 an optimal offline algorithm Opt needs one query, and any online algorithm can be forced to make at least $\sqrt{n}$ queries before it discovers any maximal independent set.

For every $n$ we consider a family of graphs on $n$ vertices. The family contains graph $G$ which has the following structure (see Figure 4 for an illustration). $G$ has a central node $c$ that 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 every vertex in $L_2$, 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 $L_2$. Note that a query at $q_1$ splits the vertices into two layers $L_1$ and $L_2$. The edge construction within $L_1$ is a recursive construction: there is a vertex $q_2$ that, when queried, 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 in $L_{1,2}$. There is no edge in $L_{1,2}$. We proceed recursively with the nodes within $L_{1,1}$. We split $L_{1,1}$ into three parts $\{q_3\}$, $L_{1,1,1}$, and $L_{1,1,2}$, with the obvious choice of size and edge-set. This recursive splitting can be repeated at least $\sqrt{n}$ times. Consider a query at vertex $q_1$. Then $\text{comp}(G, \{q_1\})$ is the family of graphs that we consider.

Let Alg be an algorithm aiming to discover a maximal independent set. We make the algorithm query $q_1$ as the first vertex. Thus, the algorithm discovers edges and non-edges between $q_1$, $L_1$ and $L_2$. Observe first that $X_1 := \{q_1\} \cup L_2$ is a maximal independent set in $G$ and there is no other one containing a vertex
from $X_1$. If the algorithm wants to discover $X_1$ as an independent set (to distinguish from graphs in $\text{comp}(G, Q)$ that contain an edge within $L_2$), it needs to query all but one nodes in $L_2$, which amounts to $\sqrt{n} - 1$ queries in addition to the initial query at $q_1$ (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 $\text{Alg}$ does not query only in $L_2$ (and thus cannot discover $X_1$ with less than $\sqrt{n}$ queries), we make the algorithm query $q_2$ as the first node in $L_1$ (remember that all nodes in $L_1$ are indistinguishable to $\text{Alg}$). This reveals new layers within $L_1$: the vertex $q_2$, the layer $L_{1,1}$, and the layer $L_{1,2}$. 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 $|L_{1,2}| - 1 = \sqrt{n} - 2$ queries to discover $X_2$. If $\text{Alg}$ queries also in $L_{1,1}$, we make the first query made by the algorithm in $L_{1,1}$ be $q_3$, etc. This recursive argument shows that no deterministic algorithm can guarantee to find a maximal independent set of the constructed graph with less than $\sqrt{n}$ queries.

**Theorem 6.** For arbitrarily large $n$, 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 $\alpha(\sqrt{n})$-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 $\lceil n/2 \rceil$-competitive, and if the graph contains a bridge, similarly, no algorithm can achieve a competitive ratio better than $\lceil n/2 \rceil$. We also present an $\lceil n/2 \rceil$-competitive algorithm for the bridge discovery problem.

We begin with the bridge discovery problem. We consider a family of graphs which contains the graph $G$ from Figure 5. $G$ has an even number of vertices, and consists of one node $v_0$ connected to all remaining $n-1$ vertices $v_1, \ldots, v_{n-1}$. Each pair of vertices $v_{2i-1}$ and $v_{2i}$, $i = 1, \ldots, (n-2)/2$, forms an edge. The graph contains exactly one bridge – the edge $\{v_0, v_{n-1}\}$. Any algorithm can be forced to make the first query at $v_0$. The considered family of graphs is then $\text{comp}(G, \{v_0\})$. After query $v_0$, 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 $\{v_0, v_{n-1}\}$. Hence, the next time the algorithm queries a vertex in this group of vertices, we can force it to query $v_3$. Continuing in this way, the adversary can force any algorithm to query at least the vertices $v_1, v_3, v_5, \ldots, v_{n-3}$, which then together discover the bridge $\{v_0, v_{n-1}\}$. If fewer queries are made, there is still a graph in $\text{comp}(G, Q)$ that contains no bridge. Observe that an optimum algorithm can
query \(v_{n-1}\) to discover the bridge in \(G\). This shows the lower bound of \(n/2\) on the competitive ratio when \(n\) is even. For odd number of vertices, we can modify the graph \(G\) from Figure 5 and add to the graph another bridge – an edge \(\{q_0, v_n\}\) (thus having in total \(n + 1\) vertices – an odd number). The same arguments show that the algorithm needs to, after querying \(q_0, v_1, v_3, \ldots, v_{n-3}\), additionally query a vertex in \(\{v_{n-1}, v_{n-2}\}\), thus needing in total at least \(1 + \lceil (n + 1)/2 \rceil = \lceil (n + 1)/2 \rceil\) vertices.

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., the graph constructed by the adversary depends on the queries made by the algorithm). The graph \(G\) will be constructed from a star centered at a node \(q\) by adding edges where “needed.” We will make sure that there is a node \(q^* \neq q\) such that \(q^*\) is incident (in \(G\)) with \(q\) only. In this case, by querying \(q^*\) we can assert 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 \(G_{Q \setminus \{q\}}\). Observe that initially, for \(Q = \{q\}\), every vertex other than \(q\) forms a \(Q\)-block. 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 for \(Q = V\). The idea of the lower bound is to prevent any algorithm from detecting this quickly.

In every \(Q\)-block \(B\) of \(G_{Q \setminus \{q\}}\) we consider a special vertex – an anchor. An anchor \(a\) of \(Q\)-block \(B\) is a vertex from \(B\) for which the query set \(Q\) does not reveal whether \(a\) is connected to a vertex of any other \(Q\)-block in the original graph.

As long as there are at least two \(Q\)-blocks, by the presence of anchors it follows that \(Q\) is not enough to distinguish \(G\) from another \(G' \in \text{comp}(G, Q)\) that is 2-vertex connected (recall that \(\text{comp}(G, Q)\) is the set of all graphs \(G'\) that give the same query results as \(G\) for queries in \(Q\)). In other words, after querying \(Q\) we do not yet know whether all \(Q\)-blocks are connected to one another, and hence we cannot claim that \(G\) is (or 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 \(\text{comp}(G, Q)\) 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 \(\text{comp}(G, Q)\) 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 \(q_0\), so that we have \(Q = \{q_0\}\) initially. For each vertex \(x\) in \(V \setminus \{q_0\}\), we have that \(\{x\}\) is a \(Q\)-block whose anchor vertex is \(x\). As all vertices in \(V \setminus \{q_0\}\) are indistinguishable,
able, we can assume that the algorithm queries some vertex \( q_1 \notin \{q^*, q_0\} \) next. Then we grow the \( Q \)-block \( B = \{q_1\} \) by merging it with two other \( Q \)-blocks \( B' = \{x'\} \) and \( B'' = \{x''\} \), with \( x', x'' \neq q^* \). For this, we add the edges \( \{q_1, x'\} \) and \( \{x', x''\} \) to \( G \) and let \( x'' \) be the new anchor vertex of the enlarged \( Q \)-block \( B = \{q_1, x', x''\} \), where the set \( Q \) is now equal to \( \{q_0, q_1\} \). Notice that there are 2-vertex connected graphs in \( \text{comp}(G, \{q_0, q_1\}) \), as we do not yet know whether there are edges connecting two anchor vertices to each other. In our construction, we always grow the \( Q \)-block \( B \) while the \( Q \)-blocks disjoint from \( B \) remain singletons.

At a generic step, let us assume that the algorithm has queried all the vertices in some set \( Q \) and that \( \text{comp}(G, Q) \) contains at least one 2-vertex connected graph and at least one graph with an articulation node. The algorithm can either choose to query a vertex \( q' \) in the \( Q \)-block \( B \) that we have grown so far or not. In the first case, notice that the new information discovered is maximized when \( q' \) is the anchor vertex of the \( Q \)-block \( B \), and thus we can assume that the query vertex \( q' \) is equal to the anchor vertex \( a \) of \( B \). In this case we merge \( B \) with two other \( Q \)-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 \( G \) and letting \( x'' \) be the anchor of the enlarged \( Q' \)-block \( B' \) that contains the old \( Q \)-block \( B \), where \( Q' = Q \cup \{q'\} \). In the case where query \( q' \) is made outside \( B \), we merge the 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 \( B \) that is not equal to \( q', q^* \). In this case, \( a \) remains the anchor of the new \( Q' \)-block \( B' \) that contains the original \( Q \)-block \( B \) (where \( Q' = Q \cup \{q'\} \)).

Observe that after \( k \) generic steps, when \( k + 1 \) queries have been made (including the initial query \( q_0 \)), block \( B \) has size at most \( 2k + 1 \) (it can be smaller if the algorithm queries a vertex from \( B \) that is not an anchor). At the end, depending on whether \( n \) is odd or even and whether the algorithm queries only anchors or not, just before the algorithm discovers that there is \( q^* \) (a node of degree 1), there is, besides \( B \), either only one singleton \( Q \)-block – vertex \( q^* \), or two singletons – vertex \( q^* \) and another vertex, let us call it \( w \). In the latter case, we let \( w \) be connected to the anchor of \( B \) in \( G \). In both cases, a query at the anchor of \( B \), or a query at \( q^* \), or a query at \( w \) discovers \( q^* \). In both cases, the set \( B \), before the final query, has size at least \( n - 3 \) and the algorithm makes at least \( |n/2| \) queries (the initial query \( q \), at least \( \lceil (|B| - 1)/2 \rceil \) \( = \lceil (n - 4)/2 \rceil \) \( = \lceil n/2 \rceil \) - 2 queries in \( B \) before the final query, and the final query).

**Theorem 7.** For the problem of discovering a bridge, and for the problem of discovering an articulation node, there cannot be a deterministic algorithm with competitive ratio better than \( \lceil n/2 \rceil \).

We now present a simple algorithm for determining whether a graph \( G \) is 2-edge connected. The algorithm needs at most \( \lceil n/2 \rceil \) queries. The algorithm makes an arbitrary initial query \( q_0 \). The 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 $j$ such that there is only one edge $e$ between $L_j$ and $L_{j+1}$, 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, when we choose query $q_{i+1}$, we can concentrate on those edges of $G_{q_i}$ that 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 $q_0$ (breaking ties arbitrarily), and queries it. We claim that this algorithm terminates, that the algorithm knows at the end whether the graph has a bridge or not, and that it makes at most $\lceil (n-1)/2 \rceil$ queries in addition to $q_0$ (and is thus $[n/2]$-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 $\ell_i$ denote the distance of $q_i$ from $q_0$ (both in $G$ and $G_{Q_{i-1}}$). Let $R(q_i)$ be the set of vertices from layers $L_j$, $j \geq \ell_i$, that can be reached from $q_i$ by a path that uses at most one vertex from each $L_j$, $j \geq \ell_i$ (i.e., if we orient the edges according to 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 $G$ that contains the edge $e_i$. The cycle $C$ has to contain a not yet discovered edge $e_c = \{w, w'\}$ that 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 $\lceil (n-1)/2 \rceil$ queries are made by the algorithm after the query $q_0$, we want to assign one unqueried vertex to each 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 $\{w'', w''\}$ that was discovered by query $q_k$, and that is a part of a cycle that shows that $q_k$ is not an endpoint of

![Figure 6: Assigning vertex $w'$ to the query $q_i$](image-url)
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 $q_i$ 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 $G$ is 2-edge connected and $G_{Q_i}$ contains a bridge, then its endpoint further from $q_0$ 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 Minimum or Maximum 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 bridge-discovery problem (Section 2.4) presents a family of graphs containing graph $G$ from Figure 5 where any deterministic algorithm needs at least $n/2$ queries to discover the only vertex of degree one in $G$. This shows that any deterministic algorithm needs at least $n/2$ queries to discover the minimum degree of $G$, whereas an optimum algorithm needs only one query. This yields 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 family of graphs containing the graph $G$ with $n = 2k$ vertices, which is constructed from a complete graph $K_n$ by deleting the $k - 1$ “even” edges $\{v_{2i}, v_{2i+1}\}, i = 1, \ldots, k - 1$, from the cycle $v_1, v_2, v_3, \ldots, v_n$. An example of such a graph for $n = 8$ is shown in Figure 7. The considered family of graphs is $\text{comp}(G, \{v_2\})$. Observe that $v_1$ and $v_n$ have degree $n-1$, and thus the maximum degree of $G$ can be discovered by one query at $v_1$ or $v_n$. On the other hand, any other vertex $v_i$ has only $n - 2$ neighbors, and they are indistinguishable by the query at $v_1$ (i.e., the query cannot distinguish two graphs which differ in an edge among two of the neighbors). Thus, every deterministic algorithm can be forced to make its first $k - 1$ queries at endpoints of non-edges (and not at $v_1$ or $v_n$). At this time it is still unknown to the algorithm whether the edge
{v_1, v_n} is present in the graph. Therefore, the algorithm must make at least one additional query before it can assert that the maximum degree is n – 1.

**Theorem 9.** For the problem of discovering the maximum degree, and for the problem of discovering the minimum degree, there cannot be a deterministic algorithm with competitive ratio better than $n/2$.

### 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 discovery of a minimal dominating set, and an optimal $\lceil n/2 \rceil$-competitive algorithm for the bridge discovery problem. We have also introduced a technique of querying an interface of a graph $G_Q$ and employed it to design algorithms for approximate diameter discovery. This technique may prove to be helpful in other discovery settings. Furthermore we have shown an adversarial lower bound construction where the graph depends on the queries made by the algorithm. This is the first such construction in the discovery setting as introduced in [5].

This paper does not completely resolve the problems considered. For example, for the problem of discovering a maximal independent set, finding an algorithm with competitive ratio close to the presented lower bound is certainly a challenging problem. Similarly, one could search for an $\lceil n/2 \rceil$-competitive algorithm for the problem of discovering an articulation node. Another interesting goal is to discover the maximum degree or the minimum degree only approximately – then our lower bounds no longer hold. An immediate consideration for future research are other graph-theoretic properties of communication networks, and studying the property-discovery setting with different query models. We note that we have only considered deterministic algorithms. Certainly, using randomization might help, and studying the effect of randomization is an inviting topic.
Our work was motivated by the current intensive activities in the area of mapping the Internet. The all-shortest-paths 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 that 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.

Acknowledgments

This work was 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.

Part of this work was done during a period of study leave of the second author granted by University of Leicester.

The authors would like to thank the anonymous referees for useful comments and suggestions that helped to improve the presentation of this article.

References


