INSTYTUT INFORMATYKI
UNIWERSYTETU WROCAWSKIEGO

INSTITUTE OF COMPUTER SCIENCE
UNIVERSITY OF WROCAW

ul. Joliot-Curie 15
50–383 Wrocław
Poland

Report 04/08

Marcin Bienkowski, Paweł Zalewski

Network Exploration with Traceroute Calls

October 2008

Network Exploration with Traceroute Calls*

Marcin Bienkowski and Paweł Zalewski

Institute of Computer Science, University of Wrocław

Abstract. We consider the shortest path sampling of the graph (or, alternatively speaking, network exploration using traceroute probes). The goal is to explore the whole network using as few queries as possible. We deal with offline and online variants of this problem and give sharp estimates for the number of the queries for several graph classes.

1 Introduction

There is a growing interest in modelling the topology of the Internet. Coarse methods such as BGP tables investigation can be used to build the map of the Internet on the level of autonomous systems, each consisting of thousands or millions of nodes. However, in order to capture the shape of the Internet on the network interface level, one has to resort to more accurate, and also more demanding, methods like probing the network with traceroute calls.

A traceroute operation, implemented in most operating systems, returns a forward path from a computer at which it is run to a remote system. The path is chosen on the basis of the routing table information stored at intermediate computers or routers. In turn, the aim of the routing protocols used in the Internet is to organize the routing tables, so that the forward paths are the shortest paths between source and destination.

Most of the research in this area concentrates on querying a large set of *destinations* from a chosen set of *monitors* and reconstructing the links from the answers to these queries. An example of a working system is CAIDA [7], which uses several monitors and addresses millions of destinations. A more distributed, community-based approach, with a large number of monitors querying a few destinations each, is implemented in DIMES [1]. One of the goals of such a distributed system is to avoid duplicated effort and reducing number of links which are traveled by multiple traceroute queries. Several heuristics were proposed to deal with this problem, most notably a *doubletree* algorithm [5] or stopping rules [4, 3]. Several papers investigated the statistical relations between the number of monitors, destinations, topological properties of the graph and their influence on the expected number of explored edges and vertices (see e.g. [2, 6]).

1.1 Our Contribution

The research mentioned above is of *best-effort* type, i.e. the algorithms constructed ask a fixed number of queries and try to reconstruct as large fragment of the Internet as possible. Moreover, these algorithms are evaluated only experimentally.

In this paper, we take a complementary strategy. First of all, we want to explore the *whole* network, and we want to do this using as few traceroute queries as possible. Second, we are no longer satisfied with path covering of the whole network. We want the answers to the traceroute queries to form a *proof* that we have really discovered the shortest paths (or

* Supported by MNiSW grant number N206 001 31/0436, 2006–2008.

guaranteed approximation of them) between any pair of nodes. Third, we make no assumptions about the statistical properties of the graph. To our best knowledge this is the first approach to tackle such a variant of the shortest path sampling problem.

We present results for several classes of graphs. First, we investigate how much can be inferred from returned traceroute paths. Next, we consider *offline* exploration of a graph of n nodes. In this scenario, the graph is given and we want to explore it, using as few traceroute queries as possible. We identify several types of hard cases, which need as many as $(\Omega(n^2))$ queries, even when approximate solutions are allowed and graphs have special form (sparse, dense). One of the presented results is showing the exact number of queries $(\Theta(n \log n))$ which are needed and sufficient to explore a complete binary tree. Finally, we present $\mathcal{O}(\log n)$-competitive *online* algorithm for exploring trees.

2 General Observations

We are given an unweighted, undirected, connected graph $G = (V, E)$. Let n be the number of nodes; the set V is known. For $a, b \in V$, a *traceroute* operation (a, b) returns a shortest path between a and b.

In our model, we assume that the consecutive traceroute operations (a, b) return the same path between a and b. Query (b, a) returns the same path (with nodes in reverse order) as query (a, b). Moreover, if the path returned by query (a, b) is $(a = a_1, a_2, \ldots, a_k = b)$, then the path returned by (a_i, a_j) is $(a_i, a_{i+1}, \ldots, a_{j-1}, a_j)$, for $1 \leq i < j \leq k$.

We assume that the paths returned by a traceroute query are given with the graph. Thus, for a fixed graph, a path returned by a traceroute query is a deterministic function of this query.

We want to compute the "shape" of graph G on the basis of the traceroute operations on this graph using as few queries as possible. Note that if we considered a weighted graph or a multigraph, it might not be possible to discover all the edges (edges which do not lie on unique shortest paths).

Let $d(a, b)$ denote the length of the shortest path (in terms of the number of hops) between a and b. Clearly, d is a distance function satisfying the triangle inequality. We may view d as a symmetric, two-dimensional matrix, which we want to compute.

2.1 Gaining Knowledge

In this section we formalize the process of reasoning and decide what can be computed on the basis of the traceroute queries.

Clearly, the existence of edge $e = (u, v)$ may only be inferred if e is a part of some path p, returned as an answer. Then, we say that p gives a *yes-certificate* for e. Producing a *no-certificate*, i.e. a proof that edge e does not exist, can be more complicated. An example is depicted in Figure 1. After asking queries q_1, q_2, q_3 we know that edge (a, b) cannot exist, as p_1 is the shortest path from v_1 to v_2. However, more types of edges can be excluded. Edges (v_5, a) and (c, d) cannot exist either, as they would create shortcuts for p_3 and p_1, respectively. Note also that three edges (v_1, e), (e, f), (f, v_2) cannot simultaneously exist but every pair is possible until further queries are asked. It is easy to observe that an edge e (or a set of edges) is excluded if and only if its existence would change a set of answers.

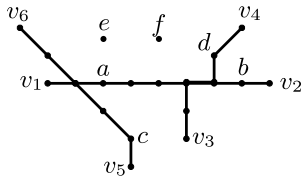

Fig. 1. Queries $q_1 = (v_1, v_2)$, $q_2 = (v_3, v_4)$, $q_3 = (v_5, v_6)$ return paths p_1, p_2, p_3, respectively. p_1 and p_2 have one common edge.

Answers to queries give us more knowledge than simple no-certificates. For instance, in the graph depicted in Figure 1, by the triangle inequality, we can deduce that $\mathrm{d}(v_1, v_4) \leq 6 + 2 = 8$ and $\mathrm{d}(v_1, v_4) \geq 6 - 2 = 4$.

However, although the triangle inequality helps to improve lower bounds on distance, using only such reasoning may not yield the best lower bound possible, as illustrated in Figure 2. If we ask queries (v_1, b) and (v_2, a), we have full knowledge about the underlying graph, as we have all yes- and no-certificates. Yet, the triangle inequality can only give the lower bound $\mathrm{d}(v_1, v_2) \geq 2$. Therefore, we assume that after every knowledge upgrade resulting from an answered query, a suitably modified algorithm for computing all shortest paths (see below) is automatically applied.

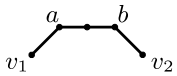

Fig. 2. Queries $q_1 = (v_1, b)$ and $q_2 = (v_2, v_a)$ should give full knowledge about the graph.

We define two auxiliary matrices L and U, which hold currently best lower and upper estimates on the shortest distances between any pair of nodes. At the beginning, for any $a \neq b$, it holds that $L(a, a) = U(a, a) = 0$, $L(a, b) = 1$, $U(a, b) = n - 1$.

A path $P = (a_{i_1}, \ldots, a_{i_k})$ resulting from a traceroute query (a_{i_1}, a_{i_k}) increases our knowledge in four ways:

direct update $L(i_s, i_t) = U(i_s, i_t) = |s - t|$, for $a_s, a_t \in P$.
upper inference Iteratively apply triangle inequality to upgrade U:
$U(i, j) = \min\{U(i, j), U(i, k) + U(k, j)\}$.
lower inference Iteratively apply triangle inequality to upgrade L:
$L(i, j) = \max\{L(i, j), L(i, k) - U(k, j)\}$.
bound augmenting If $L(i, j) > 1$, then upgrade L:
$L(i, j) = \max\{L(i, j), \mathrm{d}_{G'}(i, j)\}$, where $G' = (V, E')$ is a graph in which edge (s, t) exists if $L(s, t) = 1$.

Obviously, $L(a, b) \leq \mathrm{d}(a, b) \leq U(a, b)$. Hence, the graph is fully described when $L = U$. Note that the only rule above which can produce a new no-certificate is lower inference. In particular, bound augmenting cannot do this and can be applied only if the paths returned so far form a connected, spanning subgraph of G.

If $\mathcal{T}$ is the set of traceroute queries, then $L^{\mathcal{T}}$ and $U^{\mathcal{T}}$ are the matrices representing the knowledge gained by these queries. Sometimes, we omit superscripts at L and U if they are clear from the context.

3 Offline Scenarios

3.1 Complete Sets

First, we consider the offline complexity of the graph, i.e. a graph G is given and we want to "describe" it using as few traceroute operations as possible. A set of traceroute queries $\mathcal{T}$ is *complete* for a graph G if matrix d can be computed on the basis of these traceroutes. Formally, we define it as follows.

Definition 1. *Fix any set of traceroute queries $\mathcal{T}$. $\mathcal{T}$ is* complete *if $L^{\mathcal{T}} = U^{\mathcal{T}}$. A* traceroute complexity *of a graph G is the cardinality of the smallest complete set for G.*

Since queries (a, b) and (b, a) are equivalent (they deliver the same information), we treat a query as an *unordered* pair. Thus, for simplicity we assume that if (a, b) is in a query set then (b, a) also belongs to this set, however, they count as a single query.

In the remaining part of this section, we are interested in the traceroute complexity of various graphs. There are graphs whose traceroute complexity is 1. For example a complete set for a path (node i connected with node $i + 1$ for $1 \leq i \leq n - 1$) is $\{(1, n)\}$. Obviously, the traceroute complexity of any graph is at most $\binom{n}{2}$. This bound is matched for example for a n-node clique: we need $\binom{n}{2}$ yes-certificates and each query can only deliver one such certificate.

3.2 Approximable Complete Sets

Large traceroute complexity of cliques motivates us to relax the requirements of the complete set. For some applications it is not needed to have the exact length of the shortest path between any pair of nodes, but some approximation of it.

Definition 2. *Fix any set of traceroute operations $\mathcal{T}$. $\mathcal{T}$ is called* γ-approximable complete *if $U^{\mathcal{T}}(a, b)/L^{\mathcal{T}}(a, b) \leq \gamma$ for any a and b. A* traceroute γ-complexity *of a graph G is the cardinality of the smallest γ-approximable complete set for G.*

Lemma 1. *The traceroute 2-complexity of a n-node clique is $n - 1$.*

Proof. Clearly, a set of answers (paths) must form a connected, spanning subgraph, therefore $n - 1$ queries are necessary for a clique.

Let $\mathcal{A}$ be the set of all $n - 1$ traceroute queries $(1, i)$ for $2 \leq i \leq n$. Then for any pair i, j we have a trivial bound on the distances, i.e. $L(i, j) = 1$ and, on the other hand, $U(i, j) \leq U(1, i) + U(1, j) = 2$. □

However, it turns out that there exist graphs, whose traceroute 2-complexity is $\Omega(n^2)$. Surprisingly, there are even graphs of diameter 2 with this property.

Before we prove it, we introduce a concept of query set majorization. Informally, we call a query set $\mathcal{A}$ *not worse than* $\mathcal{B}$ (and denote $A \geq B$) if the lower and upper estimates yielded by set $\mathcal{A}$ are not worse than those of set $\mathcal{B}$.

Definition 3. *Fix two traceroute query sets, $\mathcal{A}$ and $\mathcal{B}$. Then $\mathcal{A} \geq \mathcal{B}$ if for all nodes i, j, $L^{\mathcal{B}}(i, j) \leq L^{\mathcal{A}}(i, j)$ and $U^{\mathcal{A}}(i, j) \leq U^{\mathcal{B}}(i, j)$.*

Clearly, the relation described above is transitive. If $\mathcal{A} \supseteq \mathcal{B}$, then $\mathcal{A} \geq \mathcal{B}$. Moreover, if we take any set $\mathcal{A}$ and replace a query $T(a, b) \in A$ by a query $T(a', b')$, which returns a path containing a path $a \rightsquigarrow b$, then the resulting set $\mathcal{A}'$ is not worse than $\mathcal{A}$.

Theorem 1. *There are n-node graphs of diameter 2 whose traceroute 2-complexity is $\Omega(n^2)$.*

Proof. Take any $n = 4 \cdot k$, where k is an integer. Consider the following graph G, whose example for $n = 12$ is depicted below.

Divide the set of nodes into two parts, $V = \{v_1, \ldots, v_{2k}\}$ and $W = \{w_1, \ldots, w_{2k}\}$. V and W form themselves two separate cliques. Additionally, each node v_i is connected with w_{2i-1} and w_{2i}, for $1 \leq i \leq k$. Similarly, each w_i is connected with v_{2i-1} and v_{2i}, for $1 \leq i \leq k$.

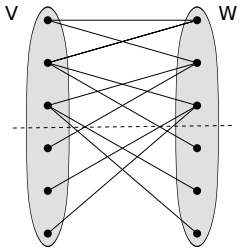

Fig. 3. Graph G of diameter 2

Let F be the subset of all pairs (i, j) such that $k < i, j \leq 2k$. Clearly, for the set of queries $\mathcal{T}$ to be 2-approximable complete, for any pair $(i, j) \in F$ one of the following conditions has to hold.

1. $L^{\mathcal{T}}(i, j) \geq 2$. As the diameter of the graph is 2, this knowledge may only result from a query (v_i, w_j).
2. $U^{\mathcal{T}}(i, j) \leq 2$. This knowledge may either may come directly from a query (v_i, w_j), or be inferred from $U^{\mathcal{T}}(v_i, w_{\lceil \frac{i}{2} \rceil}) = U^{\mathcal{T}}(w_{\lceil \frac{i}{2} \rceil}, w_j) = 1$, or from $U^{\mathcal{T}}(v_i, v_{\lceil \frac{j}{2} \rceil}) = U^{\mathcal{T}}(v_{\lceil \frac{j}{2} \rceil}, w_j) = 1$.

Thus, for any pair $(i, j) \in F$, at least one of the queries (v_i, w_j), $(v_i, w_{\lceil \frac{i}{2} \rceil})$ $(v_{\lceil \frac{j}{2} \rceil}, w_j)$ must belong to $\mathcal{T}$. We call these queries *witnesses* of (i, j). Note that any query (a, b) is a witness of at most two pairs from F. Therefore, $\mathcal{T} \geq |F|/2 = \Omega(n^2)$. □

Moreover, a similar property holds even for some trees.

Theorem 2. *There are trees whose 2-approximable complete set has $\Omega(n^2)$ elements.*

Proof. Let $n = 2 \cdot k + 1$, where k is an integer. Consider a tree T depicted in Figure 4.

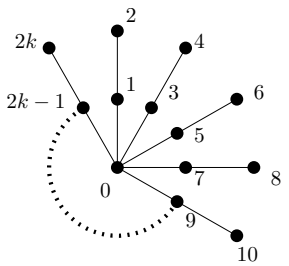

Fig. 4. Tree T

Pick any set $\mathcal{A}$ containing less than $\binom{k}{2}$ traceroute queries. We transform it into a set $\mathcal{A}' \geq A$ and show that $\mathcal{A}'$ is not 2-approximable complete for T.

Our transformation works as follows. We take any traceroute query $(a, b) \in \mathcal{A}$. Recall that $a < b$. If $a = 0$ then we replace it by a query $(2, b)$. If a or b is an odd number then we increase it by 1. We proceed till no transformation is pending. The resulting set $\mathcal{A}'$ is not worse than $\mathcal{A}$, has the same number of queries, and all queries are between pairs of leaves.

Hence, there is a pair of leaves $a < b$, such that $(a, b) \notin \mathcal{A}'$. In this case, $U(a, b) \geq 4$. For $\mathcal{A}'$ to be 2-approximable complete, $L(a, b) \geq 2$. For this to happen either, there has to be a vertex $k \neq a, b$, such that

$$L(a, k) - U(k, b) \geq 2 \quad \text{or} \quad L(a, k) - U(k, b) \geq 2 \ . \tag{1}$$

We consider the former case, the latter is identical. For (1) to hold, one of the following conditions must hold.

(i) $L(a, k) \geq 3$ and $U(k, b) = 1$. In this case, the only choice for k is $b-1$, but it would mean that $(a, b) \in \mathcal{A}'$.
(ii) $L(a, k) = 4$. In this case, k is a leaf different from b, and thus $U(k, b) = 4$.

□

The result above can easily be generalized to any constant approximation factor. Note that if we have a graph which is a normal star (all nodes connected to node n), then the argument above applies for complete sets, but for 2-approximable complete sets, $\lceil n-1 \rceil/2$ queries suffice. Indeed, it is sufficient to ask $\lceil n-1 \rceil/2$ queries between mutually disjoint pairs of different leaves. Then for any pair of leaves (a, b) we get a guarantee $U(a, b) \leq 2$ and we have a trivial lower bound $L(a, b) = 1$. This shows that there is difference between computing complete sets and 2-approximable ones. However, 2 is the smallest constant for which such separation exists.

Lemma 2. *Fix $\epsilon \in (0, 1]$. If Q is a set of queries $(2-\epsilon)$-complete for G, then Q is also complete for G.*

Proof. Let Q be $(2-\epsilon)$-complete for $G = (V, E)$. As for any $(i, j) \in E$, $d(i, j) = 1$, $U^Q(i, j) \leq 2-\epsilon$. Moreover, for any $(i, j) \notin E$, $U^Q(i, j) \geq d(i, j) \geq 2$. Therefore, we can distinguish between an existing and non-existing edge. □

4 An Offline Scenario: Binary Trees

In this section, we prove that the traceroute complexity of a complete binary trees of height $k \geq 2$ is $\frac{3}{4}(k-2)(2^{k-1}) + 2^k$. We start with an easy observation.

Observation 1 *Four queries are sufficient and necessary to describe a tree of height 2.*

These queries are depicted in Figure 5. Note that they yield no-certificates for all pairs of not connected vertices (not only between leaves).

Our next observation is based on query set majorization and similar to the one made in the proof of Theorem 2.

Observation 2 *We may assume that all the queries are between leaves of the tree.*

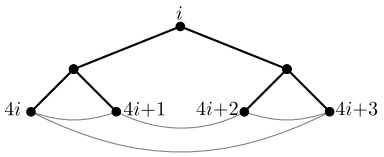

Fig. 5. Tree of height 2 with 4 queries describing it.

4.1 Lower Bound

Throughout this section, we simplify the goal of the algorithm. Instead of requiring that both matrices U and L are filled, we assume that the matrix U is already known and $U(i,j) = \mathrm{d}(i,j)$ (we say that *the set of edges is known*). We ask what is the impact of a query set on the matrix L. Hence, in the process of building the knowledge of T, we use only direct updates and lower inferences. Obviously, the knowledge of U can only make the life of the exploring algorithm easier and the obtained lower bound will also apply to the more general case of full graph exploration.

Definition 4. *Assume that the set of edges is known.* $\mathrm{Ex}(Q)$ *is the set of* no*-certificates, delivered by the set of queries* Q*. Conversely,* $\mathrm{minq}(S) = \min\{|Q| : S \subseteq \mathrm{Ex}(Q)\}$*, the minimum number of queries yielding* no*-certificate for every edge in* S*.*

First, we observe that a set of queries asked sequentially excludes the same set of edges as the sum of sets of edges excluded by isolated queries.

Lemma 3. *Let* $Q = \{q_i\}_i$ *be a set of queries. Then* $\mathrm{Ex}(Q) = \bigcup_i \mathrm{Ex}(\{q_i\})$*.*

Proof. Assume, to the contrary, that edge e is excluded by Q but not by any single q_i. Let $P = \{p_i\}_i$ be a set of answers (paths) to queries q_i. Since E is already known, no new edge is discovered by any p_i. Excluding e is equivalent to stating that its existence would change P, which means that some $p \in P$ is different (shorter)[1] when e belongs to E. But then e is excluded by q_i. □

We simplify the goal of the algorithm once more: we only require that the set of queries Q proves that there are no edges between leaves of T.

For any tree T, T_{L},T_{R} denote its left and right subtrees, $\underline{T}$ denote the set of its leaves and $\mathrm{root}(T)$ the root of T. When we speak of a subtree of T, we always mean a complete subtree rooted at some vertex. $T(r)$ denotes a subtree of T rooted at r. $T_\ell(v)$ denotes a (unique) subtree of T of height ℓ containing vertex v. By h-subtree we mean a subtree of height h. We use $A \otimes B$ as a shorthand for $A \times B \cup B \times A$.

Separating the queries In this subsection, we show that in order to exclude all the edges between leaves of the tree, we have to exclude all these edges in its subtrees and between subtrees, separately.

Lemma 4. *Let* T *be a complete binary tree of edge-height* $k \geq 2$*. Let* $\mathcal{F}$ *be any of the products:* $\underline{T_{\mathrm{L}}} \times \underline{T_{\mathrm{L}}}$*,* $\underline{T_{\mathrm{R}}} \times \underline{T_{\mathrm{R}}}$*, or* $\underline{T_{\mathrm{L}}} \otimes \underline{T_{\mathrm{R}}}$*. Then for any* $Q \subseteq \mathcal{F}$*,* $\mathrm{Ex}(Q) \subseteq \mathcal{F}$*.*

[1] Technically, adding e to E could produce as an answer to Q_i another path p_i' of the same length as p_i. Then, e would not be excluded despite having ability to change P. However, we can easily avoid this obstacle by requiring that out of all possible paths of the same length, the one existing in original graph is returned.

Proof. We show the observation for $\mathcal{F} = \underline{T_{\mathrm{L}}} \times \underline{T_{\mathrm{L}}}$. The proof for $\mathcal{F} = \underline{T_{\mathrm{R}}} \times \underline{T_{\mathrm{R}}}$ is identical. We show that at any stage of our reasoning we cannot apply the lower inference rule for any pair $(x, y) \notin \underline{T_{\mathrm{L}}} \times \underline{T_{\mathrm{L}}}$ (thus we cannot infer a no-certificate for such a pair). We can infer $L(x, y) > 1$ only if there exists $a \in T$ such that $L(x, a) - U(y, a) > 1$. For this to happen both x and a has to belong to T_{L} (otherwise $L(x, a) = 1$). This implies that $y \in T_{\mathrm{R}}$ and hence $U(y, a) = 2 \cdot k$. Therefore, $L(x, a) - U(y, a) < 0$.

Second, we take $\mathcal{F} = \underline{T_{\mathrm{L}}} \otimes \underline{T_{\mathrm{R}}}$. We follow the same pattern, fixing a pair $(x, y) \notin \underline{T_{\mathrm{L}}} \otimes \underline{T_{\mathrm{R}}}$ W.l.o.g. $x, y \in \underline{T_{\mathrm{L}}}$. Again, for inferring $L(x, y) > 1$, we need a node $a \in T$, such that $L(x, a) - U(y, a) > 1$. But then $a \in T_{\mathrm{R}}$ and $L(x, a) = U(y, a) = 2 \cdot k$. □

Corollary 1. *If T is a complete binary tree of edge-height $k \geq 2$, then*

$$\mathrm{minq}(\underline{T} \times \underline{T}) = \mathrm{minq}(\underline{T_{\mathrm{L}}} \otimes \underline{T_{\mathrm{R}}}) + \mathrm{minq}(\underline{T_{\mathrm{L}}} \times \underline{T_{\mathrm{L}}}) + \mathrm{minq}(\underline{T_{\mathrm{R}}} \times \underline{T_{\mathrm{R}}}) \ .$$

Proof. We prove two inequalities.

(i) Let A, B and C be any minimal sets of queries excluding, respectively, $\underline{T_{\mathrm{L}}} \otimes \underline{T_{\mathrm{R}}}$, $\underline{T_{\mathrm{L}}} \times \underline{T_{\mathrm{L}}}$, and $\underline{T_{\mathrm{R}}} \times \underline{T_{\mathrm{R}}}$. Then $A \cup B \cup C$ excludes $\underline{T} \times \underline{T}$, and thus

$$\begin{aligned} \mathrm{minq}(\underline{T} \times \underline{T}) \ &\leq |A \cup B \cup C| \leq |A| + |B| + |C| \\ &= \mathrm{minq}(\underline{T_{\mathrm{L}}} \otimes \underline{T_{\mathrm{R}}}) + \mathrm{minq}(\underline{T_{\mathrm{L}}} \times \underline{T_{\mathrm{L}}}) + \mathrm{minq}(\underline{T_{\mathrm{R}}} \times \underline{T_{\mathrm{R}}}) \ . \end{aligned}$$

(ii) Take any minimal set of queries Q excluding $\underline{T} \times \underline{T}$. By Lemma 3, any pair from $\underline{T_{\mathrm{L}}} \otimes \underline{T_{\mathrm{R}}}$ is excluded by some query $q \in Q$. By Lemma 4, $q \in \underline{T_{\mathrm{L}}} \times \underline{T_{\mathrm{R}}}$. Hence, $Q \cap (\underline{T_{\mathrm{L}}} \otimes \underline{T_{\mathrm{R}}})$ excludes $\underline{T_{\mathrm{L}}} \otimes \underline{T_{\mathrm{R}}}$. Identical reasoning can be performed for $\underline{T_{\mathrm{L}}} \times \underline{T_{\mathrm{L}}}$ and $\underline{T_{\mathrm{R}}} \times \underline{T_{\mathrm{R}}}$.
Hence,

$$\begin{aligned} \mathrm{minq}(\underline{T} \times \underline{T}) = |Q| \ &= |Q \cap (\underline{T_{\mathrm{L}}} \otimes \underline{T_{\mathrm{R}}})| + |Q \cap (\underline{T_{\mathrm{L}}} \times \underline{T_{\mathrm{L}}})| + |Q \cap (\underline{T_{\mathrm{R}}} \times \underline{T_{\mathrm{R}}})| \\ &\geq \mathrm{minq}(\underline{T_{\mathrm{L}}} \otimes \underline{T_{\mathrm{R}}}) + \mathrm{minq}(\underline{T_{\mathrm{L}}} \times \underline{T_{\mathrm{L}}}) + \mathrm{minq}(\underline{T_{\mathrm{R}}} \times \underline{T_{\mathrm{R}}}) \ . \end{aligned}$$

□

Excluded sets

Lemma 5. *Let T be a complete binary tree of height k. Let $v_i \in \underline{T_{\mathrm{L}}}$ and $v_j \in \underline{T_{\mathrm{R}}}$. Then*

$$(\underline{T_{\mathrm{L}}} \otimes \underline{T_{\mathrm{R}}}) \cap \mathrm{Ex}(\{(v_i, v_j)\}) = \bigcup_{s+t=k-1} \underline{T_s(v_i)} \times \underline{T_t(v_j)}. \tag{2}$$

Proof. We get no-certificates only for these edges, whose existence would change the answer for the query (v_i, v_j). First, we show that all edges of the right hand side of (2) are indeed excluded. Suppose, to the contrary, that there exists edge $(w_1, w_2) \in \underline{T_s(v_i)} \times \underline{T_t(v_j)}$, for some s, t, such that $s + t = k - 1$. Then,

$$1 = d(w_1, w_2) \geq L(w_1, w_2) \geq L(v_i, v_j) - U(v_i, w_1) - U(v_j, w_2) = 2k - 2s - 2t \geq 2.$$

Using a similar technique, it is easy to observe that the existence of other edges does not affect the answer to our query and thus no other edge is excluded. □

Now we focus on the lower bound on the cardinality of a set Q eliminating all edges from a given subset of $\underline{T_L}$ to every leaf of T_R.

Definition 5. *A set of queries Q is* sufficient within $T' \subseteq T_L$ *if*

$$\underline{T'} \otimes \underline{T_R} \subseteq \mathrm{Ex}(Q \cap (\underline{T'} \otimes \underline{T_R})).$$

The deficit *of Q in T' is the number of edges from $\underline{T'} \otimes \underline{T_R}$ not excluded by $Q \cap (\underline{T'} \otimes \underline{T_R})$.*

Lemma 6. *For any subtree $A \subseteq T_L$ of height $h \geq 2$, if a query set Q is sufficient within A, then $|Q \cap (\underline{A} \otimes \underline{T_{\mathrm{R}}})| \geq \frac{3}{4} \cdot 2^h$.*

Proof. The proof is by induction on h. For $h = 2$, it is easy to observe that two queries are not sufficient to exclude every edge in $\underline{A} \otimes \underline{T_R}$.

Suppose that the lemma holds for every $2 \leq h' < h$. Fix any Q sufficient within A. If Q is sufficient within both A_{L} and A_{R} then, by induction, $|Q \cap (\underline{A} \otimes \underline{T_{\mathrm{R}}})| = |Q \cap (\underline{A_{\mathrm{L}}} \otimes \underline{T_{\mathrm{R}}})| + |Q \cap (\underline{A_R} \times \underline{T_R})| \geq 2 \cdot \frac{3}{4} \cdot 2^{h-1} = \frac{3}{4} \cdot 2^h$.

Otherwise, we show how to transform Q obtaining a query set with the same cardinality with smaller total deficit in A_{L} and A_{R}. This process must eventually stop, leaving Q sufficient within both A_{L} and A_{R}. In the remaining part of the proof, we assume that Q is not sufficient within A_{L} and show such transformation.

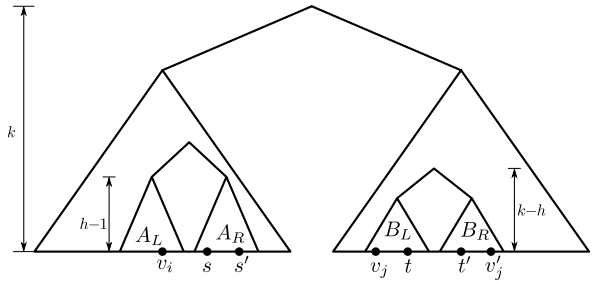

Fig. 6. Illustration of Lemma 6

By the definition, there exists $(v_i, v_j) \in \underline{A_{\mathrm{L}}} \times \underline{T_{\mathrm{R}}}$ which is not excluded by $Q \cap (\underline{A_{\mathrm{L}}} \otimes \underline{T_{\mathrm{R}}})$. Let $B = T_{k-h}(v_j)$. We show that no pair $v_i \times \underline{B}$ is excluded by $Q \cap (\underline{A_{\mathrm{L}}} \otimes \underline{T_{\mathrm{R}}})$. Assume the contrary, i.e. there exists a pair $(v_i, v'_j) \in \underline{A_{\mathrm{L}}} \times \underline{B}$ which is excluded by $Q \cap (\underline{A_{\mathrm{L}}} \otimes \underline{T_{\mathrm{R}}})$. By Lemma 3, (v_i, v'_j) is excluded by a single query $(w_1, w_2) \in A_{\mathrm{L}} \times T_{\mathrm{R}}$ and by Lemma 5, there exists a, such that $v_i \in T_{a-1}(w_1)$ and $v'_j \in T_{k-a}(w_2)$. As $v_i, w_1 \in A_{\mathrm{L}}$, $a \leq h$, and thus $B = T_{k-h}(v'_j) \subseteq T_{k-a}(v'_j) = T_{k-a}(w_2)$. But this implies that $v_i \in T_{a-1}(w_1)$ and $v_j \in T_{k-a}(w_2)$, i.e. (v_i, v_j) would be also excluded by (w_1, w_2).

Therefore, no pair from $v_i \times \underline{B}$ is excluded by $Q \cap (\underline{A_{\mathrm{L}}} \otimes \underline{T_{\mathrm{R}}})$. On the other hand, since Q is sufficient within A, all such pairs have to be excluded by $Q \cap (\underline{A_{\mathrm{R}}} \otimes \underline{T_{\mathrm{R}}})$. Now we take any query from $\underline{A_{\mathrm{R}}} \otimes \underline{T_{\mathrm{R}}}$. By Lemma 5, it may only exclude a pair either from $v_i \times \underline{B_{\mathrm{L}}}$ (and only if one end of the query belongs to $\underline{B_{\mathrm{L}}}$) or from $v_i \times \underline{B_{\mathrm{R}}}$ (if one end belongs to $\underline{B_{\mathrm{R}}}$).

Thus, Q contains at least two queries from $\underline{A_{\mathrm{R}}} \otimes \underline{T_{\mathrm{R}}}$, say (s, t) and (s', t'), where $t \in \underline{B_{\mathrm{L}}}$ and $t' \in \underline{B_{\mathrm{R}}}$. Note that "from the point of view" of A_R, these queries are equivalent. Both queries exclude every edge of $A_R \times B$. Moreover, they exclude the same set of edges from $A_R \times (T_R \setminus B)$. Hence, $\mathrm{Ex}(\{(s,t)\}) \cap (A_R \otimes T_R) = \mathrm{Ex}(\{(s',t')\}) \cap (A_R \otimes T_R)$. Finally, we may replace (s', t') by (v_i, v_j) not changing the deficit of Q in A_{R} and reducing it in A_{L}. □

Corollary 2. *For any tree T' of height $h \geq 3$, it holds that* $\mathrm{minq}(\underline{T'_{\mathrm{L}}} \otimes \underline{T'_{\mathrm{R}}}) \geq \frac{3}{4} \cdot 2^{h-1}$.

Theorem 3. *Let $n = 2^{k+1} - 1$ be the number of nodes in a complete binary tree T of height $k \geq 2$ and let Q be a complete set of queries for T. Then, $|Q| \geq \frac{3}{4}(k-2)(2^{k-1}) + 2^k$.*

Proof. For $k = 2$ the theorem follows directly from Observation 1.

Assume that the theorem holds for trees of height $k-1$, we show it for tree T of height k. By Corrolary 2, at least $\frac{3}{4} \cdot 2^{k-1}$ queries are needed to exclude all edges from the set $\underline{T_{\mathrm{L}}} \otimes \underline{T_{\mathrm{R}}}$. Moreover, by Lemma 4, answers to these queries give no information about possible existence of edges from $\underline{T_{\mathrm{L}}} \times \underline{T_{\mathrm{L}}}$ or $\underline{T_{\mathrm{R}}} \times \underline{T_{\mathrm{R}}}$. By the inductive assumption, the number of queries needed for giving all necessary no-certificates for $\underline{T_{\mathrm{L}}} \times \underline{T_{\mathrm{L}}}$ is at least $\frac{3}{4}(k-3)(2^{k-2}) + 2^{k-1}$ (the same holds for $\underline{T_{\mathrm{R}}} \times \underline{T_{\mathrm{R}}}$). Hence, at least $2 \cdot (\frac{3}{4}(k-3)(2^{k-2}) + 2^{k-1}) + \frac{3}{4} \cdot 2^{k-1} = \frac{3}{4}(k-2)(2^{k-1}) + 2^k$ queries are necessary. □

4.2 Upper Bound

In this section, we show that $\frac{3}{4}(k-2)(2^{k-1})+2^k$ queries are also sufficient for describing a binary tree of height k. As in the previous section, our construction will be recursive. However, this time, we have to handle the pairs whose ends are not leaves as well. Below we present the crucial lemma of this section.

Lemma 7. *Let T' be any subtree with height $h \geq 3$. Then $\frac{3}{4} \cdot 2^{h-1}$ queries suffice to generate no-certificates for all pairs from $(T'_{\mathrm{L}} \otimes T'_{\mathrm{R}}) \cup (\mathrm{root}(T)) \otimes (T'_{\mathrm{L}} \setminus \mathrm{root}(T'_{\mathrm{L}})) \cup T'_{\mathrm{R}} \setminus \mathrm{root}(T'_{\mathrm{R}})))$.*

First, we show our construction. We label all the vertices in a natural way: the root has empty label; for the vertex v with label s_v its left son has label s_v0 and its right son has label s_v1. Thus, all the leaves of T' have h-bit labels. For any bit string s, $\overline{s}$ denotes this string in reversed order. For any such 2-subtree of T'_{L} whose root has label $0s$, we ask three queries, $(v_{0s00}, v_{10\overline{s}0})$, $(v_{0s10}, v_{110\overline{s}})$, and $(v_{0s11}, v_{111\overline{s}})$. Thus, the right ends of these queries are from T'_{RL}, T'_{RRL}, and T'_{RRR}, respectively. We call any such pair *queried*. An example is shown in Figure 7.

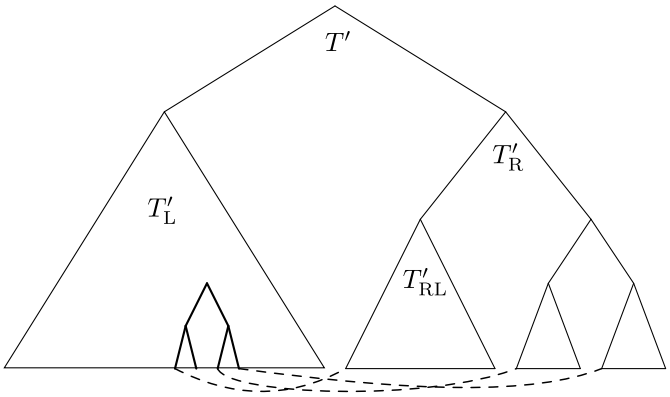

Fig. 7. An example of a 2-subtree and related queries.

Proof (of Lemma 7). In the following, we show that the construction above generates all no-certificates for pairs from $T'_{\mathrm{L}} \otimes T'_{\mathrm{RL}}$. Showing no-certificates for pairs from $T'_{\mathrm{L}} \otimes T'_{\mathrm{RRL}}$ and $T'_{\mathrm{L}} \otimes T'_{\mathrm{RRR}}$ is analogous.

Fix any $(v_\ell, v_r) \in T'_{\mathrm{L}} \times T'_{\mathrm{RL}}$ and let $f \leq h-1$ be the height of v_ℓ. We assume that $f \geq 2$. The case $f < 2$ can be proven similarly.

If the height of v_r is at least $h-f$, then let T'' be any $h-f$ subtree of v_r. Otherwise, let $T'' = T_{h-f}(v_r)$. In either case, T'' has height $h-f$ and the distance between v_r and any leaf of T'' is at most $2 \cdot h - f - 2$.

Now let $0s_\ell$ be a label of v_ℓ and let $10s_r$ be a label of the root of T''. Then a pair $(a, b) = (v_{0s_\ell \overline{s_r} 00}, v_{10s_r \overline{s_\ell} 0}) \in T(v) \times T''$ is queried. An edge between v_ℓ and v_r would imply that the distance between a and b is at most $d(a, v_\ell) + d(v_\ell, v_r) + d(v_r, b) \leq f + 1 + 2 \cdot h - f - 2 = 2 \cdot h - 1$, i.e. such a path would be returned by a query (a, b).

Finally, we observe that the same approach works for proving no-certificates between root(T) and all nodes of T which are not connected to root(T). □

Theorem 4. *$\frac{3}{4}(k-2)(2^{k-1}) + 2^k$ queries are sufficient for describing a binary tree of height k.*

Proof. We observe that the queries described above cover the whole tree. Thus, for any tree edge we get a yes-certificate.

We prove the theorem inductively. For $k = 2$ the theorem is equivalent to Observation 1. Assume that the statement holds for trees of height $k-1$, we show it for tree T of height k. This means that $2 \cdot (\frac{3}{4}(k-3)(2^{k-2}) + 2^{k-1})$ queries are sufficient to generate all no-certificates within T_{L} and T_{R}. By Lemma 7, $\frac{3}{4} \cdot 2^{k-1}$ queries are sufficient to generate all remaining no-certificates in T. Hence at most $2 \cdot (\frac{3}{4}(k-3)(2^{k-2}) + 2^{k-1}) + \frac{3}{4} \cdot 2^{k-1} = \frac{3}{4}(k-2)(2^{k-1}) + 2^k$ queries suffice. □

5 Online Algorithms

In this section, we show how to construct an online algorithm Cover, which constructs a complete set of queries not knowing the graph, under the assumption that a graph is a tree. Moreover, if the graph is not a tree, the algorithm will detect it. The cost of such an algorithm will be at most $\mathcal{O}(\log n)$ times higher than the cost of an optimal offline algorithm.

Consider the following online approach, which explores any tree T using traceroute queries. It consists of two parts: *covering* and *verification*. In the covering part, the algorithm asks several traceroute queries, such that the paths returned by them cover the whole tree. In the verification part, the algorithm asks queries which verify that the structure is indeed a tree. This, however, is an offline problem. Moreover, if the explored graph is not a tree, then the algorithm will discover it during verification.

Thus, in the following, we describe and analyze only the *covering part* of the algorithm Cover. It maintains a set of *explored nodes* S. At the beginning $S = \{1\}$. The algorithm is simple: it chooses (uniformly at random) a node $x \notin S$, asks a query $(1, x)$, and adds all the nodes returned by this traceroute call to S.

We observe that during the runtime of the algorithm, S contains nodes which belong to a contiguous subtree of T. Moreover, the paths returned to queries asked so far by the algorithm create a edge-cover of this subtree. We may prove an upper bound on the expected number of traceroute needed by Cover.

Lemma 8. *Pick any set of paths $\{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k\}$, which together cover all the vertices of a tree T. Then* Cover *covers this tree using on expectation $\mathcal{O}(k + \sum_{i=1}^{k} \log |\mathcal{P}_i|)$ traceroute queries.*

Before we prove this lemma, we note that the lemma above works for any choice of such paths. Obviously, we get the best bounds by choosing path set which minimizes the expression $k + \sum_{i=1}^{k} \log |\mathcal{P}_i|$. In particular, one path suffices for a line and we get that Cover needs $\mathcal{O}(\log n)$ queries.

Proof (of Lemma 8). Fix any set of paths $\{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k\}$. Note that we do not assume that they are vertex- or edge-disjoint.

Let $Y := |\{1 \leq i \leq k : S \cap P_i = \emptyset\}|$ be the number of paths which are completely unexplored by Cover. Let $X_i = |P_i \setminus S|$ be the number of unexplored vertices on path P_i. Let $U_i = 1$ if $X_i \geq 1$ and $U_i = 0$ otherwise, i.e. $U_i = \min\{X_i, 1\}$

We define the following potential function.

$$\Phi = Y + c \cdot \sum_{i=1}^{k} \log(\max\{X_i, 1\}) + \sum_{i=1}^{k} U_i \ , \tag{3}$$

where $c = (\log(3/2))^{-1}$. At the very beginning $\Phi \leq (k-1) + \sum_{i=1}^{k} c \cdot \log |P_i| + (k-1) = \mathcal{O}(k + \sum_{i=1}^{k} \log |P_i|)$. Φ is always non-negative and when $\Phi = 0$, then $Y = 0$, $X_i = 0$ and $U_i = 0$ for all i, i.e. the exploration ends.

We observe that all the terms ocurring in the definition of Φ may only decrease during runtime of Cover. Below, we prove that each traceroute call performed by Cover decreases the potential at least by 1 with probability at least 1/3. Let $(1, x)$ be such a query; we consider three cases.

1. Node x lies on a completely unexplored path $\mathcal{P}_i$. Y, and in effect Φ, decreases at least by 1.
2. Node x lies on a path $\mathcal{P}_i$ and is the last unexplored vertex of $\mathcal{P}_i$, i.e. $X_i = 1$. Then U_i decreases from 1 to 0, causing potential to decrease at least by 1.
3. Node x lies on a path $\mathcal{P}_i$ which is already partially explored and $X_i \geq 2$. In this case, we look at the change of X_i. We observe that $S \cap \mathcal{P}_i$ occupies a contiguous fragment of $\mathcal{P}_i$; we denote it S_i. We denote the number of vertices remaining on both sides of S_i by X_i' and X_i'', respectively. By the definition, $X_i = X_i' + X_i''$. An example is depicted on Figure 8.

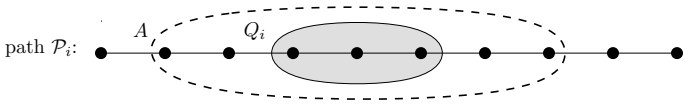

Fig. 8. Exploration of a path $\mathcal{P}_i$

Now, let A be the set of nodes of $\mathcal{P}_i$, whose distance to (the closest vertex of) the S_i is at most $X_i/3$. By this definition, $A \setminus S_i$ has at most $2 \cdot (X_i/3)$ vertices, thus with probability at least 1/3, Cover chooses a node x outside A. Further, the distance between any such x and S_i is greater than $X_i/3$, and thus any such choice leads to decreasing X_i by at least one third.

In this case, the term $c \cdot \log(\max\{X_i, 1\}) = c \cdot \log X_i$ decreases at least to $c \cdot \log(\max\{(2/3) \cdot X_i, 1\}) = c \cdot \log(2/3) \cdot X_i = c \cdot \log X_i - 1$.

Altogether, we obtain the following random process. Each traceroute call decreases Φ at least by 1 with probability at least 1/3. It follows that the expected number of traceroute calls to cover the whole tree T (i.e. to decrease Φ from $\mathcal{O}(k + \sum_{i=1}^{k} \log |P_i|)$ to 0) is $3 \cdot \mathcal{O}(k + \sum_{i=1}^{k} \log |P_i|) = \mathcal{O}(k + \sum_{i=1}^{k} \log |P_i|)$. □

Theorem 5. COVER *uses at most* $\mathcal{O}(\log n)$ *times more queries to explore any tree* T *than the traceroute complexity of the tree.*

Proof. Fix any tree T of n nodes and let OPT be its traceroute complexity. We observe that OPT is lower-bounded by the minimum number of paths which can cover T. Fix any such path cover $\mathcal{P} = \{P_1, P_2, \ldots, P_k\}$. Then, by Lemma 8, COVER uses $\mathcal{O}(k + \sum_{i=1}^{k} \log |\mathcal{P}_i|)$ queries in the covering phase and at most OPT queries in the verification phase. As OPT $\geq k$, we get that COVER uses at most $\mathcal{O}(2 \cdot \text{OPT} + \text{OPT} \cdot \max_i \log |\mathcal{P}_i|) = \mathcal{O}(\log n) \cdot \text{OPT}$. □

References

1. S. Carmi, S. Havlin, S. Kirkpatrick, Y. Shavitt, and E. Shir. A model of internet topology using k-shell decomposition. *Proceedings of the National Academy of Sciences of the United States of America*, 104(27):11150–11154, 2007.
2. L. Dall'Asta, I. Alvarez-Hamelin, A. Barrat, A. Vázquez, and A. Vespignani. Exploring networks with traceroute-like probes: Theory and simulations. *Theoretical Computer Science*, 355(1):6–24, 2006.
3. B. Donnet and T. Friedman. Topology discovery using an address prefix based stopping rule. In *Proc. of the Networks and Applications Towards a Ubiquitously Connected World (EUNICE)*, pages 119–130, 2006.
4. B. Donnet, T. Friedman, and M. Crovella. Improved algorithms for network topology discovery. In *Proc. of the Passive and Active Network Measurement*, pages 149–162, 2005.
5. B. Donnet, P. Raoult, T. Friedman, and M. Crovella. Efficient algorithms for large-scale topology discovery. *ACM SIGMETRICS Performance Evaluation Review*, 33(1):327–338, 2005.
6. A. Fekete and G. Vattay. Shortest path discovery in complex networks. *CoRR*, abs/0810.1428v2, 2008.
7. B. Huffaker, D. Plummer, D. Moore, and k claffy. Topology discovery by active probing. In *Proc. of the 2002 Symposium on Applications and the Internet (SAINT) Workshops*, pages 90–96, 2002.