Learning and Verifying Graphs using Queries with a Focus on Edge Counting

Lev Reyzin[*] and Nikhil Srivastava[**]

Department of Computer Science
Yale University, New Haven, CT 06520, USA
{lev.reyzin,nikhil.srivastava}@yale.edu

Abstract. We consider the problem of learning and verifying hidden graphs and their properties given query access to the graphs. We analyze various queries (edge detection, edge counting, shortest path), but we focus mainly on edge counting queries. We give an algorithm for learning graph partitions using $O(n \log n)$ edge counting queries. We introduce a problem that has not been considered: verifying graphs with edge counting queries, and give a randomized algorithm with error ϵ for graph verification using $O(\log(1/\epsilon))$ edge counting queries. We examine the current state of the art and add some original results for edge detection and shortest path queries to give a more complete picture of the relative power of these queries to learn various graph classes. Finally, we relate our work to Freivalds' 'fingerprinting technique' – a probabilistic method for verifying that two matrices are equal by multiplying them by random vectors.

1 Introduction

Graph learning appears in many different contexts. Suppose we are presented with a circuit containing a set of chips on a board. We can test the resistance between two chips with an ammeter. In as few measurements as possible, we want to learn whether the entire circuit is connected, or whether we need to power the components separately. This can be seen as a graph learning problem, in which the chips are vertices of a hidden graph and the ammeter measurements are queries into the graph, which tell whether a pair of vertices is connected by a path. If we are given a strong enough ammeter to tell not only whether two chips are connected, but also how far apart they are in the underlying circuit, we get the stronger 'shortest path' queries.

In a different setting [3], testing which pairs of chemicals react in a solution is modeled by 'edge detection' queries. Here, vertices correspond to chemicals, edges designate chemical reactions, and a set of chemicals 'reacts' iff it induces an edge. Applications of this model extend to bioinformatics, where learning a

---

hidden matching [2] turns out to be useful in DNA sequencing. With each setup we have different tools and target concepts to learn.

Our goal is to explore several graph-learning problems and queries. We consider the following types of queries, defined on graphs $G = (V, E)$:

- **Edge detection query (ED)**: Check if there is edge between any two vertices in $S \subseteq V$. *This model has applications in genome sequencing and was studied in [1–4, 10].*
- **Edge counting query (EC)**: Return the number of edges in the subgraph induced by $S \subseteq V$. *This has extensive uses in bioinformatics and was studied in [6, 11].*
- **Shortest Path query (SP)**: Return the length of shortest path in G between two vertices; if no path exists, return ∞. *This is the canonical model in the evolutionary tree literature; see [12–14].*

The second kind of task we consider is graph verification. Suppose we are interested in learning the structure of some protein networks, and after months of careful measurement, we complete our learning task. If we then find out there is a small chance we made a mistake in our measurements or if we have reason to believe our equipment may have been broken during experimentation, can we verify the structures we've learned more efficiently than learning them over again? More concretely, we are interested in how efficiently can we decide whether a graph presented to us is indeed the "true graph." This is a natural question to ask, especially since real world data is often noisy, or we sometimes have reason to mistrust results we are given. Every learning problem induces a new verification problem.

We consider different classes of graphs for our learning and verification tasks. The first class is **arbitrary graphs**, where there are no restrictions on the topology of the graph. Any algorithm that learns or verifies an arbitrary graph can also be used for more restricted settings. We also consider learning **trees**, where we know the graph we are trying to learn is a tree, but we are not aware of its topology. This is a natural setting for learning structures that we know do not have underlying cycles, for example evolutionary trees. Finally, we consider the problem of learning the **partition** of a graph into connected components. Here, we do not restrict the underlying class of graphs, but instead relax the learning problem. This is a natural question in settings where different partitions represent qualitative differences, for example in electrical networks, a power generator in one partition cannot power any nodes outside its own partition. Note that this also subsumes the natural question of whether or not a graph is connected.

In this paper we fill in some gaps in the literature on these problems and introduce the verification task for these queries. We also introduce the problem of learning partitions and present results in the **EC** query case. We then show what problems remain open. After presenting a summary of the past work done on these problems, we divide our results into two sections: Graph Learning and Graph Verification.

2   Previous Work

In one of the earliest works in graph discovery, Hein [12] tackles the problem of learning a degree d restricted tree with **SP** queries. He describes an $O(dn \lg n)$ algorithm that builds the tree by inserting one node at a time, in a carefully chosen order under which each insertion takes $O(d \lg n)$ queries. Among other results, King et al. [13] provide a matching lower bound by showing that solving this problem requires solving multiple partition problems whose difficulty they then analyze.

Angluin and Chen [3] show that $O(\lg n)$ adaptive **ED** queries per edge are sufficient to learn an arbitrary hidden graph. Their algorithm repeatedly divides the graph into independent subgraphs (i.e., it colors the graph), so as to eliminate interference to **ED** queries from previously discovered edges, and uses a variant of binary search to find new edges within each subgraph. It is worth noting that this is not far from an information-theoretic lower bound of $\Omega(\epsilon \lg n)$ **ED** queries per edge for the family of graphs with $n^{2-\epsilon}$ edges. A later paper [4] generalizes these results to hypergraphs using different techniques.

The work of Angluin and Chen is preceded by a few papers $[1, 2, 10]$ that tackle learning restricted families of graphs, such as stars, cliques, and matchings. Alon et al. [2] provide lower and upper bounds of $.32\binom{n}{2}$ and $(1/2 + o(1))\binom{n}{2}$ respectively on learning a matching using nonadaptive **ED** queries, and a tight bound of $\Theta(n \lg n)$ **ED** queries in expectation if randomization is allowed. Alon and Asodi [1] prove similar bounds for the classes of stars and cliques. Grebinski and Kucherov [10] study reconstructing Hamiltonian paths with **ED** queries. It turns out that many of these results are subsumed by those of [3] if we ignore constant factors.

Grebinski and Kucherov [11] also study the problem of learning a graph using **EC** queries and give tight bounds of $\Theta(dn)$ and $\Theta(n^2 / \lg n)$ nonadaptive queries for d-degree-bounded and general graphs respectively. They also prove tight $\Theta(n)$ bounds for learning trees. Their constructions make heavy use of separating matrices. In [6], Grebinski and Kucherov present a survey on learning various restricted cases of graphs, including Hamiltonian cycles, matchings, stars, and $k-$degenerate graphs, with **ED** and **EC** queries.

In the graph verification setting, Beerliova et al. [5] consider the problem of discovering and verifying networks using distance queries. In this setting that models discovering nodes on the internet, the learner can query a vertex, and the answer to the query is the set of all edges whose endpoints have different graph-theoretic distance from the query vertex. They show there is no $o(\log n)$ competitive algorithm unless $P = NP$.

Both the learning and verification tasks also bear some relation to the field of Property Testing, where the object is to examine small parts of the adjacency matrix of a graph to determine a global property of the graph. For a survey of this area, see [9].

3 Graph Learning

We first note that **EC** queries are at least as strong than **ED** queries and that the problem of learning an arbitrary graph is at least as hard as learning trees or partitions. Hence, in this paper, any lower bounds for stronger queries and easier targets apply to weaker queries and harder target classes. Conversely, any upper bounds we establish for weaker queries and harder problems apply for stronger queries and more restricted classes.

We first establish that $\Theta(n^2)$ **SP** and **ED** queries is essentially tight for learning arbitrary graphs and partitions.

Proposition 1 $\Omega(n^2)$ **SP** *queries are needed to learn the **partition** of a hidden graph on n vertices.*

Proof. We prove this by an adversarial argument; the adversary simply answers '∞' (i.e., not connected) for all pairs of vertices i, j. If fewer than $\binom{n}{2}$ queries are made, then some pair i, j is not queried, and the algorithm cannot differentiate between the graph with no edges and the graph with a single edge $\{i, j\}$ (for which $\mathbf{SP}(i, j) = 1$). But these graphs have different partitions. $\square$

If k is the number of components in a graph, there is an obvious algorithm that does better for $k < n$, even without knowledge of k:

Proposition 2 $O(nk)$ **SP** *queries are sufficient to determine the **partition** of a hidden graph on n vertices, if k is the number of components in the graph.*

Proof. We use a simple iterative algorithm:

- Step 1: Place 1 in its own component.[1]
- Step $i > 1$: Query $\mathbf{SP}(i, w)$ for an item w from each existing component; if $\mathbf{SP}(i, w) \neq \infty$, place i in the corresponding component and move to the next step. Otherwise, create a new component containing i and move to the next step.

Correctness is trivial. For complexity, note that there at most k components at any step (since there are at most k components at phase n and components are never destroyed); hence n vertices take at most nk queries. $\square$

Proposition 3 $\Omega(n^2)$ **ED** *queries are needed to learn the **partition** of a hidden graph on n vertices.*

Proof. Consider the class of graphs on n vertices consisting of two copies of $K_{\frac{n}{2}}$, which we will call C_1 and C_2, and one possible edge between C_1 and C_2. If there is an edge, all the vertices are in a single component; otherwise there are two components. Any algorithm that learns the partition must distinguish

---

[1] we use numbers $1, 2, \ldots, n$ to represent the vertices of the graph

between the two cases. Observe that an **ED** query on a set S containing more than one vertex from either C_1 or C_2 will not yield any information since an edge is guaranteed to be present in S and any such query will be answered with a 'yes'. Hence, all informative queries must contain one vertex from C_1 and one vertex from C_2. An adversary can keep on answering 'no' to all such queries, and unless all possible pairs are checked, an edge may be present between C_1 and C_2. Hence, the algorithm cannot tell whether the graph has one component or two until it asks all $\approx (\frac{n}{2})^2 = \Omega(n^2)$ queries. $\qquad\square$

It turns out that **EC** queries are considerably more powerful than **ED** queries for this problem.

Proposition 4 $\Omega(n)$ **EC** *queries are needed to learn the **partition** of a hidden graph on n vertices.*

Proof. We use an information-theoretic argument. The number of partitions of an n element set is given by the Bell number B_n; according to de Bruijn [7]:

$$\ln B_n = \Omega(n \ln n)$$

Since each **EC** query gives a $\lg(\binom{n}{2}) = 2\lg n$ bit answer, we need $\Omega(\frac{\lg(B_n)}{2\lg n}) = \Omega(\frac{n\lg n}{\lg n}) = \Omega(n)$ queries. $\qquad\square$

Theorem 5 $O(n \lg n)$ **EC** *queries are sufficient to learn the **partition** of a hidden graph on n vertices.*

Proof. Consider the following $n-$phase algorithm, in which the components of $G[1\ldots i]$ are determined in phase i.

- *Phase 1*: Set $\mathcal{C} = \{c_1\}$ with $c_1 = \{1\}$. $\mathcal{C}$ will keep track of the components $c_1, c_2, \ldots$ known at any phase, and we will let $\mathcal{C} + v$ denote $\{v\} \cup \bigcup_{c_i \in \mathcal{C}} c_i$.
- *Phase $(i+1)$*: Let $v = (i+1)$, and query $\mathbf{EC}(\mathcal{C} + v)$. If $\mathbf{EC}(\mathcal{C} + v) = \mathbf{EC}(\mathcal{C})$ (i.e., there are no edges between v and $\mathcal{C}$), add a new component $c = \{v\}$ to $\mathcal{C}$.
  Otherwise, split $\mathcal{C}$ into roughly equal halves $\mathcal{C}_1$ and $\mathcal{C}_2$ and query $\mathbf{EC}(\mathcal{C}_1 + v), \mathbf{EC}(\mathcal{C}_2 + v)$. Pick any half $h \in \{1, 2\}$ for which $\mathbf{EC}(\mathcal{C}_h + v) > \mathbf{EC}(\mathcal{C}_h)$ and repeat recursively until $\mathbf{EC}(\{c_j\} + v) > \mathbf{EC}(c_j)$ for a single component $c_j \in \mathcal{C}^2$. This implies that there are edges between c_j and v; we will call c_j a *live* component.
  Repeat on $\mathcal{C} \setminus \{c_j\}$ to find another live component $c_{j'}$, if it exists; repeat again on $\mathcal{C} \setminus \{c_j, c_{j'}\}$ and so on until no further live components remain (or equivalently, no new edges are found). Remove all live components from $\mathcal{C}$ and add a new component $\{v\} \cup \bigcup_{\text{live } c_j} c_j$.

---

[2] Notice that this is essentially a binary search.

Correctness is simple, by induction on the phase: we claim that $\mathcal{C}$ contains the components of $G[1 \ldots i]$ at the end of phase i. This is trivial for $i = 1$. For $i > 1$, suppose $\mathcal{C} = \{c_1, \ldots, c_m\}$ at the beginning of phase i, and by the inductive hypothesis $\mathcal{C}$ contains precisely the components of $G[1 \ldots (i-1)]$. The components that do not have edges to v are unaffected by its introduction in $G[1 \ldots i]$, and these are not changed by the algorithm. All other components are connected to v and therefore to each other in $G[1 \ldots i]$; but these are marked 'live' and subsequently merged into a single component at the end of the phase. This completes the proof.

To analyze complexity, we use a "potential argument." Let Δ_i denote the increase in the number of components in $\mathcal{C}$ during phase i. There are three cases:

- $\Delta_i = 1$: There are no live components (v has no edges to any component in $\mathcal{C}$), and this is determined with a single $\mathbf{EC}(\mathcal{C} + v)$ query.
- $\Delta_i = 0$: There is exactly 1 live component (v connects to exactly one member of $\mathcal{C}$). Since there are at most n components to search, it takes $O(\lg n)$ queries to find this component.
- $\Delta_i < 0$: There are $k > 1$ live components with edges to v, bringing the number of components down by $k - 1$.[3] Finding each one takes $O(\lg n)$ queries, for a total of $O(k \lg n) = O((-\Delta_i + 1) \lg n)$.

The total number of queries is

$$\sum_{i:\Delta_i=1} 1 + \sum_{i:\Delta_i=0} (\lg n) + \sum_{i:\Delta_i<0} O((-\Delta_i + 1) \lg n)$$

The first two sums are bounded by $O(n \lg n)$ since there are n phases, and the last one becomes

$$O(n \lg n) + O(\lg n) \sum_{\Delta_i<0} (-\Delta_i).$$

But $\sum_{\Delta_i<0}(-\Delta_i)$, the total *decrease* in the number of components, cannot be greater than n since the total *increase* is bounded by n (one new component per phase) and the final number of components is nonnegative. So the total number of queries is $O(n \lg n)$, as desired.

To see that this analysis is tight, consider the case where G has exactly $n/2$ components, with $\Delta_i = 1$ for $i < n/2$, $\Delta_i = 0$ for $i \geq n/2$. The first $n/2$ phases take only $O(n/2)$ queries, but the remaining $n/2$ take $O(\lg(n/2))$ queries each, for a total of $O(n/2 \lg(n/2) + n/2) = O(n \lg n)$ queries. $\qquad\square$

Proposition 6 $O(|E| \lg n)$ $\mathbf{EC}$ *queries are sufficient to learn a hidden* **graph** *on n vertices.*

Proof. The algorithm of Angluin and Chen ([3]) achieves this since $\mathbf{EC}$ queries are more powerful than $\mathbf{ED}$ queries, but we present a simpler method here that

---

[3] The k components previously in $\mathcal{C}$ are replaced by a single component, hence $\Delta_i = -(k-1)$.

exploits the counting ability of **EC**. The key observation is that we can learn the degree of any vertex v in two queries:

$$d(v) = \mathbf{EC}(V) - \mathbf{EC}(V \setminus \{v\})$$

We use this to find all of the neighbors of v, using a binary search similar to that in the algorithm of theorem 5. Split $V \setminus \{v\}$ into halves V_1, V_2 and query $\mathbf{EC}(V_1 + v), \mathbf{EC}(V_2 + v)$. Pick a half such that $\mathbf{EC}(V_i + v) > \mathbf{EC}(V_i)$ and recurse until $\mathbf{EC}(w + v) > 0$ for some vertex w. This implies that w is a neighbor of v. Repeat the procedure on $V \setminus \{w, v\}$ to find more neighbors, and so on, until $d(v)$ neighbors are found.

We can reconstruct the graph by finding the neighbors of each vertex; this uses a total of

$$\sum_v d(v) \lg n = \lg n \sum_v d(v) = 2|E| \lg n = O(|E| \lg n)$$

queries, as desired. $\qquad\square$

It follows from the above proof that the degree sequence of a graph can be computed in $2n$ queries, and consequently any property that is determined by it takes only linear queries.

Proposition 7 $\Omega(n^2)$ **SP** *queries are needed to learn a hidden* ***tree***.

Proof. Consider a graph G on $2n + 1$ vertices, which are of three kinds: a single center vertex s, n 'inner' vertices $x_1 \ldots x_n$, and n 'outer' vertices $y_1 \ldots y_n$. The center and inner vertices form a star (with edges $\{x_i, s\}$) and the outer vertices are matched with the inner vertices (for each y_i there is a unique x_{j_i} such that $\{x_{j_i}, y_i\}$ is an edge; no x_{j_i} is repeated).

Suppose a learning algorithm knows that G is a quasi-star. There are only three kinds of **SP** queries: $\mathbf{SP}(s, x_i) = 1$, $\mathbf{SP}(s, y_i) = 2$, and

$$\mathbf{SP}(x_i, y_j) = \begin{cases} 1 \text{ if } \{x_i, y_j\} \text{ is an edge} \\ 3 \text{ otherwise} \end{cases}$$

The only query that gives any information is the last kind, and the problem reduces to that of learning a matching using **ED** queries, which we know by [2] takes $\Omega(n^2)$ queries. $\qquad\square$

Table 1 shows the known bounds for the problems we consider. We can see that tight asymptotic bounds exist for all of these learning problems, except for learning partitions with **EC**.

We note that learning a tree becomes significantly easier when the degrees of its vertices are restricted, and in many cases, knowing a bound on the degree of a graph can help with the learning problem.

Query	partition	graph	tree
ED	$\Theta(n^2)$	$\Theta(|E|\lg n), \Theta(n^2)$[3]	$\Theta(n\lg n)$
EC	$O(n\lg n)$	$O(|E|\lg n), O(\frac{n^2}{\lg n}), O(dn)$[3, 11]	$\Theta(n)$
	$\Omega(n)$	$\Omega(dn), \Omega(\frac{n^2}{\lg n})$[11]	
SP	$\Theta(nk)$	$\Theta(n^2)$	$\Theta(n^2), \Theta(dn\lg n)$ [12, 13]

Table 1. Summary of results. n denotes the number of vertices, $|E|$ the number of edges, d the degree restriction, and k the number of components

4 Graph Verification

In this setting, a verifier is presented a graph $G(V, E)$ and asked to check whether it is the same as a hidden graph $G^*(V, E^*)$, given query access to G^*. In this section, we explore the complexity of graph verification using various queries. Mainly, we show that while verifying unrestricted graphs is hard using **SP** and **ED** queries, there is a fast randomized algorithm that uses **EC** queries.

Proposition 8 *Verifying an arbitrary **graph** takes $\Theta(n^2)$ **SP** queries and $\Theta(n^2)$ **ED** queries.*

Proof. Consider the problem of verifying a clique, when the hidden graph is a clique with some edge (u, v) removed, and the verifier knows this. $\mathbf{SP}(u', v') = 2$ if and only if $u' = u$ and $v' = v$. A simple adversarial argument shows that $\Omega(n^2)$ queries are necessary. Similarly, for **ED** queries, let $S = \{u, v\}$. The answer to query $\mathbf{ED}(U)$, where $|U| \neq 2$ is predetermined. Otherwise, $\mathbf{ED}(U) = 0$ if and only if $U = S$. There are $\binom{n}{2}$ choices for S such that $|S| = 2$; hence $\Omega(n^2)$ are needed. For both **SP** and **ED** queries the $O(n^2)$ algorithm of checking all pairs of vertices is obvious. $\square$

Given that **SP** queries are most often considered in evolutionary tree learning, we also consider the problem of verifying a tree with **SP** queries. In this setting, the verifier knows the hidden graph is a tree and is presented with a tree to verify.

Proposition 9 *Verifying a **tree** takes $\Theta(n)$ **SP** queries.*

Proof. Consider the problem of verifying a path graph (from the class of path graphs). This reduces to verifying that a given ordering of the vertices is correct. If the answers to each query are consistent with the graph to be verified, each query verifies at most two vertices in the ordering. An adversary can choose whether or not to swap any pair of vertices that have not been queried and either stay consistent with the input path graph or not until at least $n/2$ **SP** queries have been performed. Conversely, we can verify each edge individually in $n - 1$ queries. $\square$

We now consider the problem of verifying a graph with **EC** queries. Here, we see that **EC** queries are quite powerful for verifying arbitrary graphs.

Theorem 10 *Any **graph** can be verified by a randomized algorithm using* 1 **EC** *query, with success probability* 1/4.

Proof. We define $\mathbf{EC}(V, G)$ to be the query $\mathbf{EC}(V)$ on graph G. The algorithm is simple. We let Q be a random subset of vertices of V, with each vertex chosen independently with probability $\frac{1}{2}$. We query $\mathbf{EC}(Q, G^*)$ and compute $\mathbf{EC}(Q, G)$. If the two quantities are not equal, we say G and G^* are different. Otherwise we say they are the same. We will show that if $G = G^*$ the algorithm always returns the correct answer, and otherwise gives the correct answer with probability at least $\frac{1}{4}$.

Consider the symmetric difference $S = (V, E \Delta E^*)$. Let $A = \{(u, v) \in E \backslash E^* : u, v \in Q\}$ and $B = \{(u, v) \in E^* \backslash E : u, v \in Q\}$. If $G = G^*$ then $|A| = |B| = 0$ and we are always right in saying the graphs are identical; otherwise $G \neq G^*$ and $E \Delta E^* \neq \varnothing$, so by the following lemma $|E \Delta E^*| = |A| + |B|$ is odd with probability $\frac{1}{4}$. But this immediately implies that $|A| \neq |B|$, as desired. $\qquad \square$

Lemma 11 *Let $G(V, E)$ be a graph with at least one edge. Let $G'(V', E')$ be the subgraph induced by taking each vertex in G independently with probability $\frac{1}{2}$. If G is non-empty, the probability that $|E'|$ is odd is at least $\frac{1}{4}$.*

Proof. Fix an ordering $v_1 \ldots v_n$ so that $(v_{n-1}, v_n) \in E$. Select each of $v_1 \ldots v_{n-2}$ independently with probability $1/2$, and let H' be the subgraph induced by the selected vertices. Suppose the probability that H' contains an odd number of edges (i.e., $\mathtt{parity}(H') = 1$) is p.

Let i (resp. j) be the number of edges between v_{n-1} and H' (resp. v_n and H'). Consider two cases:

- $i \equiv j \mod 2$ If both are chosen an odd number of edges is added to H' and $\mathtt{parity}(H') = 1 - \mathtt{parity}(G')$. This happens with probability $1/4$.
- $i \not\equiv j \mod 2$. Assume w.l.o.g. that i is odd and j is even. Then, if v_{n-1} is chosen and v_n is *not* chosen, an odd number of edges is added to H', and again $\mathtt{parity}(H') = 1 - \mathtt{parity}(G')$. This happens with probability $1/4$.

On the other hand, if neither v_{n-1} nor v_n is chosen then $\mathtt{parity}(G') = \mathtt{parity}(H')$, and this happens with probability $1/4$. So upon revealing the last two vertices, the parity of H' is flipped with probability at least $1/4$ and not flipped with probability at least $1/4$, independently of what happens in H'. Let F denote the event that it is flipped (i.e., that $\mathtt{parity}(H') \neq \mathtt{parity}(G')$). Then,

$$
\begin{aligned}
\mathbb{P}[\mathtt{parity}(G') = 1] &= \mathbb{P}[\mathtt{parity}(G') = 1 | \mathtt{parity}(H') = 1] \mathbb{P}[\mathtt{parity}(H') = 1] \\
&\quad + \mathbb{P}[\mathtt{parity}(G') = 1 | \mathtt{parity}(H') = 0] \mathbb{P}[\mathtt{parity}(H') = 0] \\
&= \mathbb{P}[\overline{F} | \mathtt{parity}(H') = 1] p + \mathbb{P}[F | \mathtt{parity}(H') = 0](1 - p) \\
&= \mathbb{P}[\overline{F}] p + \mathbb{P}[F](1 - p) \quad \text{by independence} \\
&\geq 1/4(p + 1 - p) = 1/4
\end{aligned}
$$

as desired. $\qquad \square$

This finishes the proof of Theorem 10. Since this result has 1-sided error, we can easily boost the $\frac{1}{4}$ probability to any constant, and Corollary 12 follows immediately.

Corollary 12 *Any graph can be verified by a randomized algorithm with error ϵ using $O(\log(\frac{1}{\epsilon}))$ **EC** queries.*

4.1 Relation to Fingerprinting

Suppose A and B are $n \times n$ matrices over a field $\mathbb{F}$. It is known that if $A \neq B$, then for a vector $v \in \{0,1\}^n$ chosen uniformly at random we have

$$\mathbb{P}[Av \neq Bv] \geq 1/2.$$

This is Freivalds' fingerprinting technique [8]. It is was originally developed as a technique for verifying matrix multiplications, and can be used for testing for equality of any two matrices.

An easy extension of this method says that for vectors $v, w \in \{0,1\}^n$ chosen independently uniformly at random, if $A \neq B$ we have

$$\begin{aligned}
\mathbb{P}[w^T Av \neq w^T Bv] &= \mathbb{P}[w^T Av \neq w^T Bv | Av = Bv]\mathbb{P}[Av = Bv] \\
&\quad + \mathbb{P}[w^T Av \neq w^T Bv | Av \neq Bv]\mathbb{P}[Av \neq Bv] \\
&\geq 0 \times \mathbb{P}[Av = Bv] + \frac{1}{2} \times \frac{1}{2} \\
&= \frac{1}{4}
\end{aligned}$$

This bears a strong resemblance to graph verification with **EC** queries. Let A and B be the incidence matrices of G and G^*, respectively. Then an **EC** query Q corresponds to multiplication on the left and right by the characteristic vector of Q, and the algorithm becomes: choose $v \in \{0,1\}^n$ uniformly at random and return 'same' iff $v^T Av = v^T Bv$. By Theorem 10 if $A \neq B$ then $Pr[v^T Av \neq v^T Bv] \geq \frac{1}{4}$.

This raises a natural question. For *arbitrary* $n \times n$ matrices A and B over a field, if $A \neq B$, then for a vector $v \in \{0,1\}^n$ chosen uniformly at random, is $\mathbb{P}[v^T Av \neq v^T Bv] \geq 1/4$ (or some other constant > 0)?

This turns out not to be the case. Consider the two matrices

$$A = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \qquad B = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

$A \neq B$, but it is not hard to check that for any vector $v \in \{0,1\}^n$, $v^T Av = v^T Bv$. In fact, this holds true for adjacency matrices of 'opposite' directed cycles on > 3 vertices. A graph theoretic interpretation of this fact is that if the number of directed edges on any induced subset of the two opposite directed cycles is the same, then an **EC** query will always return the same answer for the two

different cycles. Needless to say, this property is not limited to the adjacency matrices of directed cycles: in fact, it holds for any two matrices A and B such that $A + A^T = B + B^T$, since

$$v^T(A + A^T)v = v^T Av + v^T A^T v = v^T Av + (v^T Av)^T = 2v^T Av$$

for all v, so that $v^T Av = v^T Bv$ for all v.

Hence, we know that standard fingerprinting techniques do not imply Theorem 10. Furthermore, the proof to Theorem 10 generalizes easily to weighted graphs and a more general form of **EC** queries, where the answer to the query is the sum of the weights of its induced edges. Since any symmetric matrix can be viewed as an adjacency matrix of an undirected graph, we have the following fingerprinting technique for symmetric matrices.

Theorem 13 *Let A and B be $n \times n$ symmetric matrices over a field such that $A \neq B$,[4] then for v chosen uniformally at random from $v \in \{0,1\}^n$, $Pr[v^T Av \neq v^T Bv] \geq \frac{1}{4}$.*

Proof. Let $C = A - B \neq 0$, and note that $v^T Av \neq v^T Bv \iff v^T Cv \neq 0$. Identify C with the weighted graph $G = (V, E)$, where $V = \{v_1 \ldots v_n\}$ and $E = \{(u, v) : C(u, v) \neq 0\}$, and $\mathtt{wt}(u, v) = C(u, v)$. We proceed as in the proof of Lemma 11. Fix $v_1 \ldots v_n$ so that $\mathtt{wt}(v_{n-1}, v_n) \neq 0$, and let H' be as before. Define:

$$\mathtt{wt}(H) = \sum_{(u,v) \in H} \mathtt{wt}(u, v); \quad \mathtt{wt}(w, H) = \sum_{(w,v) \in G, v \in H} \mathtt{wt}(w, v).$$

The first quantity is a generalization of $\mathtt{parity}$, the second of the number of edges from a vertex to a subgraph. Let $T = \mathtt{wt}(v_{n-1}, H') + \mathtt{wt}(v_n, H') + \mathtt{wt}(v_{n-1}, v_n)$, and consider two cases:

- $T = 0$. Since $\mathtt{wt}(v_{n-1}, v_n) \neq 0$, we know that at least one of the other terms must be nonzero. Assume w.l.o.g. that this is $\mathtt{wt}(v_n, H')$. So choosing v_n but not v_{n-1} is will make $\mathtt{wt}(G') \neq \mathtt{wt}(H')$, and this happens with probability $1/4$.
- $T \neq 0$. Choosing both v_n and v_{n-1} sets $\mathtt{wt}(G') = \mathtt{wt}(H') + T \neq \mathtt{wt}(H')$. This happens with probability $1/4$.

Again, we choose *neither* vertex with probability $1/4$, in which case $\mathtt{wt}(G') = \mathtt{wt}(H')$. Finally,

---

[4] Or, more generally, any matrices A and B with $A + A^T \neq B + B^T$.

$$\begin{aligned}
\mathbb{P}[\mathtt{wt}(G') \neq 0] &= \mathbb{P}[\mathtt{wt}(G') \neq 0 | \mathtt{wt}(H') \neq 0]\mathbb{P}[\mathtt{wt}(H') \neq 0] \\
&\quad + \mathbb{P}[\mathtt{wt}(G') \neq 0 | \mathtt{wt}(H') = 0]\mathbb{P}[\mathtt{wt}(H') = 0] \\
&\geq \mathbb{P}[\mathtt{wt}(G') = \mathtt{wt}(H') | \mathtt{wt}(H') \neq 0]\mathbb{P}[\mathtt{wt}(H') \neq 0] \\
&\quad + \mathbb{P}[\mathtt{wt}(G') \neq \mathtt{wt}(H') | \mathtt{wt}(H') = 0]\mathbb{P}[\mathtt{wt}(H') = 0] \\
&= \mathbb{P}[\mathtt{wt}(G') = \mathtt{wt}(H')]\mathbb{P}[\mathtt{wt}(H') \neq 0] \\
&\quad + \mathbb{P}[\mathtt{wt}(G') \neq \mathtt{wt}(H')]\mathbb{P}[\mathtt{wt}(H') = 0] \qquad \text{by independence} \\
&\geq 1/4(\mathbb{P}[\mathtt{wt}(H') = 0] + \mathbb{P}[\mathtt{wt}(H') \neq 0]) = 1/4
\end{aligned}$$

as desired. $\qquad\square$

5 Discussion

There is a tantalizing asymptotic gap of $O(\lg n)$ in our bounds for **EC** queries for learning the partition of the graph. It would also be interesting to know under which, if any, query models it is easier to learn the number of components than the partition itself. There is also the open question whether for general graphs, the $O(|E| \lg n)$ bound can be improved to $O(E)$ for **EC** queries. This is the open question asked by Bouvel et. al. [6] on whether a hidden graph of *average* degree d can be learned with $O(dn)$ **EC** queries.[5]

Some other problems left to be considered are learning and verification problems for other restricted classes of graphs. For example, of theoretical interest is the problem of verifying trees with **ED** queries. There is an obvious $O(n)$ brute-force algorithm, but it may be possible to do better. Also, other classes of graphs have been studied in the literature (see the Section 2) including Hamiltonian paths, matchings, stars, and cliques. It may be revealing to see the power of the queries considered herein for learning and verifying these restricted classes of graphs.

It would also be useful to look at this problem from a more economic perspective. Since edge counting queries are strictly more powerful than edge detecting queries, they ought to be more expensive in some natural framework. Taking costs into account and allowing learners to be able to choose queries with the goal of both learning the graph and minimizing cost should be an interesting research direction.

Finally, our work shows that graph verification is possible even for many classes of directed graphs. It would be interesting to redefine these queries for directed graphs and explore their power.

---

[5] [6] restrict themselves to a non-adaptive framework, where all queries must be asked simultaneously.

Acknowledgments

References

1. ALON, N., AND ASODI, V. Learning a hidden subgraph. *SIAM J. Discrete Math. 18*, 4 (2005), 697–712.
2. ALON, N., BEIGEL, R., KASIF, S., RUDICH, S., AND SUDAKOV, B. Learning a hidden matching. *SIAM J. Comput. 33*, 2 (2004), 487–501.
3. ANGLUIN, D., AND CHEN, J. Learning a hidden graph using O(log n) queries per edge. In *COLT* (2004), pp. 210–223.
4. ANGLUIN, D., AND CHEN, J. Learning a hidden hypergraph. *Journal of Machine Learning Research 7* (2006), 2215–2236.
5. BEERLIOVA, Z., EBERHARD, F., ERLEBACH, T., HALL, A., HOFFMANN, M., MI-HALÁK, M., AND RAM, L. S. Network discovery and verification. In *WG* (2005), pp. 127–138.
6. BOUVEL, M., GREBINSKI, V., AND KUCHEROV, G. Combinatorial search on graphs motivated by bioinformatics applications: A brief survey. In *WG* (2005), pp. 16–27.
7. DE BRUIJN, N. G. *Asymptotic Methods in Analysis*. Dover, 1981.
8. FREIVALDS, R. Probabilistic machines can use less running time. In *IFIP Congress* (1977), pp. 839–842.
9. GOLDREICH, O., GOLDWASSER, S., AND RON, D. Property testing and its connection to learning and approximation. *J. ACM 45*, 4 (1998), 653–750.
10. GREBINSKI, V., AND KUCHEROV, G. Reconstructing a hamiltonian cycle by querying the graph: Application to dna physical mapping. *Discrete Applied Mathematics 88*, 1-3 (1998), 147–165.
11. GREBINSKI, V., AND KUCHEROV, G. Optimal reconstruction of graphs under the additive model. *Algorithmica 28*, 1 (2000), 104–124.
12. HEIN, J. J. An optimal algorithm to reconstruct trees from additive distance data. *Bulletin of Mathematical Biology 51*, 5 (1989), 597–603.
13. KING, V., ZHANG, L., AND ZHOU, Y. On the complexity of distance-based evolutionary tree reconstruction. In *SODA '03: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms* (Philadelphia, PA, USA, 2003), Society for Industrial and Applied Mathematics, pp. 444–453.
14. REYZIN, L., AND SRIVASTAVA, N. On the longest path algorithm for reconstructing trees from distance matrices. *Inf. Process. Lett. 101*, 3 (2007), 98–100.