Graph Connectivity with Noisy Queries

Dimitris Fotakis
National Technical University of Athens, Greece

Evangelia Gergatsouli
University of Wisconsin-Madison, WI, USA

Charilaos Pipis
National Technical University of Athens, Greece

Miltiadis Stouras
École Polytechnique Fédérale de Lausanne, Switzerland

Christos Tzamos
University of Wisconsin-Madison, WI, USA
University of Athens, Greece

Abstract

Graph connectivity is a fundamental combinatorial optimization problem that arises in many practical applications, where usually a spanning subgraph of a network is used for its operation. However, in the real world, links may fail unexpectedly deeming the networks non-operational, while checking whether a link is damaged is costly and possibly erroneous. After an event that has damaged an arbitrary subset of the edges, the network operator must find a spanning tree of the network using non-damaged edges by making as few checks as possible.

Motivated by such questions, we study the problem of finding a spanning tree in a network, when we only have access to noisy queries of the form "Does edge e exist?". We design efficient algorithms, even when edges fail adversarially, for all possible error regimes; 2-sided error (where any answer might be erroneous), false positives (where "no" answers are always correct) and false negatives (where "yes" answers are always correct). In the first two regimes we provide efficient algorithms and give matching lower bounds for general graphs. In the False Negative case we design efficient algorithms for large interesting families of graphs (e.g. bounded treewidth, sparse). Using the previous results, we provide tight algorithms for the practically useful family of planar graphs in all error regimes.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases algorithms under uncertainty, graph connectivity, spanning tree, noisy queries, online algorithms, stochastic optimization

Digital Object Identifier 10.4230/LIPIcs.MFCS.2023.47

Related Version *Full Version*: `https://arxiv.org/abs/2208.10423`

Funding This work is partially supported by NTUA Basic Research Grant (PEBE 2020) "Algorithm Design through Learning Theory: Learning-Augmented and Data-Driven Online Algorithms (LEADAlgo)".

1 Introduction

From road and railway networks to electric circuits and computer networks, maintaining an operational spanning subgraph of a network is crucial in many real-life applications. This problem can be viewed as a traditional combinatorial optimization problem, graph connectivity. However, in real-life applications, edges sometimes fail; a natural destruction on the road or a malfunction in the connector cable can render a link in the network non

© Dimitris Fotakis, Evangelia Gergatsouli, Charilaos Pipis, Miltiadis Stouras, and Christos Tzamos; licensed under Creative Commons License CC-BY 4.0

operational. Worse still, detecting faulty connections is not always that simple; our tools may be unreliable, resulting in untrustworthy indications. Can we efficiently find an operational spanning tree, when we cannot obtain reliable signals on the operation of edges?

In this work we study the problem of finding a spanning tree using noisy queries on the existence of edges. Specifically, given a moldgraph G, where some subgraph H of G is realized, we have access to an oracle that answers questions of the form *"Does edge e exist in H?"*. The answers the oracle gives are *inconsistent*: for a specific edge e, the answer might differ every time. Queries to the oracle are costly, therefore our goal is to find a spanning tree asking as few queries as possible. We design efficient algorithms that achieve this goal, in all 3 different error regimes; 2-sided (where any answer might be erroneous), 1-sided false positives (where "no" answers are always correct) and 1-sided false negatives (where "yes" answers are always correct).

1.1 Our Contribution

As a warm-up, we begin by solving the simpler problem of learning whether the tree is connected (Theorem 7). Following that, we proceed to our main problem, which is finding a realized spanning tree in three different error regimes. In the 2-sided error case, we give an algorithm performing $O(m \log m)$ queries and show that no algorithm can perform better, even on special cases like sparse graphs. In the 1-sided False Negative error regime, we design an algorithm that is optimal on planar graphs and yields efficient guarantees for other special families like graphs with treewidth k or degeneracy k ($O(km)$ queries) and graphs with Hadwiger number k ($O(k\sqrt{\log k}m)$ queries). The same algorithm can be used for general graphs, and outside the aforementioned families, its performance gracefully degrades to that of the naive strategy (that is $O(m \log n)$ queries). It is noteworthy that this is the only algorithm that breaks the $O(m \log m)$ barrier and it achieves it by employing an adaptive strategy.

In the False-Positive error case, our algorithm obtains tight guarantees for general graphs (Theorem 19), while in the special case where the realized graph is acyclic and the moldgraph is planar, the query complexity becomes linear. Both in the False-Negative and the False-Positive case, our algorithms *do not need to know* whether the graph has any special properties (sparsity, acyclic realized subgraph etc), they can simply run a unified algorithm that achieves the best guarantee according to each case.

Our results, imply tight algorithms for planar graphs, which is a family of graphs that is frequently encountered in road/railway networks, electrical circuits, image processing/computer vision [15, 14].

1.2 Related Work

Variations of this problem have been studied in literature, however they differ from our setting. A crucial assumption in all of the previous works, is that each edge has an independent probability of existence, which however, does not always bear out in practice. For example, a power outage may deem several local network links non-operational. In our work, we allow the existence of different edges to be arbitrarily correlated.

Feige et al. in [4] first studied the evaluation of boolean decision trees where the nodes are noisy and the goal is to find the correct leaf within a tolerance parameter Q. In [6, 5] the authors study finding spanning trees in Erdos-Renyi graphs, a special class of random graphs where each edge exists independently with some probability. In their setting each edge has a *query cost*, and the goal is to find a spanning tree using the least number of queries. However, contrary to their setting, we handle both adversarially selected realized graphs and noisy answers to existence queries. More recently, Lyu et al. in [11] also studied the problem of finding a MST when the edges can fail with some probability. The work of Hoffmann et al. [7] and Erlebach and Hoffmann [3] considers verifying spanning trees where each edge has a weight inside an uncertainty region and the algorithm can ask the edge to reveal it. The goal is to find a small weight spanning tree without asking all the edges for their weight. None of these settings account for *inconsistent* queries, like in our setting.

In Bhaskara at al. [1] they studied the problem of finding a shortest path in a graph where there are ML-generated hints on the edge lengths, and also costly access to a zero-error oracle. The goal is to find a good enough shortest path, without asking the oracle too many times.

On a different problem in Mazumdar and Saha [12], the authors study clustering in a similar setting to ours where they have access to queries of the form "is u and v in the same cluster?". However in their model, the oracle gives *consistent* answers on each query.

2 Preliminaries

In GRAPH CONNECTIVITY WITH NOISY QUERIES we are given a graph $G = (V, E)$, with edge set E and node set V called the *moldgraph*, then an adversary selects a *connected* subgraph G' of G to be realized. The goal of the algorithm is to find a spanning tree in the subgraph G' spending as little time as possible gathering information. The algorithm does not directly observe the subgraph that is realized, but only has access to an oracle that answers questions of the form "*Is edge $e \in E$ realized?*". Each call to this oracle costs 1, therefore our goal is to find a realized spanning tree, with constant probability using the minimum number of queries to the oracle.

This oracle, however, is noisy and inconsistent; it might not give the correct answer and when asked multiple times on the same edge, it may give different answers. More formally, the oracle is a function $\mathcal{O} : E \to \{\text{Yes}, \text{No}\}$, that given an edge $e \in E$ answers either *Yes* or *No*, indicating whether the edge is realized or not. We study all the possible error types for the oracle, outlined below.

2-sided error. the oracle's answer is wrong (ie. with "No" for realized edges and "Yes" for non-realized) with *constant* error probability $p < 1/2$.

1-sided error, False Negative (FN). if the edge is not realized then the response is always "No". If it is realized then the response is "No" with *constant* error probability $p < 1/2$, and "Yes" with probability $(1 - p)$. Thus, when the oracle responds "Yes" then it is certain that this edge is realized, but when it responds "No" then the edge may or may not be realized, hence the False Negative responses.

1-sided error, False Positive (FP). if the edge is realized then the response is always "Yes". If it is not realized then the response is "Yes" with *constant* error probability $p < 1/2$, and "No" with probability $(1 - p)$. Thus, when the oracle responds "No" then it is certain that this edge is not realized, but when it responds "Yes" then the edge may or may not be realized, hence the False Positive responses.

2.1 Graphs notation

We present some useful definitions for concepts we use throughout the paper. We begin by our definition of sparsity. Another possible name for the ρ parameter in the literature is "average degree" of the graph.

▶ **Definition 1** (ρ-sparse Graphs). *A graph G with m edges and n vertices is called ρ-sparse if $m \le \rho n$.*

It is noteworthy that our definition of ρ-sparsity for graphs is weaker than the usual definitions of graph sparsity, in the sense that it can be easily satisfied without requiring local sparsity properties. This means that constructing efficient algorithms for this property, directly translates to efficient algorithms for other usual sparsity parameters. For example, using our definition, k-degenerate graphs are k-sparse [9], k-treewidth graphs are also k-sparse [13], and graphs with Hadwiger number k are $O(k\sqrt{\log k})$-sparse [8, 2].

▶ **Definition 2** (Edge Contraction). *The edge contraction operation on an edge $e = \{u, v\}$ of a graph G results in a new graph G' wherein u and v are replaced by a new vertex uv which is connected to all vertices of G that were incident to either u or v.*

▶ **Definition 3** (Graph minor [10]). *A graph H is called a minor of a graph G, if H can occur after applying a series of edge deletions, vertex deletions, and edge contractions.*

▶ **Definition 4** (Minor-closed Graph Family). *A set $\mathcal{F}$ of graphs is called a minor-closed graph family if for any $G \in \mathcal{F}$ all minors of G also belong to $\mathcal{F}$.*

The algorithms that we present for FN and FP queries produce and handle multigraphs during intermediate steps. Here, we present and define some of the main concepts we are using from multigraphs.

An edge $e = \{u, v\}$ in a multigraph is a link connecting u and v. There might be other parallel edges between u and v, and each one is distinct. Sometimes we will need to work with the set of all parallel edges between u and v. We call this set a *super-edge* between u and v. For simplicity, we consider simple edges to be super-edges with size 1.

▶ **Definition 5** (Neighborhood). *For a vertex $v \in V$, we denote by $N(v)$ its* neighborhood*, which is the set of all super-edges with v as one of the endpoints.*

▶ **Definition 6** (Degree). *For a node $v \in V$, degree $deg(v)$ of v is the size of $N(v)$.*

Finally, we note that whenever an algorithm performs an edge-contraction operation on edge e of a multigraph, then we delete all the other parallel edges to e as well, leaving no self-loops in the graph. Moreover, if any parallel super-edges result after the contraction, they are replaced by a larger super-edge, their union.

3 Verifying Connectivity

As a warm-up, before attempting to find a realized spanning tree of the graph, we begin by presenting an algorithm for verifying the connectivity of a tree. Specifically, given a tree, which could be the operating spanning tree of a network, we want to verify whether it is connected or not, that is whether all of its edges are realized.

Naively verifying this property (ie. performing a fixed number of queries on every edge) needs $O(n \log n)$ queries to yield a constant probability guarantee. However, we show that it is possible to verify the connectivity of the tree using only $O(n)$ queries, which is asymptotically the same as if our oracle had no noise.

Algorithm 1 $Verify(T)$: A verification protocol for tree connectivity with 2-sided error.

Input: Tree $\mathcal{T}$ of n edges, **Parameters**: ε, δ, p
Output: True iff T is connected

$threshold \leftarrow \lceil \log_{\frac{1-p}{p}} \left(\frac{1}{\delta}\right) \rceil$
$budget \leftarrow \lceil \frac{1}{\epsilon} \cdot \frac{1}{1-2p} \rceil \cdot threshold \cdot n$
for $e \in E(T)$ **do**
  counter $\leftarrow 0$
  while (counter $<$ threshold) & (budget > 0) **do**
    q $\leftarrow$ Query edge e
    budget $\leftarrow$ budget -1
    if q is "Yes" **then**
      counter $\leftarrow$ counter $+1$
    else
      counter $\leftarrow$ counter -1
    end if
  end while
  if budget $= 0$ **then**
    return False
  end if
end for
return True

Our algorithm (Algorithm 1) has a predetermined budget of queries that depends on the requested guarantees. It fixes an ordering of the edges, and for each edge it performs queries until it receives c more positive answers than negative ones (where c is also fixed and depends on the requested guarantees). For realized edges, since $p < 1/2$, the expected number of queries until they reach c is polynomial in c. On the other hand, the probability to never yield c more positive answers than negative ones, within the budget, is exponentially small in c. Using a global budget instead of a fixed per-edge number of queries, allows us to dynamically allocate it based on need. We save up queries from realized edges that reach the threshold quickly and spend it on other edges that are slow to reach the threshold. On the other hand, the probability that a non-realized edge reaches c is exponentially small in c. As a result, if a non-realized edge exists in the tree, the algorithm will consume all of its budget on this edge before reaching the threshold. Using this approach, Algorithm 1 achieves the following guarantees.

▶ **Theorem 7.** *Algorithm 1 correctly classifies connected trees with probability $1 - \varepsilon$ and disconnected trees with probability $1 - \delta$, while performing at most $O\left(\frac{1}{\varepsilon} n \log \frac{1}{\delta}\right)$ queries.*

The proof is deferred to the full version. As a Corollary of Theorem 7, we get that, for constant probability guarantees, Algorithm 1 requires $O(n)$ queries.

▶ **Corollary 8.** *Algorithm 1 correctly classifies connected trees with probability $.99$ and disconnected trees with probability $.99$, using $O(n)$ queries.*

4 Two-sided error oracle

Moving on to our main result, we show how to find a realized spanning tree in the 2-sided error oracle. Recall that in this case, the oracle may give false responses either when the edge is realized or when it is non-realized. The main result for this regime is the following theorem.

▶ **Theorem 9.** *In the 2-sided error regime, there exists an algorithm that finds a realized spanning tree with high probability, using $O(m \log m)$ queries in a moldgraph of m edges. Moreover, no algorithm can do better than $\Omega(m \log m)$.*

To show this theorem, we separately show the upper and lower bounds in Lemmas 10 and 12. Combining them we immediately get the theorem.

4.1 Upper Bound

In order to show the upper bound (Lemma 10), we describe an algorithm that achieves this query complexity and prove its correctness in Lemma 11

▶ **Lemma 10.** *In the 2-sided error regime, there exists an algorithm that finds a realized spanning tree with high probability, using $O(m \log m)$ queries in a moldgraph of m edges.*

Our algorithm, described below, uses the same idea as the naive algorithm for the connectivity verification problem.

Naive algorithm

The algorithm is as follows. First, query each edge of the moldgraph $O(\log m)$ times. Second, by treating all edges with more "Yes" than "No" responses as realized, compute a spanning tree on the discovered realized subgraph. If the graph is disconnected, output any spanning tree of the moldgraph.

The algorithm performs $O(m \log m)$ queries in total. The following lemma shows the correctness of the algorithm, and its proof is deferred to the full version.

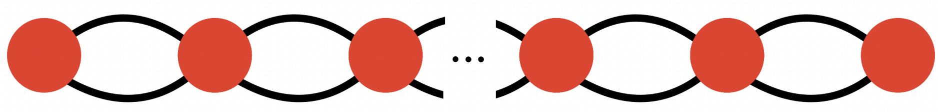
▶ **Lemma 11.** *The Naive algorithm finds a realized spanning tree with high probability.*

4.2 Lower Bound

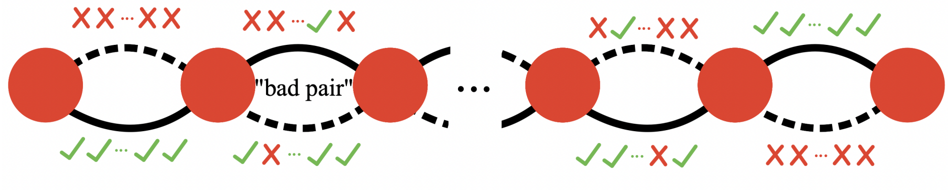
Switching gears towards the lower bound, we give an instance showing that we cannot hope for anything better than the $O(m \log m)$ upper bound shown in the previous section. As we expected, this is a strictly harder problem than verifying the connectivity of a tree, since in this case we have to identify a realized edge in each independent cut of the moldgraph.

▶ **Lemma 12.** *In the 2-sided error regime, there exists a graph where any algorithm requires $\Omega(m \log m)$ queries to discover a spanning tree with constant failure probability.*

Proof of Lemma 12. We consider the graph in Figure 1 which has $n + 1$ vertices and $2n$ edges. Between any two consecutive vertices there exist 2 parallel edges, one of which is realized and the other is not. Note that this moldgraph is a multigraph for simplicity of presentation. But the arguments hold analogously for a 2-by-n grid moldgraph, if we additionally give our algorithms the extra information that all vertical edges are realized. This only makes an algorithm more powerful as, in the worst-case, it can choose to ignore this information and execute independently. Then, we show that even with this extra knowledge, no algorithm can achieve a better query complexity than the desired bound.



(a) The moldgraph used for the proof of Lemma 12.



(b) A realized graph from the family mentioned in the proof of Lemma 12. The dotted lines are non-realized edges and by the edges lie the answers of the 2-sided noisy oracle.

Figure 1

Any algorithm on this graph will have to treat each edge-pair independently of the others, because uncovering a realized edge in one pair does not give any information on the solution to other pairs. Moreover, the optimal strategy for any algorithm is to query both edges in a pair an equal number of times and then pick the one with the majority of positive responses. Otherwise, picking the one with the least positive responses gives us smaller probability of success.

Let us suppose that the algorithm performs k queries on a specific edge-pair. If more than $k/2$ of them give false responses then the wrong edge will be picked as realized. Consequently, the algorithm will have to perform more than k queries on this edge-pair to discover the correct edge. We set $k = \Theta(\log(n/\log n)) = \Theta(\log n)$ and call such a faulty edge-pair a "bad" pair. Then, by the fact that false queries follow a binomial distribution, we get $\mathbf{Pr}\left[\text{bad}\right] > p^{k/2} > \log n/n$. Thus the probability that *at least* one such "bad pair" exists is

$$\mathbf{Pr}\left[\text{bad exists}\right] = 1 - \mathbf{Pr}\left[\text{all good}\right] \geq 1 - \left(1 - \frac{\log n}{n}\right)^n \geq 1 - \frac{1}{n}.$$

Assuming that there exists such a "bad pair" then the algorithm will have to perform $\Omega(\log n)$ queries on it. However, there is no way to know from the beginning which pair it will be. This means that any algorithm in this case will have to pick an α-fraction of edge-pairs on which to perform these many queries, for some (not necessarily constant) α. Define Q_b to be the event that the algorithm performs $\Omega(\log n)$ queries on the "bad pair". Then, $\mathbf{Pr}\left[Q_b\right] = \alpha$ and the probability of failure for the algorithm when a bad pair exists is

$$\mathbf{Pr}\left[\text{fail}|\text{bad exists}\right] = \alpha \mathbf{Pr}\left[\text{fail}|Q_b\right] + (1-\alpha) \geq (1-\alpha)$$

Thus, the total probability of failure to discover a spanning tree will be

$$\mathbf{Pr}\left[\text{fail}\right] \geq \mathbf{Pr}\left[\text{fail}|\text{bad exists}\right] \mathbf{Pr}\left[\text{bad exists}\right] \geq (1-\alpha)\left(1 - \frac{1}{n}\right)$$

If we want the failure probability to be constant then it must hold that $\alpha = 1 - o(1)$. Consequently, we perform $\Omega(\log n)$ queries on $\alpha n = \Theta(n)$ edges, giving us an $\Omega(n \log n)$ lower bound for this graph instance. The desired lower bound comes from the fact that any algorithm performing better than $\Omega(m \log m)$ queries would perform better than $\Omega(n \log n)$ in this instance, as it has $m = O(n)$. ◀

5 One-sided error oracle: False Negatives

Moving on to the one-sided error regime, we first consider the case of False Negative errors in the oracle. Our main contribution is an algorithm for the case of sparse and minor-closed graph families. We also show how we can obtain an algorithm for general graphs, that still performs well on ρ-sparse graphs, even when this property is unknown to us.

Finally, we show how to combine these results to obtain an algorithm for planar graphs that uses $O(m)$ queries to find a spanning tree, while no algorithm can do better (Corollary 17). This also implies better guarantees for other special families of graphs, as shown in Corollary 18.

▶ **Theorem 13.** *In the 1-sided False Negative regime, given a moldgraph of m edges and n nodes, there exists an algorithm that finds a realized spanning tree using $O(m \log n)$ queries in expectation. If the moldgraph belongs to a ρ-sparse and minor-closed family, the algorithm uses $O(\rho m)$ queries in expectation.*

Proof of Theorem 13. To create such an algorithm, we can run in parallel both the Naive Algorithm presented in the next subsection (Lemma 14) and Algorithm 3 for sparse minor-closed graph families. In particular, it suffices to let the two algorithms perform their queries alternately and independently of each other.

If the moldgraph belongs in a ρ-sparse and minor-closed family, then by Lemma 15, Algorithm 3 will perform $O(\rho m)$ queries in expectation before finding a spanning tree. For each one of these queries, we have just performed one extra query for the naive algorithm. Thus, eventually the combined algorithm will find the spanning tree in the same query complexity.

Otherwise, in general graphs, by using Lemma 14 for the Naive Algorithm and a similar argument as before, we get $O(m \log n)$ queries in expectation. ◀

5.1 Warm-up: naive algorithm

We start by describing the naive algorithm used for general graphs, that obtains the $O(m \log n)$ guarantee. The algorithm proceeds in rounds, where in each round it performs one query on all m edges of the moldgraph. It repeats this process until $n - 1$ realized edges forming a spanning tree are discovered. Note that in the case of FN queries we can be completely certain when a realized edge is discovered. The proof is deferred to the full version.

▶ **Lemma 14.** *The Naive Algorithm finds a spanning tree while performing $O(m \log n)$ queries on expectation.*

5.2 An Algorithm for ρ-sparse graphs

We present the algorithm used to obtain better guarantees for the family of ρ-sparse and minor closed graphs. The algorithm is presented in Algorithm 3 and the guarantees in Lemma 15.

▶ **Lemma 15.** *Given a moldgraph of m edges, belonging to a ρ-sparse and minor-closed family of graphs, Algorithm 3 performs $O(\rho m)$ queries in expectation and uncovers a realized spanning tree with probability* 1.

The basic primitive that our algorithm uses is a `DISCOVER` subroutine that cleverly orders queries in a way that it allows it to explore multiple edge sets in parallel. The subroutine is presented in more detail in Algorithm 2 and the Lemma that follows.

Algorithm 2 DISCOVER($\mathcal{S}$): Discovers an edge in a collection $\mathcal{S}$ of edge sets.

Input: Set $\mathcal{S} = \{E_1, E_2, \ldots, E_k\}$
Output: Realized edge e

while no edge is found **do**
  for $i = 1$ **to** k **do**
    Let e be the next edge in the cyclic order of E_i
    $q \leftarrow$ Query edge e
    if q is "Yes" **then**
      return e
    end if
  end for
end while

Algorithm 3 SolveSparseFN(G): Solves the problem in sparse moldgraphs using an 1-sided error oracle with False Negatives.

Input: Graph G
Output: A realized spanning tree

if G is a single node **then**
  return $\emptyset$
end if
$u = \text{argmin}\{\deg(v)\}$
$e = \text{DISCOVER}(N(u))$ // Let G' be the resulting graph after contracting edge e.
return SolveSparseFN(G') $\cup$ $\{e\}$

▶ **Lemma 16.** *Let $\mathcal{S}$ be a collection of k sets of edges. Assume as well that there exists at least one set in $\mathcal{S}$ containing some realized edge, then Algorithm 2 finds and returns such an edge using at most $2k|E_f|$ queries in expectation, where E_f is the set containing the found edge.*

The proof is deferred to the full version. We are now ready to prove the main result of the section, Lemma 15.

Proof of Lemma 15. We use induction on the number n of vertices of G to prove that Algorithm 3 performs at most $4\rho m$ queries in expectation. Throughout this proof we denote by m the total number of simple edges of the graph, and by m_s the number of super-edges.

The base case of induction for $n = 1$ holds trivially, as the algorithm needs no queries. Now assume that it holds for all graphs of at most $n - 1$ vertices. We will prove that it also holds for graphs of n vertices.

By using the sparsity property that $m_s \leq \rho n$, we can see that there always exists at least one vertex u with degree $\deg(u) \leq 2\rho$. Otherwise, the number of super-edges would be

$$m_s = \frac{1}{2} \sum_{v \in V} \deg(v) > \rho n$$

which leads to the graph not being ρ-sparse, a contradiction.

The algorithm uses Algorithm 2 as a basic subroutine, passing as input the neighborhood $N(u)$ of the least-degree vertex u. This is allowed because $N(u)$ is a cut of the moldgraph and thus satisfies the assumption of Lemma 16 that at least one edge-set contains a realized edge.

The `DISCOVER` subroutine will find a realized edge inside a super-edge E_f, spending in expectation at most $2 \cdot \deg(u)|E_f| \leq 4\rho|E_f|$ queries.

After contracting the super-edge E_f that contains the realized edge found by the subroutine, the algorithm will continue on the contracted graph G' that has $n-1$ vertices. It is important to note here that the graph G' will still be ρ-sparse, because of our assumption that graphs belong in a minor-closed family. By the induction hypothesis, the algorithm will find a realized spanning tree of G', by performing at most $4\rho\ (m - |E_f|)$ queries in expectation. Therefore, the total number of queries performed will be at most

$$4\rho|E_f| + 4\rho\ (m - |E_f|) = 4\rho m$$

in expectation, and thus gives us the desired $O(\rho m)$ upper bound. ◀

▶ **Corollary 17.** *If the moldgraph is planar, then Algorithm 3 performs $O(m)$ queries in expectation and succeeds in finding a realized spanning tree with probability 1. Moreover, this is tight and no better algorithm exists for this case.*

Proof of Corollary 17. The upper bound follows directly by Lemma 15 and the fact that planar graphs are a minor-closed and 3-sparse family, as $m \leq 3n - 6$.

To argue that the algorithm is tight, we remind that in all cases the realized spanning subgraph is chosen arbitrarily, and even in an adversarial way. This means that any algorithm aiming to find a realized spanning tree has to perform at least $n-1$ queries, for the $n-1$ realized edges of the spanning tree that it discovers. As planar graphs have $m = \Theta(n)$ the desired lower bound follows. ◀

Apart from planar graphs, as we hinted when defining ρ-sparsity, an efficient algorithm for ρ-sparse graphs immediately gives us efficient algorithms for minor-closed families of graphs that are sparse with respect to other widely-used parameters. Based on [13, 8], we can see that the parameters of treewidth and Hadwiger Number define minor-closed graph families. Moreover, k-treewidth graphs have sparsity k, while graphs with Hadwiger number k are $O(k\sqrt{\log k})$-sparse. Hence, applying Lemma 15 we get the following Corollary.

▶ **Corollary 18.** *Algorithm 3 performs $O(km)$ queries for graphs with treewidth k and $O(k\sqrt{\log k}m)$ queries for graphs with Hadwiger number k.*

6 One-sided error oracle: False Positives

In the 1-sided, False-Positive error case, we initially present an $O(m \log m)$ algorithm for general graphs, while dropping the complexity to linear when the realized graph is a tree. Each of these guarantees are tight for their respective case. We also show how using one algorithm, we can capture both cases, without actually knowing the structure of the realized graph, and obtain the tight guarantees for each case. Our formal guarantees are formally stated in Theorem 19.

▶ **Theorem 19.** *In the 1-sided error FP regime, there exists an algorithm that performs $O(m \log m)$ queries in the general case and finds a realized spanning tree with high probability. Moreover, if the realized subgraph is a tree, the algorithm performs $O(m)$ queries in expectation and succeeds to find the spanning tree with high probability. These complexities are tight for their respective cases.*

Proof of Theorem 19. Similarly to the case of False Negative oracle errors (Lemma 13) we can combine the Naive Algortihm with Algorithm 4, letting them perform queries alternately. When any of the two algorithms halts, the process ends.

Consider the case of general graphs. By Lemma 20 the naive algorithm succeeds in finding a realized spanning tree with high probability by performing $O(m \log m)$ queries. Thus, the combined algorithm also succeeds with at most $O(m \log m)$ queries.

Now, we look at the special case where the moldgraph is planar and the realized subgraph is a tree. By Lemma 23, Algorithm 4 performs at most $O(m)$ queries in expectation. Let X be the random variable denoting the number of queries performed by Algorithm 4, and let $E = \{X > m \log_2(m^2)\}$ be the event that X is more than the queries of the naive algorithm. Using Markov's inequality we get

$$\mathbf{Pr}\left[E\right] \leq \frac{\mathbb{E}\left[X\right]}{m \log_2(m^2)} \leq O\left(\frac{1}{\log m}\right)$$

Thus, the number Q of queries performed before the combined algorithm terminates for this case is

$$\mathbb{E}\left[Q\right] = \mathbb{E}\left[Q|\neg E\right] \mathbf{Pr}\left[\neg E\right] + \mathbb{E}\left[Q|E\right] \mathbf{Pr}\left[E\right] \leq \mathbb{E}\left[X\right] + (m \log_2(m^2)) O\left(\frac{1}{\log m}\right) \leq O(m)$$

When the algorithm has terminated, there still exists some probability of failure, in the case that both event E happens and the naive algorithm fails to find a realized spanning tree. By using the failure probability from Lemma 20 and the previously calculated probability of event E happening, we get

$$\mathbf{Pr}\left[E \wedge \text{naive algo fails}\right] = O\left(\frac{1}{m \log m}\right)$$

Thus, the combined algorithm succeeds in this case with high probability as well, and the proof is concluded. ◀

6.1 Upper & Lower Bounds

We start by showing, similar to the previous section, how to obtain the upper bound in the case of general graphs. The proof is deferred to the full version.

▶ **Lemma 20.** *In the 1-sided error FP regime, there exists an algorithm that performs $O(m \log m)$ queries in the general case and finds a realized spanning tree with high probability.*

Furthermore, we show the lower bound for the general case graphs, stated formally below. Observe that from the construction of this lower bound, no algorithm can do better in planar or sparse graphs. The proof is deferred to the full version.

▶ **Lemma 21.** *In the regime of 1-sided FP errors, there exists a realized graph with cycles, for which any algorithm requires $\Omega(m \log m)$ queries to discover a spanning tree with constant failure probability.*

6.2 Special Case: Acyclic Realized Graphs

In this section we focus on the special case where the realized subgraph is a tree.This property implies that for each cycle in the moldgraph, at least one edge of the cycle must be non-realized. Since our oracle only commits False Positive errors, the negative answers

Algorithm 4 SolvePlanarFP(G): Discovers a realized spanning tree in planar moldgraphs using an 1-sided error oracle with False Positives.

Input: Planar Graph $G(V, E)$
Output: A realized spanning tree

```
// Construct the dual graph of G
G' ← dual(G)
// Inverse all the oracle answers and use SolveSparseFN
return E(G) \ SolveSparseFN(G')
```

---

are definitive, meaning that we can completely remove the edges which have yielded such answers. We design an algorithm (Algorithm 4) that utilizes this fact and progressively removes non-realized edges until it is left with a tree which will also be the realized subgraph. This result is stated formally in the theorem below.

▶ **Theorem 22.** *In the regime of 1-sided FP errors, if the moldgraph G is a planar graph and the realized subgraph is a tree, Algorithm 4 recovers the realized spanning tree with probability* 1 *while performing at most $O(m)$ queries in expectation.*

Our main observation is that the FP setting is very similar, and in fact can be reduced to, the FN setting. In the FN setting, we know that at least one realized edge lies in every cut of the moldgraph and we repeatedly query the edges of a cut in order to unveil the realized edge. In a similar manner, in the FP setting, we can repeatedly query the edges of the cycle in order to discover the non-realized edge in every cycle. Hence, the question that arises is: how can one decide a sequence of cycles to query such that the expected number of queries is minimized? Algorithm 4 does so by utilizing the planarity of the moldgraph to consistently find small cycles to query.

Description of Algorithm 4

Our algorithm reduces the FP setting of this special case to solving an FN instance on the *dual graph* of the moldgraph. The dual of a planar graph has one node for each face of the planar graph, as illustrated in a Figure found in the full version. Two face nodes are connected by an edge in the dual graph if and only if they share an edge in the planar graph. With this transformation we can now focus on cuts of the dual graph which directly correspond to cycles of the moldgraph. Notice that removing an edge in the moldgraph (after identifying it as non-realized) is the same as contracting it in the dual graph. As a result, removing all the non-realized edges until we are left with a tree in the moldgraph is exactly equivalent to contracting the same edges in the dual until we are left with one node (Lemma 23). For this reason, Algorithm 4 constructs the dual graph of the moldgraph and calls Algorithm 3 to retrieve a non-realized spanning tree of the dual graph. Note that we reverse all the oracle's answers before we hand them out to Algorithm 3, since by design it contracts the edges for which we get positive answers but in the FP setting we want to contract the edges of the dual graph whose corresponding edges yielded negative answers.

We will now prove the correctness of Algorithm 4 and bound its query complexity.

▶ **Lemma 23.** *Let G be a planar moldgraph, T be the realized tree of G and G' be a dual graph of G. The induced subgraph of G' containing all the edges corresponding to non-realized edges of G, is a spanning tree of G'.*

The proof of Lemma 23 is deferred to the full version.

Proof of Theorem 22. Algorithm 4 constructs the dual graph of G and uses Algorithm 3 to find a spanning tree of G' consisting of non-realized edges. From Lemma 23 we know that by removing these edges, we are left with the realized tree T of G. All the queries performed during this process are made through Algorithm 3. Therefore, from Lemma 15, the total number of queries conducted is $O(m)$. ◀

References

1 Aditya Bhaskara, Sreenivas Gollapudi, Kostas Kollias, and Kamesh Munagala. Adaptive probing policies for shortest path routing. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL: `https://proceedings.neurips.cc/paper/2020/hash/62da5a6d47be0029801ba74a17e47e1a-Abstract.html`.

2 Béla Bollobás, Paul A Catlin, and Paul Erdös. Hadwiger's conjecture is true for almost every graph. *Eur. J. Comb.*, 1(3):195–199, 1980.

3 Thomas Erlebach and Michael Hoffmann. Minimum spanning tree verification under uncertainty. In Dieter Kratsch and Ioan Todinca, editors, *Graph-Theoretic Concepts in Computer Science – 40th International Workshop, WG 2014, Nouan-le-Fuzelier, France, June 25-27, 2014. Revised Selected Papers*, volume 8747 of *Lecture Notes in Computer Science*, pages 164–175. Springer, 2014. `doi:10.1007/978-3-319-12340-0_14`.

4 Uriel Feige, Prabhakar Raghavan, David Peleg, and Eli Upfal. Computing with noisy information. *SIAM Journal on Computing*, 23(5):1001–1018, 1994. `doi:10.1137/S0097539791195877`.

5 Luoyi Fu, Xinzhe Fu, Zhiying Xu, Qianyang Peng, Xinbing Wang, and Songwu Lu. Determining source-destination connectivity in uncertain networks: Modeling and solutions. *IEEE/ACM Trans. Netw.*, 25(6):3237–3252, 2017. `doi:10.1109/TNET.2017.2725905`.

6 Luoyi Fu, Xinbing Wang, and P. R. Kumar. Optimal determination of source-destination connectivity in random graphs. In Jie Wu, Xiuzhen Cheng, Xiang-Yang Li, and Saswati Sarkar, editors, *The Fifteenth ACM International Symposium on Mobile Ad Hoc Networking and Computing, MobiHoc'14, Philadelphia, PA, USA, August 11-14, 2014*, pages 205–214. ACM, 2014. `doi:10.1145/2632951.2632990`.

7 Michael Hoffmann, Thomas Erlebach, Danny Krizanc, Matús Mihalák, and Rajeev Raman. Computing minimum spanning trees with uncertainty. In Susanne Albers and Pascal Weil, editors, *STACS 2008, 25th Annual Symposium on Theoretical Aspects of Computer Science, Bordeaux, France, February 21-23, 2008, Proceedings*, volume 1 of *LIPIcs*, pages 277–288. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Germany, 2008. `doi:10.4230/LIPIcs.STACS.2008.1358`.

8 Alexandr V. Kostochka. Lower bound of the hadwiger number of graphs by their average degree. *Combinatorica*, 4(4):307–316, 1984.

9 Don R. Lick and Arthur T. White. k-degenerate graphs. *Canadian Journal of Mathematics*, 22(5):1082–1096, 1970. `doi:10.4153/CJM-1970-125-1`.

10 László Lovász. Graph minor theory. *Bulletin of the American Mathematical Society*, 43(1):75–86, 2006.

11 Jianhua Lyu, Yiran Ren, Zeeshan Abbas, and Baili Zhang. Reliable route selection for wireless sensor networks with connection failure uncertainties. *Sensors*, 21(21):7254, 2021. `doi:10.3390/s21217254`.

12 Arya Mazumdar and Barna Saha. Clustering with noisy queries. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL: `https://proceedings.neurips.cc/paper/2017/file/db5cea26ca37aa09e5365f3e7f5dd9eb-Paper.pdf`.

13 Neil Robertson and Paul D Seymour. Graph minors. III. planar tree-width. *Journal of Combinatorial Theory, Series B*, 36(1):49–64, 1984.

14 Frank R. Schmidt, Eno Töppe, and Daniel Cremers. Efficient planar graph cuts with applications in computer vision. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*, pages 351–356. IEEE Computer Society, 2009. doi:10.1109/CVPR.2009.5206863.

15 Julian Yarkony and Charless C. Fowlkes. Planar ultrametrics for image segmentation. In Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 64–72, 2015. URL: https://proceedings.neurips.cc/paper/2015/hash/3416a75f4cea9109507cacd8e2f2aefc-Abstract.html.