Optimal Graph Reconstruction by Counting Connected Components in Induced Subgraphs

Hadley Black HABLACK@UCSD.EDU
Arya Mazumdar ARYA@UCSD.EDU
Barna Saha BARNAS@UCSD.EDU
Yinzhan Xu XYZHAN@UCSD.EDU
University of California, San Diego

Editors: Nika Haghtalab and Ankur Moitra

Abstract

The graph reconstruction problem has been extensively studied under various query models. In this paper, we propose a new query model regarding the number of connected components, which is one of the most basic and fundamental graph parameters. Formally, we consider the problem of reconstructing an n-node m-edge graph with oracle queries of the following form: provided with a subset of vertices, the oracle returns the number of connected components in the induced subgraph. We show $\Theta(\frac{m \log n}{\log m})$ queries in expectation are both sufficient and necessary to adaptively reconstruct the graph. In contrast, we show that $\Omega(n^2)$ non-adaptive queries are required, even when $m = O(n)$. We also provide an $O(m \log n + n \log^2 n)$ query algorithm using only two rounds of adaptivity.

Keywords: Graph reconstruction, query complexity, group testing

1. Introduction

Graph reconstruction is a classical problem that aims to learn a hidden graph by asking queries to an oracle. Over the last three decades, the problem has been extensively studied under various query models (e.g., Grebinski (1998); Grebinski and Kucherov (1998); Alon et al. (2004); Alon and Asodi (2005); Reyzin and Srivastava (2007); Angluin and Chen (2008); Choi and Kim (2010); Mazzawi (2010); Bshouty and Mazzawi (2011, 2012); Choi (2013); Bshouty and Mazzawi (2015); Konrad et al. (2025)). Given a graph $G = (V, E)$ with an unknown edge set E, an algorithm may submit queries to an oracle (what the oracle computes is problem-specific) in order to recover E. The main objective is to minimize the number of queries.

Among all types of queries, one of the simplest and oldest is the *edge-detection* model (also referred to as *independent set* (IS) queries), which was studied since Grebinski and Kucherov (1998). In this model, the algorithm may submit a set $S \subseteq V$ to the oracle, which returns whether the induced subgraph $G[S]$ contains at least one edge. At the beginning, this model was studied for special classes of graphs such as Hamiltonian cycles (Grebinski and Kucherov, 1998), matchings (Alon et al., 2004), stars, and cliques (Alon and Asodi, 2005). The problem was then studied on general graphs by Alon and Asodi (2005); Reyzin and Srivastava (2007) and the best query complexity known is $O(m \log n)$ by Angluin and Chen (2008), which is optimal when $m \leq n^{2-\varepsilon}$ for any $\varepsilon > 0$.

Several natural ways to strengthen the edge-detection query model have also been studied:

- *Additive (*ADD*) queries:* Given set $S \subseteq V$, the oracle returns the number of edges in $G[S]$ (Grebinski, 1998; Grebinski and Kucherov, 2000; Bshouty and Mazzawi, 2011; Choi and Kim, 2010; Mazzawi, 2010; Bshouty and Mazzawi, 2012; Choi, 2013; Bshouty and Mazzawi, 2015).
- *Cross-additive (*CUT*) queries:* Given disjoint sets $S, T \subseteq V$, the oracle returns the number of edges between S and T (Choi and Kim, 2010; Choi, 2013).
- *Maximal independent set (*MIS*) queries:* Given set $S \subseteq V$, the oracle returns an adversarially chosen maximal independent set in $G[S]$ (Konrad et al., 2025).

The additive and cross-additive models have garnered significant attention and are known to have query complexity $\Theta(\frac{m \log(n^2/m)}{\log m})$ (Choi and Kim, 2010; Mazzawi, 2010). For weighted graphs, where instead of the number of edges, the oracle returns the total weights of these edges and the goal is to reconstruct all edges with their weights, the query complexity becomes $\Theta(\frac{m \log n}{\log m})$ (Bshouty and Mazzawi, 2011; Choi, 2013; Bshouty and Mazzawi, 2015). Interestingly, there exist non-adaptive algorithms that attain these query complexities in both models and for both unweighted and weighted graphs.[1] These models also have connections with classic algorithmic and combinatorial questions such as *group testing* and *coin-weighing*.

Maximal independent set queries were introduced very recently by Konrad et al. (2025), motivated by the existence of efficient algorithms for maximal independent set in various computation models such as the Congested-Clique model, the LOCAL model and the semi-streaming model. These efficient algorithms make maximal independent set queries a potential candidate to use as a building block in more complicated algorithms. Konrad et al. (2025) was also motivated to study whether MIS-queries are strictly more powerful than IS-queries. They obtained upper and lower bounds on the query complexity mainly in terms of the maximum degree of the graph.

Connected Component Count Queries. In this paper, we initiate the study of graph reconstruction using an oracle which returns the *number of connected components* in $G[S]$ (CC-queries), which is one of the most basic and fundamental graph parameters. This model is a natural way to strengthen IS-queries, as there exists an edge in $G[S]$ if and only if the number of connected components is strictly less than $|S|$.

Similar to maximal independent set, the number of connected components can also be computed efficiently in other computation models. For instance, Ghaffari and Parter (2016) showed how to compute the number of connected components in $O(\log^* n)$ rounds in the Congested-Clique model. An efficient $\tilde{O}(n)$-space streaming algorithm also trivially exists, by maintaining an arbitrary spanning forest. These efficient algorithms make it hopeful to use CC-queries as a basic building block in other algorithmic applications. It is worth noting that, estimating the number of connected components was also considered in the statistics literature (Frank, 1978; Bunge and Fitzpatrick, 1993).

An additional motivation for our study of graph reconstruction with CC queries is that it generalizes the problem of learning a partition using rank/subset queries, studied recently by Chakrabarty and Liao (2024) and Black et al. (2024). In particular, this problem corresponds to the special case when the hidden graph G is known to be a disjoint union of cliques and is itself a generalization of the well-studied problem of clustering using pair-wise same-set queries (e.g. Reyzin and Srivastava (2007); Balcan and Blum (2008); Mitzenmacher and Tsourakakis (2016); Ashtiani et al. (2016);

[1] An algorithm is called non-adaptive if all its queries are specified in one round, and adaptive otherwise.

Mazumdar and Saha (2017a,b,c); Saha and Subramanian (2019); Huleihel et al. (2019); Bressan et al. (2020); Liu and Mukherjee (2022); Del Pia et al. (2022); DePavia et al. (2024)).

1.1. Results

As our main result, we settle the query complexity of graph reconstruction with CC-queries, by showing that $\Theta(\frac{m \log n}{\log m})$ queries are both necessary and sufficient for any value of m. Our main result can be more formally described as the following two theorems:

Theorem 1 (Adaptive Algorithm) *There is an adaptive, randomized polynomial time algorithm that, given* CC*-query access to an underlying (unknown) n-node graph $G = (V, E)$, the vertex set V and an upper bound m on the number of edges, reconstructs G using $O(\frac{m \log n}{\log m})$* CC*-queries in expectation.*

Theorem 2 (Adaptive Lower Bound) *Any (randomized) adaptive graph reconstruction algorithm requires $\Omega(\frac{m \log n}{\log m})$* CC*-queries in expectation to achieve $1/2$ success probability for all $m \leq \binom{n}{2} - 1$ and even when m is provided as input.*

Compared with the $O(m \log n)$ bound for IS-queries (Angluin and Chen, 2008), which is tight when $m \leq n^{2-\varepsilon}$ for any $\varepsilon > 0$, our bound shows that CC-queries are indeed strictly stronger than IS-queries, at least when $\omega(1) < m < n^{2-\varepsilon}$.

The comparison between our bound and the bounds for (cross)-additive queries is more intriguing. Our bound coincides with the bound of (cross)-additive queries for *weighted graphs* (Bshouty and Mazzawi, 2011; Choi, 2013; Bshouty and Mazzawi, 2015); whereas for unweighted graphs, the $\Theta(\frac{m \log(n^2/m)}{\log m})$ bound achieved by Choi and Kim (2010); Mazzawi (2010) is smaller than our bound when m is very close to n^2. In particular, for dense graphs where $m = \Omega(n^2)$, $\Theta(n^2)$ CC-queries are required, while only $\Theta(n^2/\log n)$ (cross)-additive queries suffice for unweighted graphs.

Since our bound is similar to those of (cross)-additive queries, and there are non-adaptive algorithms that achieve the same bounds for (cross)-additive queries for both unweighted graphs (Choi and Kim, 2010) and for weighted graphs (Bshouty and Mazzawi, 2011, 2015), it is natural to ask whether it is possible to obtain the optimal query complexity for CC-queries using a *non-adaptive* algorithm. Perhaps surprisingly, we prove that this is not possible for CC queries in a strong sense: we show that non-adaptive algorithms require $\Omega(n^2)$ CC-queries even for *sparse* graphs, matching the trivial upper bound obtained by simply querying every pair of nodes. This establishes a stark contrast between CC queries and (cross)-additive queries.

Theorem 3 (Non-Adaptive Lower Bound) *Any non-adaptive graph reconstruction algorithm requires $\Omega(n^2)$* CC*-queries in expectation to achieve $1/2$ success probability, even when the graph is known to have $O(n)$ edges.*

Theorem 3 also implies $\Omega(n^2)$ IS-queries are required on sparse graphs for randomized algorithms, which is on par with the $\Omega(m^2 \log n)$ non-adaptive lower bound for IS-queries shown in Abasi and Bshouty (2019) for certain ranges of m.

In light of this strong lower bound for non-adaptive algorithms, we investigate whether efficient algorithms are possible using few rounds of adaptivity. We show there is an algorithm using only *two rounds* of adaptivity which comes within an $O(\log^2 n)$ factor of the optimal query complexity[2].

[2] In fact, when $m \geq \Omega(n \log n)$ our algorithm is within a factor of $O(\log n)$ of the optimal query complexity.

Theorem 4 (Two Round Algorithm) *There is a randomized algorithm using two rounds of adaptivity and $O(m \log n + n \log^2 n)$ CC-queries in the worst case which successfully reconstructs any arbitrary n-node graph with at most m edges with probability $1 - 1/\text{poly}(n)$. The upper bound m is provided as input to the algorithm.*

Abasi and Bshouty (2019) proved that with two rounds of adaptivity $m^{4/3-o(1)} \log n$ IS-queries are required, which again separates IS-queries with CC-queries.

Our two round algorithm utilizes an interesting connection with group testing and can be described at a high level as follows. The first round of queries is used to obtain a constant factor approximation of the degree of every vertex using $O(n \log^2 n)$ queries. Then, once an approximation of the degree d_v of a vertex v is known, a simple group testing procedure can be leveraged to learn its neighborhood with $O(d_v \log n)$ queries. The second round runs this procedure in parallel for every vertex for a total of $O(m \log n)$ queries.

Paper Overview. The main effort of our paper is in proving the correctness of our optimal adaptive algorithm, Theorem 1. Our proof uses multiple important subroutines which we describe in Section 2.1 and Section 3.2. Given these subroutines, the main proof is given in Section 3.3. A high level overview of the proof is given in Section 3.1. The proof of correctness for our two round algorithm (Theorem 4) is given in Section 4 and we prove our lower bounds (Theorem 2 and Theorem 3) in Section 5.

1.2. Other Related Works

Aside from the edge-detection, (cross-)additive, and maximal independent set already discussed, the graph reconstruction problem has also been studied extensively in the *distance* query model by Kranakis et al. (1995); Grebinski and Kucherov (1998); Beerliova et al. (2006); Erlebach et al. (2006); Reyzin and Srivastava (2007); Mathieu and Zhou (2013); Kannan et al. (2015); Kranakis et al. (2016); Kannan et al. (2018); Mathieu and Zhou (2023); Bastide and Groenland (2023). In this model, the algorithm may query a pair of vertices (x, y) and the oracle responds with the shortest path distance from x to y. This model has a notably different flavor from edge-detection and its related models. For bounded degree graphs, the current best known algorithm achieves query complexity $\widetilde{O}(n^{3/2})$, and it is a significant open question as to whether $\widetilde{O}(n)$ can be achieved.

Query algorithms have also received a great deal of attention recently for solving various (non-reconstruction) problems over graphs, for example testing connectivity, computing the minimum cut, and computing spanning forests (Harvey, 2008; Auza and Lee, 2021; Assadi et al., 2021; Apers et al., 2022; Chakrabarty and Liao, 2023; Liao and Chakrabarty, 2024; Anand et al., 2025). AND, OR, and XOR query-algorithms have also been considered for the problem of computing maximum cardinality matchings in bipartite graphs (see e.g. Blikstad et al. (2022)), which is a question with particular importance in communication complexity.

Aside from graphs, there is a rich body of literature concerning reconstruction of combinatorial objects, such as strings (e.g., Holden et al. (2018); Chase (2021)), partitions (e.g., Chakrabarty and Liao (2024); Black et al. (2024)), matrices (e.g., Boutsidis et al. (2014); Cohen et al. (2015)), and more.

2. Preliminaries

Throughout this paper, we consider unweighted undirected graphs $G = (V, E)$. For $v \in V$, we let $\deg_G(v)$ denote its degree and let $N_G(v)$ denote the set of v's neighbors. For $U \subseteq V$, we use $\deg_G(v, U)$ to denote the number of edges between v and U. Furthermore, we use $\mathsf{CC}_G(U)$ to denote the number of connected components in the induced subgraph $G[U]$, and we use $\mathsf{ADD}_G(U)$ to denote the number of edges in the induced subgraph $G[U]$. All subscripts G might be dropped if clear from context.

2.1. Useful Subroutines

Mazzawi (2010) showed that one can use $O(\frac{m\log(n^2/m)}{\log m})$ additive queries to reconstruct an n-node m-edge graph. This result is a key subroutine of our result. For our purpose, an $O(\frac{m\log n}{\log m})$ bound already suffices, so we will utilize the latter bound for simplicity.

Theorem 5 (Mazzawi (2010)) *There is a polynomial-time algorithm that, given* ADD*-query access to an underlying (unknown) n-node graph $G = (V, E)$, the vertex set V and the number of edges m, reconstructs the graph G using $\frac{Cm\log n}{\log m}$ queries in the worst case for some universal constant C.*

In the following, we describe several subroutines that will be used by our algorithm. Their proofs are deferred to Appendix A.

In some places, we will consider a more general version of graph reconstruction, where a subset of the edges are already known to the algorithm. In many scenarios, one can design algorithms whose query complexity mostly depends on the number of unknown edges, instead of the total number of edges.

The following lemma shows a procedure to reconstruct forests.

Lemma 6 (Reconstructing Forests) *There is a* CC*-query algorithm that, given* CC*-query access to an underlying (unknown) n-node graph $G = (V, E)$, the vertex set V and a set of edges K, reconstructs the graph G with $K \subseteq E$ when G is a forest, using $O(\frac{\max(2,m_u)\log n}{\log\max(2,m_u)})$ queries in the worst case for $m_u = |E| - |K|$. In addition, even if G is not a forest, the algorithm still uses $O(\frac{\max(2,m_u)\log n}{\log\max(2,m_u)})$ queries in the worst case, and only outputs edges in $E \setminus K$.*

Next, we show a simple algorithm that reconstructs the whoel graph using binary search, but uses up to poly-logarithmically more queries than the optimal algorithm.

Lemma 7 (Reconstructing with Binary Search) *There is an algorithm that, given* CC*-query access to an underlying (unknown) n-node graph $G = (V, E)$, the vertex set V and a set of edges K, reconstructs the graph G with $K \subseteq E$ using $O((n + m_u) \cdot \log(\frac{n^2}{n+m_u}))$ queries in the worst case, where $m_u = |E| - |K|$.*

The next lemma shows an algorithm that finds the neighbors of some vertex v in a subset of vertices U, when all edges in the induced subgraphs of U are already known.

Lemma 8 *There is an algorithm that, given* CC*-query access to an underlying (unknown) n-node graph $G = (V, E)$, the vertex set V, a subset of vertices $U \subseteq V$, all edges between vertices in U,* *an upper bound D on the maximum degree of the induced subgraph on U, and a vertex $v \in V \setminus U$, finds all edges between v and U using*

$$O\left((D + \deg(v, U)) \cdot \frac{\log(|U|/D + 1)}{\log(\max(2, \deg(v, U)/D))}\right)$$

CC-*queries in the worst case.*

The next lemma shows an algorithm that finds all vertices adjacent to a subset of vertices S.

Lemma 9 *There is an algorithm that, given* CC-*query access to an underlying (unknown) n-node graph $G = (V, E)$, the vertex set V and a subset of vertices $S \subseteq V$, outputs all vertices in V with at least one edge connected to some vertex in S, using $O((\sum_{s \in S} \deg(s)) \log n)$* CC-*queries in the worst case.*

The final lemma in this section shows an algorithm that finds all vertices whose degrees are approximately equal to some given parameter T.

Lemma 10 *There is an algorithm that, given* CC-*query access to an underlying (unknown) n-node graph $G = (V, E)$, the vertex set V, an upper bound m on the number of edges, an integer parameter $T \geq 10$, and a parameter $0 < \delta < 1$, outputs a set $H \subseteq V$ using $O(\frac{m(\log m + \log \frac{1}{\delta}) \log n}{T})$* CC-*queries in the worst case. With probability $\geq 1 - \delta$,*

- *for every $v \in V$ with $\deg(v) \geq 2T$, $v \in H$;*
- *for every $v \in V$ with $\deg(v) \leq T/2$, $v \notin H$.*

3. Adaptive Algorithm

3.1. Technical Overview for the Adaptive Algorithm

In this section, we describe the high-level ideas for our optimal adaptive algorithm. For simplicity, we only consider the case $m \geq n$ in the overview, so that the number of queries we aim for is $O(\frac{m \log n}{\log m}) = O(m)$.

Our key observation is that, when the input graph is a forest, we could use one CC query to efficiently simulate one ADD query. In fact, when the input graph $G = (V, E)$ is a forest, any induced subgraph $G[U]$ for $U \subseteq V$ is also a forest, so we have $\mathsf{ADD}(U) = |U| - \mathsf{CC}(U)$. Hence, we could use $O(m)$ CC-queries to simulate Theorem 5 for graph reconstruction which uses $O(\frac{m \log n}{\log m}) = O(m)$ ADD queries.

Hence, a natural next step is to utilize this efficient algorithm on forests, and generalize it to arbitrary graphs. We develop two main methods in order to achieve this goal:

Vertices with similar degrees. First, let us make a simplifying assumption that all vertices have similar degrees, i.e., all vertices have degrees $O(D)$ where $D = m/n$.[3] A natural idea to obtain forests from general graphs is to sample random subgraphs. Let $0 < p < 1$ be a sample probability whose value will be determined later. Suppose we sample a random subset $S \subseteq V$ where every

[3] We set the maximum degree to be $O(m/n)$ for simplicity in this overview. In general, our algorithm also works for certain larger maximum degree.

vertex $v \in V$ is added to S with probability p, what is the probability that $G[S]$ is a forest? To lower bound this probability, we instead consider the probability that $G[S]$ is a matching, i.e., the maximum degree of $G[S]$ is at most 1. Consider any vertex $u \in V$ and two of its neighbors v, w. The probability that they are all added in S is p^3. The total number of such pairs is $O(mD)$, because there are $O(m)$ pairs (u, v), and for fixed u, it can have at most $O(D)$ neighbors w. Hence, by union bound, the probability that any vertex in $G[S]$ has degree at least 2 is $O(p^3 mD)$. By setting $p = \Theta((mD)^{-1/3})$ with small enough constant, we obtain that $G[S]$ is a forest with constant probability.

However, it is not enough that $G[S]$ is a forest. Recall that the number of queries we spend on a forest with n' vertices and m' edges is $O(\frac{m' \log n'}{\log m'})$, which is only linear in the number of edges when $m' = (n')^{\Omega(1)}$. The expected number of vertices in $G[S]$ is np, and the expected number of edges is mp^2. Hence, we intuitively need $mp^2 = (np)^{\Omega(1)}$. By plugging in $p = \Theta((mD)^{-1/3})$ and $m = \Theta(nD)$, we get $mp^2 = \Theta((n/D)^{1/3})$ and $np = \Theta(n/D)^{2/3})$, so we indeed have $mp^2 = (np)^{\Omega(1)}$ as desired.

The above random sampling idea leads to a procedure that recovers $\Theta(mp^2)$ edges in expectation, using $O(mp^2)$ queries. When a large fraction of edges in the graphs are discovered, we also expect that a large fraction of edges in $G[S]$ are discovered. Hence, we use a version of the algorithm for forests that does not waste queries on these already discovered edges. Eventually, the expected number of undiscovered edges in the sampled subgraph $G[S]$ might go below $(np)^{\Omega(1)}$ (for instance, when there is only one undiscovered edge in the graph), so the query complexity might become superlinear in the number of undiscovered edges once when we have discovered many edges. Despite this, we can still show that the overall expected number of queries is $O(m)$.

Vertices with different degrees. To complement the above algorithm, we design the following algorithm that works when a vertex has degree much larger than some of its neighbors. Suppose $U \subseteq V$ is a subset where the maximum degrees of vertices in U is d and suppose we have found all edges in $G[U]$, and let $v \in V \setminus U$ be a vertex whose degree is $D \gg d$, say $D \geq d \cdot n^{0.01}$. Our algorithm will find all neighbors of v in U using $O(D)$ CC-queries.

Because the maximum degree of $G[U]$ is d, we can color the vertices of it using $d + 1$ colors, which further means that we can partition U into $d + 1$ sets $W_1, \ldots, W_{d+1}$ so that each set is an independent set. Thus, $G[W_i \cup \{v\}]$ is a forest for every $1 \leq i \leq d + 1$, and we can apply the CC-query algorithm for a forest on it. The total query complexity is thus

$$O\left(\sum_{i=1}^{d+1} \deg(v, W_i) \cdot \frac{\log(|W_i + 1|^2 / \deg(v, W_i))}{\log \deg(v, W_i)}\right) = O\left(\log n \cdot \sum_{i=1}^{d+1} \frac{\deg(v, W_i)}{\log \deg(v, W_i)}\right).$$

Assume for simplicity that $\deg(v, W_i) \geq e^2$ for every i, and use the concavity of $x/\log x$ for $x > e^2$, the above can be bounded by the following using Jensen's inequality:

$$O\left(\log n \cdot (d + 1) \cdot \frac{D/(d + 1)}{\log(D/(d + 1))}\right) = O\left(\log n \cdot \frac{D}{\log(n^{0.01})}\right) = O(D),$$

as desired.

Final algorithm. The final algorithm is an intricate combination of the above two ideas. As a very high-level intuition, we carefully set up several thresholds that partition vertices to $S_1, \ldots, S_\ell$

based on their degrees. Then we use the first algorithm that works for vertices with similar degrees to reconstruct edges in $G[S_i \cup S_{i+1}]$ for $1 \leq i < \ell$, and we use the second algorithm that works for vertices with very different degrees to reconstruct edges in between S_i and S_j for $i < j - 1$.

3.2. A Key Subroutine

In this section, we describe a key subroutine that will be used in the adaptive algorithm (we defer the proofs to Appendix B).

Lemma 11 *There is a* CC*-query algorithm that, given* CC*-query access to an underlying (unknown)* n*-node graph* $G = (V, E)$*, the vertex set* V*, an upper bound* m *on the number of edges, an upper bound* D *on the maximum degree, and an integer parameter* $\ell \geq 0$*, returns a subset* K *of all edges* E *so that (in the following,* $p = \frac{1}{10m^{1/3}D^{1/3}}$*)*

1. *The expected number of remaining edges* $|E \setminus K|$ *is upper bounded by* $|E|(1 - p/2)^{\ell}$*.*
2. *The expected number of* CC*-queries used by the algorithm is*

$$O\left(\left(\frac{|E|}{\log \max(2, p^2|E|)} + \ell\right) \cdot \log(100pn)\right).$$

The following two corollaries follow by applying Lemma 11 in two different ways.

Corollary 12 *There is an algorithm that, given given* CC*-query access to an underlying (unknown)* n*-node graph* $G = (V, E)$*, the vertex set* V*, an upper bound* m *on the number of edges, an upper bound* $D \leq \sqrt{m}$ *on the maximum degree, and a parameter* $0 < \delta < 1$*, outputs the graph with probability* $\geq 1 - \delta$ *using*

$$O\left(\left(\frac{|E|}{\log \max(2, p^2|E|)} + \frac{\log m + \log \frac{1}{\delta}}{p^2}\right) \cdot \log(100pn)\right)$$

CC*-queries in expectation for* $p = \frac{1}{10m^{1/3}D^{1/3}}$*.*

Corollary 13 *There is an algorithm that, given given* CC*-query access to an underlying (unknown)* n*-node graph* $G = (V, E)$*, the vertex set* V*, an upper bound* m *on the number of edges, and an upper bound* $D \leq \sqrt{m}$ *on the maximum degree of* G*, outputs* G *using*

$$O\left(\left(\frac{|E|}{\log \max(2, p^2|E|)} + \frac{\log(n^2/m)}{p^2}\right) \cdot \log(100pn) + (n + m^2/n^2) \cdot \log \frac{n^2}{n + m^2/n^2}\right)$$

CC*-queries in expectation, where* $p = \frac{1}{10m^{1/3}D^{1/3}}$*.*

3.3. Details of the Adaptive Algorithm

In this section, we prove our Theorem 1, which we recall below:

Theorem 1 (Adaptive Algorithm) *There is an adaptive, randomized polynomial time algorithm that, given* CC*-query access to an underlying (unknown)* n*-node graph* $G = (V, E)$*, the vertex set* V *and an upper bound* m *on the number of edges, reconstructs* G *using* $O(\frac{m \log n}{\log m})$ CC*-queries in expectation.*

Proof First, we define a sequence α where

$$\alpha_1 = m^{1/3} \text{ and } \alpha_i = \alpha_{i-1}^{0.9} \text{ for } i > 1.$$

We stop at $\alpha_\ell \le 100$. Clearly, $\ell = O(\log\log m)$.

Then we define a sequence

$$T_i = \sqrt{m/\alpha_i} \text{ for } i \ge 1.$$

Next, for every $1 \le i \le \ell$, we run Lemma 10 with parameters $V, m, T = T_i$ and $\delta = 1/10\ell$. We let the output of Lemma 10 be H_i. By a union bound, with probability 0.9, all vertices in H_i have degree more than $T_i/2$, and all vertices with degree at least $2T_i$ are in H_i. In the following, we assume these degree bounds hold. The query complexity of these calls of Lemma 10 is

$$O\left(\sum_{1\le i\le \ell} \frac{m(\log m + \log(1/\delta))\log n}{T_i}\right) \le O\left(\ell \cdot \frac{m(\log m + \log(\ell))\log n}{m^{1/3}}\right) \le O\left(\frac{m\log n}{\log m}\right).$$

We additionally set $H_0 = V$. Finally, for every $1 \le i \le \ell$, define $S_i = H_{i-1} \setminus H_i$. Note that if the guarantees of Lemma 10 hold, then the maximum degree of vertices in S_i are $\le 2T_i$ for $i \ge 1$, and the minimum degree of vertices in S_i are $\ge T_{i-1}/2$ for $i > 1$.

Then, our algorithm reconstructs edges in two cases, depending on whether the two endpoints of the edge are from two sets S_i, S_j with $|i-j| \le 1$ or not. Below, we deferred some calculations to Appendix C.

Reconstruct edges inside $S_i \cup S_{i+1}$. Here, we further consider two different cases, depending on whether $i = 1$ or not.

- Reconstruct edges inside $S_1 \cup S_2$. Here, we run Corollary 12 with parameters $V = S_1 \cup S_2, m, D = 2T_2 = \Theta(m^{0.35})$ and $\delta = 1/100$. The success rate is $1-\delta = 0.99$, and the expected query complexity can be bounded by $O\left(\frac{m\log n}{\log m}\right)$.
- Reconstruct edges inside $S_i \cup S_{i+1}$ for $i \ge 2$. In this case, we run Corollary 13 with parameters $V = S_i \cup S_{i+1}, m, D = 2T_{i+1}$. The total expected query complexity over all i can be bounded by $O(\frac{m\log n}{\log m})$.

Reconstruct edges between S_i and S_j for $1 \le i \le j-2$. Recall the maximum degree of vertices in S_i is $2T_i$ and we have computed all edges in $G[S_i]$ by the previous case. Hence, we can apply Lemma 8 with $V = S_i \cup S_j, U = S_i, D = 2T_i$ to compute all adjacent edges between any given $v \in S_j$ and S_i using

$$O\left((T_i + \deg(v, S_i)) \cdot \frac{\log(|S_i|/T_i + 1)}{\log(\max(2, \deg(v, S_i)/T_i))}\right)$$

queries. Here, we also need to consider two cases, depending on whether $i = 1$ or not.

- $i = 1$. In this case, we simply use n to upper bound $|S_i|$, and the query complexity can be bounded by $O(\frac{m\log n}{\log m})$.
- $i \ge 2$. In this case, we use $O(m/T_{i-1})$ to bound $|S_i|$, as all vertices in S_i have degree $\Omega(T_{i-1})$. The total query complexity can be bounded by $O(m)$ over all $i \ge 2$.

Summing over each stage of the algorithm, we observe that the total expected number of queries is bounded by $O(\frac{m \log n}{\log m})$. So far, the algorithm has constant success probability. As we can verify whether the graph we reconstruct is correct by verifying all discovered edges are indeed edges and the total number of discovered edges is m, we can repeat the algorithm until we find the correct graph. The expected number of repeats is $O(1)$ and hence the expected query complexity is $O(\frac{m \log m}{\log n})$. ■

4. Algorithm Using Two Rounds of Adaptivity

In this section we obtain a simple algorithm using two rounds of adaptivity making $O(m \log n + n \log^2 n)$ queries which successfully reconstructs the underlying graph with probability $1-1/\text{poly}(n)$. This is in contrast with non-adaptive algorithms (one round) which require $\Omega(n^2)$ queries, even for $m = O(n)$ (recall Theorem 3).

Theorem 4 (Two Round Algorithm) *There is a randomized algorithm using two rounds of adaptivity and $O(m \log n + n \log^2 n)$ CC-queries in the worst case which successfully reconstructs any arbitrary n-node graph with at most m edges with probability $1 - 1/\text{poly}(n)$. The upper bound m is provided as input to the algorithm.*

The first round of queries is used to attain a constant factor approximation of the degree of every vertex. This is accomplished using the following lemma, which we prove in Appendix D.1.

Lemma 14 *Fix a vertex $u \in V$. There is a non-adaptive algorithm, Algorithm 3, which makes $O(\log n \cdot \log(1/\varepsilon))$ CC-queries and returns a number $\widetilde{d}_u$ for which $\widetilde{d}_u \in [\deg_G(u), 4\deg_G(u)]$ with probability $1 - \varepsilon$.*

The second round of queries is used to learn the neighborhood of each vertex using the degree approximation obtained in the previous round. This is accomplished using the following lemma which we prove in Appendix D.2 using a basic group testing primitive.

Lemma 15 *Fix a vertex $u \in V$ and a value $d \geq \deg_G(u)$. There is a non-adaptive algorithm using $O(d \log(n/\alpha))$ CC-queries which returns a set $T \subseteq V$ such that $T = N_G(u)$ with probability $1 - \alpha$. Additionally, if $d < \deg_G(u)$, then the algorithm outputs* fail *with probability* 1.

Proof of Theorem 4. The algorithm is defined using the above two lemmas as follows.

- (1st Round) For every $u \in V$, run the procedure of Theorem 14 using error probability $\varepsilon = 1/n^2$, and let $\widetilde{d}_u$ be the corresponding output. If $\sum_{u \in V} \widetilde{d}_u > 8m$, then return fail.
- (2nd Round) For every $u \in V$, run the procedure of Theorem 15 using $d = \widetilde{d}_u$ and error probability $\alpha = 1/n^2$. Use the output as the neighborhood of u.

The first round uses $O(n \log^2 n)$ queries and outputs $\widetilde{d}_u \in [\deg_G(u), 4\deg_G(u)]$ for every $u \in V$ with probability at least $1 - \varepsilon n = 1 - 1/n$ by a union bound. Conditioned on this, we have

$$\sum_{u \in V} \widetilde{d}_u \leq 4 \sum_{u \in V} \deg_G(u) = 8|E(G)| \leq 8m \tag{1}$$

and so the algorithm continues to the second round, which successfully returns the neighborhood of every vertex with probability at least $1 - \alpha n = 1 - 1/n$ again by a union bound. Therefore, the algorithm succeeds with probability at least $1 - 2/n$ by a union bound over the two rounds. By Theorem 15 and Equation (1), the number of queries made in the second round is bounded as

$$\sum_{u \in V} O(\widetilde{d}_u \log n) \leq O(m \log n)$$

in the worst case, and this completes the proof. ■

5. Lower Bounds

5.1. Non-Adaptive Lower Bound

In this section we prove Theorem 3, establishing a CC-query complexity lower bound of $\Omega(n^2)$ for non-adaptive algorithms, even when the graph is known to have only $O(n)$ edges.

Theorem 3 (Non-Adaptive Lower Bound) *Any non-adaptive graph reconstruction algorithm requires $\Omega(n^2)$ CC-queries in expectation to achieve $1/2$ success probability, even when the graph is known to have $O(n)$ edges.*

Proof For every distinct pair of vertices $\{u, v\} \in \binom{V}{2}$, we define a pair of graphs $G^0_{u,v}(V, E_0)$ and $G^1_{u,v}(V, E_1)$ as follows. The edges of $G^0_{u,v}$ are given by $E_0 := \{(u, w), (w, v) \colon w \in V \setminus \{u, v\}\}$, i.e. $G^0_{u,v}$ is a union of 2-paths joining u and v, through every other vertex w. The edges of $G^1_{u,v}$ are simply defined as $E_1 := E_0 \cup \{(u, v)\}$. Observe that for an algorithm to distinguish these two graphs it clearly must query a set S that contains both u and v. However, if S contains any additional vertices besides u, v, then the subgraph induced on S is connected for both graphs, and so $\mathsf{CC}(S)$ returns 1 for both $G^0_{u,v}$ and $G^1_{u,v}$. Therefore, if a set of queries $Q \subseteq 2^V$ distinguishes these two graphs then it must be the case that $\{u, v\} \in Q$. Moreover, this implies that the number of pairs $\{u, v\} \in \binom{V}{2}$ such that Q distinguishes $G^0_{u,v}$ and $G^1_{u,v}$ is at most $|Q|$.

Now, let A be any non-adaptive CC-query algorithm which successfully recovers any arbitrary input graph with probability $\geq 1/2$. The algorithm A queries a random set $Q \subseteq 2^V$ according to some distribution, $\mathcal{D}_A$. In particular, for every $\{u, v\} \in \binom{V}{2}$, Q distinguishes $G^0_{u,v}$ and $G^1_{u,v}$ with probability $\geq 1/2$. Thus,

$$\begin{aligned}\frac{1}{2}\binom{n}{2} &\leq \sum_{\{u,v\} \in \binom{V}{2}} \mathbf{Pr}_{Q \sim \mathcal{D}_A}[Q \text{ distinguishes } G^0_{u,v} \text{ and } G^1_{u,v}] \\ &= \mathbf{E}_{Q \sim \mathcal{D}_A}\left[\left|\left\{\{u, v\} \in \binom{V}{2} : Q \text{ distinguishes } G^0_{u,v} \text{ and } G^1_{u,v}\right\}\right|\right] \leq \mathbf{E}[|Q|]\end{aligned}$$

using linearity of expectation, and this completes the proof. ■

5.2. Adaptive Lower Bound

In this section we prove Theorem 2, establishing a CC-query complexity lower bound of $\Omega(\frac{m\log n}{\log m})$ for arbitrary fully adaptive algorithms, for all values of m, and even when m is provided to the algorithm.

Theorem 2 (Adaptive Lower Bound) *Any (randomized) adaptive graph reconstruction algorithm requires $\Omega(\frac{m\log n}{\log m})$ CC-queries in expectation to achieve $1/2$ success probability for all $m \le \binom{n}{2}-1$ and even when m is provided as input.*

Proof We first consider the case of relatively small m and give a simple information-theoretic argument. Concretely, let us assume $m \le n$. Observe that for any $U \subseteq V$, the number of connected components in $G[U]$ is at least $|U| - m$ and at most $|U|$. Thus, any CC-query returns at most $\log(m+1)$ bits of information. Since $m \le n$, the total number of graphs with n nodes and m edges is

$$\binom{\binom{n}{2}}{m} \ge \left(\binom{n}{2}/m\right)^m \ge \left(\binom{n}{2}/n\right)^m = \left(\frac{n-1}{2}\right)^m = 2^{\Omega(m\log n)}$$

implying that $\Omega(\frac{m\log n}{\log m})$ CC-queries are needed in this case.

Now suppose $m \ge n$. Note that in this case $\frac{m\log n}{\log m} = \Theta(m)$ and so it suffices to prove an $\Omega(m)$ lower bound. Consider the largest integer n_0 for which $\binom{n_0}{2} - 1 \le m$. Let $m_{\text{dum}} = m - (\binom{n_0}{2} - 1)$ and observe that

$$m_{\text{dum}} < \left(\binom{n_0+1}{2} - 1\right) - \left(\binom{n_0}{2} - 1\right) = n_0. \tag{2}$$

We define a family $\mathcal{G}$ of n-node graphs with m edges as follows. Fix a set K of n_0 vertices and a set I of $n - n_0$ vertices. Each graph $G \in \mathcal{G}$ is defined as follows. First, make $G[K]$ an n_0-clique, but choose a single edge and remove it. This accounts for $\binom{n_0}{2} - 1$ edges. If $n = n_0$, i.e. $m = \binom{n}{2} - 1$, then $I = \emptyset$ and this completes the construction. Otherwise, we complete the construction by taking a single vertex $z_{\text{dum}} \in I$ and adding a dummy edge from z_{dum} to m_{dum}-many vertices in K. (This is possible by Equation (2)). Note the total number of edges is now m.

Next, we reduce the task of finding the unique zero in a Boolean array of length $\binom{n_0}{2} = m + 1 - m_{\text{dum}}$ (unstructured search) to the task of reconstructing an arbitrary graph $G \in \mathcal{G}$ using CC-queries. As unstructured search requires $\Omega(m + 1 - m_{\text{dum}}) = \Omega(m)$ queries to achieve success probability $1/2$, graph reconstruction will also require $\Omega(m)$ queries. (By Equation (2), $m_{\text{dum}} < n_0 = O(\sqrt{m})$.)

We can arbitrarily map the elements in the Boolean array to potential edges in $G[K]$, where a one corresponds to an edge, and a zero corresponds to a non-edge. Suppose we have an algorithm $\mathcal{A}$ for reconstructing $G \in \mathcal{G}$, i.e., the algorithm is able to discover where the non-edge in K is. Then we construct an algorithm $\mathcal{A}'$ for finding the unique zero in the Boolean array by simulating $\mathcal{A}$. Whenever $\mathcal{A}$ makes a query to a subset S where $|S \cap K| \ne 2$, $\mathcal{A}'$ does not need to make any query. Indeed, in this case the vertices in $S \cap K$ are always connected, and all edges in $(S \cap K) \times (S \setminus K)$ are fixed, so $\mathsf{CC}(S)$ is fixed for graphs in $\mathcal{G}$. If $\mathcal{A}$ makes a query to a subset S where $S \cap K = \{u, v\}$, $\mathcal{A}'$ first makes a query to the element corresponding to the edge $\{u, v\}$. All other edges among $S \cap K$ are fixed for graphs in $\mathcal{G}$, so the value $\mathsf{CC}(S)$ can be computed using 1 query. Whenever $\mathcal{A}$ terminates and finds the non-edge in K, $\mathcal{A}'$ also terminates and claims the unique zero is the element corresponding to that non-edge. Since the number of queries $\mathcal{A}'$ makes never exceeds the number of queries $\mathcal{A}$ makes, this concludes the reduction and thus the proof. ■

6. Conclusion and Open Questions

We have shown tight bounds on the CC-query complexity of graph reconstruction. Our algorithm is adaptive, and we have shown that non-adaptive algorithms cannot do better than the $O(n^2)$ trivial upper bound even for graphs with $O(n)$ edges. This is in contrast with the graph reconstruction problem using CUT and ADD queries, where there exist non-adaptive algorithms attaining the optimal query complexity. We believe it is an interesting direction to investigate the minimal number of rounds that suffice to attain the optimal $\Theta(\frac{m \log n}{\log m})$ CC-query complexity. Towards this, we have obtained a two round algorithm using $O(m \log n + n \log^2 n)$ queries.

Question 16 (Rounds of adaptivity) *What is the minimal number of rounds of adaptivity sufficient to obtain a $O(\frac{m \log n}{\log m})$ CC-query algorithm? As a starting point, what is the optimal query complexity for algorithms using two rounds of adaptivity?*

Our results show that CC-queries and ADD/CUT-queries have different power for the graph reconstruction problem in terms of rounds of adaptivity. It would be interesting to investigate to what extent these queries are comparable. One concrete question towards this is the following.

Question 17 (Counting edges) *How many CC-queries suffice to learn, or approximate, the number of edges in an arbitrary input graph?*

In general, it will be interesting to quantify the number of CC-queries required to derive or verify (testing) some specific property of the graph. Potentially such tasks can be performed with much fewer queries than required for reconstructing the graph.

Acknowledgments

This research is funded by NSF HDR TRIPODS Phase II grant 2217058 (EnCORE Institute).

References

Hasan Abasi and Nader H. Bshouty. On learning graphs with edge-detecting queries. In *Algorithmic Learning Theory, ALT 2019*, volume 98 of *Proceedings of Machine Learning Research*, pages 3–30. PMLR, 2019.

Noga Alon and Vera Asodi. Learning a hidden subgraph. *SIAM J. Discret. Math.*, 18(4):697–712, 2005. doi: 10.1137/S0895480103431071.

Noga Alon, Richard Beigel, Simon Kasif, Steven Rudich, and Benny Sudakov. Learning a hidden matching. *SIAM J. Comput.*, 33(2):487–501, 2004. doi: 10.1137/S0097539702420139.

Aditya Anand, Thatchaphol Saranurak, and Yunfan Wang. Deterministic edge connectivity and max flow using subquadratic cut queries. In *Symposium on Discrete Algorithms, SODA*, 2025.

Dana Angluin and Jiang Chen. Learning a hidden graph using O(log n) queries per edge. *J. Comput. Syst. Sci.*, 74(4):546–556, 2008. doi: 10.1016/J.JCSS.2007.06.006.

Simon Apers, Yuval Efron, Pawel Gawrychowski, Troy Lee, Sagnik Mukhopadhyay, and Danupon Nanongkai. Cut query algorithms with star contraction. In *Proceedings of the 63rd IEEE Annual Symposium on Foundations of Computer Science (FOCS 2022)*, pages 507–518, 2022. doi: 10.1109/FOCS54457.2022.00055.

Hassan Ashtiani, Shrinu Kushagra, and Shai Ben-David. Clustering with same-cluster queries. *Advances in neural information processing systems*, 29, 2016.

Sepehr Assadi, Deeparnab Chakrabarty, and Sanjeev Khanna. Graph connectivity and single element recovery via linear and OR queries. In *European Symposium on Algorithms, (ESA)*, 2021.

Arinta Auza and Troy Lee. On the query complexity of connectivity with global queries. *CoRR*, abs/2109.02115, 2021.

Maria-Florina Balcan and Avrim Blum. Clustering with interactive feedback. In *International Conference on Algorithmic Learning Theory*, pages 316–328. Springer, 2008.

Paul Bastide and Carla Groenland. Optimal distance query reconstruction for graphs without long induced cycles. *CoRR*, abs/2306.05979, 2023. doi: 10.48550/ARXIV.2306.05979. URL https://doi.org/10.48550/arXiv.2306.05979.

Zuzana Beerliova, Felix Eberhard, Thomas Erlebach, Alexander Hall, Michael Hoffmann, Matús Mihalák, and L. Shankar Ram. Network discovery and verification. *IEEE J. Sel. Areas Commun.*, 24(12):2168–2181, 2006. doi: 10.1109/JSAC.2006.884015.

Hadley Black, Euiwoong Lee, Arya Mazumdar, and Barna Saha. Clustering with non-adaptive subset queries. In *Neural Information Processing Systems (NeurIPS)*, 2024.

Joakim Blikstad, Jan van den Brand, Yuval Efron, Sagnik Mukhopadhyay, and Danupon Nanongkai. Nearly optimal communication and query complexity of bipartite matching. In *Foundations of Computer Science, FOCS*, 2022.

Christos Boutsidis, Petros Drineas, and Malik Magdon-Ismail. Near-optimal column-based matrix reconstruction. *SIAM J. Comput.*, 43(2):687–717, 2014. doi: 10.1137/12086755X.

Marco Bressan, Nicolò Cesa-Bianchi, Silvio Lattanzi, and Andrea Paudice. Exact recovery of mangled clusters with same-cluster queries. *Advances in Neural Information Processing Systems*, 33:9324–9334, 2020.

Nader H. Bshouty and Hanna Mazzawi. Reconstructing weighted graphs with minimal query complexity. *Theor. Comput. Sci.*, 412(19):1782–1790, 2011. doi: 10.1016/J.TCS.2010.12.055.

Nader H. Bshouty and Hanna Mazzawi. Toward a deterministic polynomial time algorithm with optimal additive query complexity. *Theor. Comput. Sci.*, 2012.

Nader H. Bshouty and Hanna Mazzawi. On parity check (0, 1)-matrix over $\mathcal{F}_{\mathrm{p}}$. *SIAM J. Discret. Math.*, 29(1):631–657, 2015. doi: 10.1137/120881129.

John Bunge and Michael Fitzpatrick. Estimating the number of species: a review. *Journal of the American statistical Association*, 88(421):364–373, 1993.

Deeparnab Chakrabarty and Hang Liao. A query algorithm for learning a spanning forest in weighted undirected graphs. In *International Conference on Algorithmic Learning Theory (ALT)*, 2023.

Deeparnab Chakrabarty and Hang Liao. Learning partitions using rank queries. In *Foundations of Software Technology and Theoretical Computer Science, FSTTCS*, 2024.

Zachary Chase. Separating words and trace reconstruction. In *Symposium on Theory of Computing (STOC)*, 2021.

Sung-Soon Choi. Polynomial time optimal query algorithms for finding graphs with arbitrary real weights. In *Proceedings of the 26th Annual Conference on Learning Theory (COLT 2013)*, pages 797–818, 2013.

Sung-Soon Choi and Jeong Han Kim. Optimal query complexity bounds for finding graphs. *Artif. Intell.*, 174(9-10):551–569, 2010. doi: 10.1016/J.ARTINT.2010.02.003.

Michael B. Cohen, Sam Elder, Cameron Musco, Christopher Musco, and Madalina Persu. Dimensionality reduction for k-means clustering and low rank approximation. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing (STOC 2015)*, pages 163–172, 2015. doi: 10.1145/2746539.2746569.

Alberto Del Pia, Mingchen Ma, and Christos Tzamos. Clustering with queries under semi-random noise. In *Conference on Learning Theory*, pages 5278–5313. PMLR, 2022.

Adela DePavia, Olga Medrano Martin del Campo, and Erasmo Tani. Optimal algorithms for learning partitions with faulty oracles. In *Adv. in Neu. Inf. Proc. Sys. (NeurIPS)*, 2024.

Thomas Erlebach, Alexander Hall, Michael Hoffmann, and Matús Mihalák. Network discovery and verification with distance queries. In *Proceedings of the 6th Italian Conference on Algorithms and Complexity (CIAC 2006)*, pages 69–80, 2006. doi: 10.1007/11758471_10.

Ove Frank. Estimation of the number of connected components in a graph by using a sampled subgraph. *Scandinavian Journal of Statistics*, pages 177–188, 1978.

Mohsen Ghaffari and Merav Parter. MST in log-star rounds of congested clique. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 19–28. ACM, 2016. doi: 10.1145/2933057.2933103.

Vladimir Grebinski. On the power of additive combinatorial search model. In *Proceedings of the 4th Annual International Conference on Computing and Combinatorics (COCOON)*, pages 194–203, 1998.

Vladimir Grebinski and Gregory Kucherov. Reconstructing a hamiltonian cycle by querying the graph: Application to DNA physical mapping. *Discret. Appl. Math.*, 88(1-3):147–165, 1998. doi: 10.1016/S0166-218X(98)00070-5.

Vladimir Grebinski and Gregory Kucherov. Optimal reconstruction of graphs under the additive model. *Algorithmica*, 28(1):104–124, 2000. doi: 10.1007/S004530010033.

Nicholas James Alexander Harvey. *Matchings, matroids and submodular functions*. PhD thesis, Massachusetts Institute of Technology, 2008.

Nina Holden, Robin Pemantle, and Yuval Peres. Subpolynomial trace reconstruction for random strings and arbitrary deletion probability. In *Conference On Learning Theory, COLT*, 2018.

Wasim Huleihel, Arya Mazumdar, Muriel Médard, and Soumyabrata Pal. Same-cluster querying for overlapping clusters. *Advances in Neural Information Processing Systems*, 32, 2019.

Sampath Kannan, Claire Mathieu, and Hang Zhou. Near-linear query complexity for graph inference. In *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming (ICALP 2015)*, pages 773–784, 2015. doi: 10.1007/978-3-662-47672-7_63.

Sampath Kannan, Claire Mathieu, and Hang Zhou. Graph reconstruction and verification. *ACM Trans. Algorithms*, 2018.

Christian Konrad, Conor O'Sullivan, and Victor Traistaru. Graph reconstruction via MIS queries. In *Innovations in Theoretical Computer Science (ITCS)*, 2025.

Evangelos Kranakis, Danny Krizanc, and Jorge Urrutia. Implicit routing and shortest path information (extended abstract). In *Proceedings of the 2nd Colloquium on Structural Information and Communication Complexity (SIROCCO 1995)*, pages 101–112, 1995.

Evangelos Kranakis, Danny Krizanc, and Yun Lu. Reconstructing cactus graphs from shortest path information - (extended abstract). In *Proceedings of the 11th International Conference on Algorithmic Aspects in Information and Management (AAIM 2016)*, pages 150–161, 2016. doi: 10.1007/978-3-319-41168-2_13.

Hang Liao and Deeparnab Chakrabarty. Learning spanning forests optimally in weighted undirected graphs with CUT queries. In *International Conference on Algorithmic Learning Theory (ALT)*, 2024.

Xizhi Liu and Sayan Mukherjee. Tight query complexity bounds for learning graph partitions. In *Conference on Learning Theory*, pages 167–181. PMLR, 2022.

Claire Mathieu and Hang Zhou. Graph reconstruction via distance oracles. In *Proceedings of the 40th International Colloquium on Automata, Languages, and Programming (ICALP 2013)*, pages 733–744, 2013. doi: 10.1007/978-3-642-39206-1_62.

Claire Mathieu and Hang Zhou. A simple algorithm for graph reconstruction. *Random Struct. Algorithms*, 2023.

Arya Mazumdar and Barna Saha. Clustering with noisy queries. In *Adv. in Neu. Inf. Proc. Sys. (NeurIPS)*, 2017a.

Arya Mazumdar and Barna Saha. Query complexity of clustering with side information. In *Adv. in Neu. Inf. Proc. Sys. (NeurIPS)*, 2017b.

Arya Mazumdar and Barna Saha. A theoretical analysis of first heuristics of crowdsourced entity resolution. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2017c.

Hanna Mazzawi. Optimally reconstructing weighted graphs using queries. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2010)*, pages 608–615, 2010. doi: 10.1137/1.9781611973075.51.

Michael Mitzenmacher and Charalampos E. Tsourakakis. Predicting signed edges with $O(n^{1+o(1)} \log n)$ queries. *arXiv preprint arXiv:1609.00750*, 2016.

Lev Reyzin and Nikhil Srivastava. Learning and verifying graphs using queries with a focus on edge counting. In *Proceedings of the 18th International Conference on Algorithmic Learning Theory (ALT 2007)*, pages 285–297, 2007. doi: 10.1007/978-3-540-75225-7_24.

Barna Saha and Sanjay Subramanian. Correlation clustering with same-cluster queries bounded by optimal cost. In *27th Annual European Symposium on Algorithms (ESA 2019)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2019.

Appendix A. Deferred Proofs from Section 2

Proof of Lemma 6. At the beginning of the algorithm, we make one query to compute the number of connected components $\mathsf{CC}(G)$ of the whole graph, and set $\hat{m}_u = n - \mathsf{CC}(G) - |K| \leq m_u$.

First, we assume G is a forest. In this case, note that $\hat{m}_u = m_u$. We run Theorem 5 on the graph $G_u = (V, E \setminus K)$ and halt the algorithm once the number of queries exceeds $\frac{C\hat{m}_u \log n}{\log \hat{m}_u}$. We use CC queries on G to simulate ADD queries on G_u. Whenever Theorem 5 queries the number of edges in an induced subgraph $G_u[U]$ for some $U \subseteq V$, we make a CC query on $G[U]$. Since G is a forest, the number of edges in $G[U]$ equals $|U| - \mathsf{CC}(U)$. Then we can compute the number of edges in $G_u[U]$ by subtracting $|E(G[U]) \cap K|$ from the previous count. Hence, one CC query on G suffices to simulate one ADD query on G_u, so the first part of the proposition follows from Theorem 5.

Since we always halt the algorithm when Theorem 5 makes more than $\frac{C\hat{m}_u \log n}{\log \hat{m}_u} \leq \frac{Cm_u \log n}{\log m_u}$ queries, the number of queries is bounded by $O(1 + \frac{m_u \log n}{\log m_u}) \leq O(\frac{\max(2, m_u) \log n}{\log \max(2, m_u)})$, even when G is not a forest. Furthermore, as we can verify in one query whether an edge output by the algorithm actually belongs to $E \setminus K$, the second part of the lemma also follows. Formally, we can iterate over the candidate edges returned by the algorithm, spending one query to verify each and halting as soon as a pair not belonging to $E \setminus K$ is seen. Thus we spend at most m_u queries for the verification and since $\log(m_u) = O(\log n)$, we have $m_u = O(\frac{\max(2, m_u) \log n}{\log \max(2, m_u)})$, and so this does not dominate the query complexity. ■

Proof of Lemma 7. For each vertex $v \in V$, let U_v be the set of vertices in $V \setminus \{v\}$ that do not yet have an edge in K with v. Note that for any subset $U \subseteq V \setminus \{v\}$, there is at least one edge between v and U if and only if $\mathsf{CC}(U) = \mathsf{CC}(U \cup \{v\})$. Therefore, we can perform a binary-search style algorithm to find all neighbors of v in U_v. More specifically, we use the above method to check whether v has at least one edge with U_v. If not, we halt and report no edge. Otherwise, we split U_v into two equal halves (up to 1 vertex), and recursively solve the problem in the two halves (unless $|U_v| = 1$, in which case we can simply report this edge). Let $d = \max(1, \deg(v, U_v))$. The total number of subsets we query with size at least n/d is $O(d)$, as there are only $O(d)$ of them in the recursion process; for smaller subsets, the binary search costs $O(\log(n/d) + 1) = O(\log(2d/d))$ queries. Hence, the overall number of queries needed is $O(d \log(2n/d)) = O\left(\max(1, \deg(v, U_v)) \cdot \log\left(\frac{2n}{\max(1, \deg(v, U_v))}\right)\right)$.

Summing over all vertices, the total number of queries is

$$O\left(\sum_{v \in V} \max(1, \deg(v, U_v)) \cdot \log\left(\frac{2n}{\max(1, \deg(v, U_v))}\right)\right).$$

By Jensen's inequality, the above can be upper bounded by

$$O\left(n \cdot \left(1 + \frac{2m_u}{n}\right) \cdot \log\left(\frac{2n}{1 + \frac{2m_u}{n}}\right)\right) = O\left((n + m_u) \cdot \log\left(\frac{n^2}{n + m_u}\right)\right).$$

■

Proof of Lemma 8. Since the maximum degree of $G[U]$ is at most D, we can color $G[U]$ using $D + 1$ colors. Then each color class forms an independent set and we denote the independent sets by $I_1, \ldots, I_{D+1}$. If some independent set has size greater than $|U|/D$, we arbitrarily decompose the independent set to smaller ones of size $O(|U|/D)$. The total number of independent sets is still $\ell = O(D)$. Between every independent set $I_i \subseteq U$ and v, we run Lemma 6 on the subgraph induced on $I_i \cup \{v\}$. As I_i is an independent set, this induced subgraph is a forest, so Lemma 6 finds all edges between v and I_i in

$$O\left(\frac{\max(2, \deg(v, I_i)) \cdot \log(|U|/D + 1)}{\log \max(2, \deg(v, I_i))}\right)$$

queries. By Jensen's inequality, the total number of queries is hence

$$O\left(\sum_{i=1}^{\ell} \frac{\max(2, \deg(v, I_i)) \log(|U|/D + 1)}{\log \max(2, \deg(v, I_i))}\right).$$

By upper bounding $\max(2, \deg(v, I_i))$ by $e^2 + \deg(v, I_i)$ and use the fact that the function $x/\log x$ is concave for $x > e^2$ and apply Jensen's inequality, the above can be bounded by

$$\begin{aligned} &O\left(\ell \cdot (e^2 + \deg(v, U)/\ell) \cdot \frac{\log(|U|/D + 1)}{\log(e^2 + \deg(v, U)/\ell)}\right) \\ &= O\left((D + \deg(v, U)) \cdot \frac{\log(|U|/D + 1)}{\log(\max(2, \deg(v, U)/D))}\right). \end{aligned}$$

■

Proof of Lemma 9. Given an arbitrary subset $U \subseteq V$, we can determine whether there is an edge between S and U using $O(1)$ CC queries as follows. Let $W = S \cap U$. Then there is no edge between S and U if and only if

$$\mathsf{CC}(S \setminus W) + \mathsf{CC}(W) + \mathsf{CC}(U \setminus W) = \mathsf{CC}(S \cup U) \text{ and } \mathsf{CC}(W) = |W|.$$

Therefore, we can find all vertices adjacent to S using a binary-search approach. Start by setting $U = V$. If we determine (using $O(1)$ queries as described in the previous paragraph) that there is no edge between S and U, we do not need to do any work. Otherwise, if $|U| = 1$, we output the vertex in U as one vertex adjacent to S. Otherwise, we arbitrary split U to two halves U_1 and U_2, and recursively find all neighbors of S in U_1 and U_2. It is not difficult to bound the number of queries of this procedure by $O((\sum_{s \in S} \deg(s)) \log n)$. ■

Proof of Lemma 10. The algorithm is described in Algorithm 1.

First, we analyze the number of queries used by Algorithm 1. Each iteration of the for-loop uses $O(\frac{m \log n}{T})$ queries, so the overall number of queries is $O(\frac{m\ell \log n}{T}) = O(\frac{m(\log m + \log \frac{1}{\delta}) \log n}{T})$.

Next, we analyze the error probability. For every $v \in V$ with $\deg(v) \geq 2T$ and for some iteration i of the algorithm, let $\mathcal{E}_1$ be the event that the sample S is disjoint from the neighbors of v. We have that

$$\mathbf{Pr}[\mathcal{E}_1] \leq \left(1 - \frac{1}{T}\right)^{2T} \leq e^{-2} \leq 0.2.$$

Algorithm 1: Algorithm for finding high-degree vertices

```
Input: $V, m, T, \delta$ and CC-query access to a graph $G = (V, E)$ with $n$ nodes and at most $m$ edges.
Output: A subset $H \subseteq V$ with the property described in Lemma 10.
$\ell \leftarrow \lceil 200(\ln(2m) + \ln(1/\delta) \rceil$;
Initialize an array $a$ indexed on $V$ initially all 0;
for $i = 1$ to $\ell$ do
    Sample a subset $S \subseteq V$ where each $v \in V$ is added to $S$ independently with probability $\frac{1}{T}$;
    Run Lemma 9 with $V, S$ to get a set $U \subseteq V$ but halt Lemma 9 when the number of queries
      exceeds $O(\frac{20m}{T} \log n)$ where the constant factor hidden in $O$ is the same as the constant
      factor in Lemma 9;
    if Lemma 9 finishes then
        for $v \in U$ do
            $a_v \leftarrow a_v + 1$;
        end
    end
end
return $H = \{v \in V : a_v \geq \ell/2\}$;
```

Furthermore, let $\mathcal{E}_2$ be the event that $\sum_{s \in S} \deg(s) \geq \frac{20m}{T}$. It is not difficult to see that

$$\mathbf{E}\left[\sum_{s \in S} \deg(s)\right] \leq \frac{2m}{T}.$$

Therefore, by Markov's inequality, $\mathbf{Pr}[\mathcal{E}_2] \leq 0.1$. As long as neither $\mathcal{E}_1$ nor $\mathcal{E}_2$ happens, v will be among the vertices returned by Lemma 9, and thus a_v will be incremented in the iteration. Hence, a_v is incremented in each iteration with probability $\geq 1 - \mathbf{Pr}[\mathcal{E}_1 \vee \mathcal{E}_2] \geq 0.7$. By Chernoff bound, at the end of the algorithm,

$$\mathbf{Pr}\left[a_v \leq \ell/2\right] \leq \mathbf{Pr}\left[a_v \leq (1 - 0.2/0.7)\mathbf{E}[a_v]\right] \leq \exp(-(0.2/0.7)^2 \cdot (0.7\ell)/2) \leq \frac{\delta}{2m}.$$

In other words, v will be output by the algorithm with probability $\geq 1 - \frac{\delta}{2m}$.

Next, we consider vertices $v \in V$ with $\deg(v) \leq T/2$. For any iteration i, the probability that S is disjoint from the neighbors of v is (recall $T \geq 10$)

$$\geq \left(1 - \frac{1}{T}\right)^{T/2} \geq \left(1 - \frac{1}{10}\right)^{10/2} \geq 0.59.$$

Therefore, the probability that a_v is incremented in the iteration is ≤ 0.41. Thus, by Chernoff bound, at the end of the algorithm,

$$\mathbf{Pr}\left[a_v \geq \ell/2\right] \leq \exp(-(0.09/0.41)^2 \cdot (0.41\ell)/(2 + (0.09/0.41))) \leq \frac{\delta}{2m}.$$

Furthermore, notice that the algorithm never outputs a vertex v with degree 0, since a_v is always 0 for such vertices.

Finally, the lemma follows by union bound over all vertices whose degree is nonzero (there can be at most $2m$ such vertices). ■

Appendix B. Deferred Proofs from Section 3.2

The main strategy for the proof of Lemma 11 is to randomly sample induced subgraphs where each vertex is selected into the subgraph with probability p. This probability $p = \frac{1}{10m^{1/3}D^{1/3}}$ is chosen so that each sampled induced subgraph is a forest (more specifically, a matching) with good probability, and hence we can use Lemma 6 to reconstruct edges in the subgraph.

The algorithm is outlined in Algorithm 2.

```
Algorithm 2: A key subroutine
Input: (1) CC-query access to graph $G = (V, E)$ with $n$ vertices, at most $m$ edges and
       maximum degree at most $D$. (2) $V, m, D$ and a parameter $\ell \geq 0$.
Output: A set of edges $K \subseteq E$.
1  $p \leftarrow \frac{1}{10m^{1/3}D^{1/3}}$;
2  $K \leftarrow \emptyset$;
3  for $t = 1 \rightarrow \ell$ do
4      Sample $S \subseteq V$ with sample-rate $p$;
5      if $|S| \leq 100pn$ then
6          Run Lemma 6 on $G[S]$ with set of known edges $E(G[S]) \cap K$ to find
             $F \subseteq E(G[S]) \setminus K$;
7          $K \leftarrow K \cup F$;
8      end
9  end
10 return $K$;
```

Let m_t be the number of undiscovered edges at the end of iteration t in Algorithm 2. Initially, $m_0 = |E|$.

The proofs of the next three lemmas are deferred to Appendix B.

Lemma 18 *For any* $1 \leq t \leq \ell$, $\mathbf{E}[m_t \mid m_{t-1}] \leq (1 - p^2/2)m_{t-1}$.

Proof We say the random set S sampled at Line 4 of Algorithm 2 is *good* if the following two events occur.

- $\mathcal{E}_1$: $|S| \leq 100pn$.
- $\mathcal{E}_2$: $G[S]$ is a matching.

Claim 19 *Let* $(u, v) \in E \setminus K$ *be an arbitrary unknown edge. If* $u, v \in S$ *and* S *is good, then the edge* (u, v) *is found by Lemma 6 at Line 6.*

Hence, it suffices to bound the probability of the event above.

Claim 20 *Let $(u, v) \in E \setminus K$ be an arbitrary unknown edge. Then,* $\Pr[u, v \in S$ *and* S *is good*$] \geq p^2/2$.

Proof First, clearly $\Pr[u, v \in S] = p^2$ and so it suffices to show that $\Pr[S \text{ is good} \mid u, v \in S] \geq 1/2$. We will establish this by showing that conditioned on $u, v \in S$, each of the events $\mathcal{E}_1, \mathcal{E}_2$ above fail to occur with probability at most $1/4$ and taking a union bound.

Clearly, $\mathbf{E}_S[|S| \mid u, v \in S] = p(n-2)+2 = pn+2(1-p)$. Since $n \geq n^{1/3}D^{2/3} \geq m^{1/3}D^{1/3}$ (we have $m \leq nD$ as D is an upper bound on the max-degree), we have $pn \geq 1/10$ and thus $pn + 2(1-p) \leq pn + 2 \leq 21pn$. By Markov's inequality, $\mathbf{Pr}[\neg\mathcal{E}_1 \mid u, v \in S] \leq \frac{21pn}{100pn} \leq 1/4$.

We now prove the desired bound for $\mathcal{E}_2$. Note that $G[S]$ is *not* a matching iff either (a) some vertex $z \in N(u) \cup N(v)$ appears in S, or (b) some vertex $w \in V \setminus (\{u, v\} \cup N(u) \cup N(v))$ appears in S and has at least two neighbors appearing in S. Since $|N(u) \cup N(v)| \leq 2D$, the probability of the former occurring is at most $2pD = \frac{1}{5} \cdot \frac{D^{2/3}}{m^{1/3}} \leq \frac{1}{5}$ (recall $D \leq \sqrt{m}$). For the latter, fix $w \notin \{u, v\} \cup N(u) \cup N(v)$, and two of its neighbors $x, y \in N(w)$. The probability that w, x, y all appear in S is p^3. By union bound over all choices of w, x, y, the probability of (b) is bounded by

$$p^3 \cdot |\{(w, x, y) : w \in V, x, y \in N(w)\}| \leq p^3 \cdot 2mD \leq \frac{1}{500},$$

which completes the proof. ■

By combining Claim 19 and Claim 20, the probability that each undiscovered edge before the t-th iteration of the for loop at Line 3 becomes discovered after the iteration is at least $p^2/2$. Therefore, $\mathbf{E}[m_t \mid m_{t-1}] \leq (1 - p^2/2)m_{t-1}$ follows by linearity of expectation. ■

Lemma 21 *For $1 \leq t \leq \ell$, the expected number of queries by Algorithm 2 in the t-th iteration of the loop at Line 3 is*

$$O\left(\frac{\max(2, p^2\mathbf{E}[m_{t-1}])}{\log \max(2, p^2\mathbf{E}[m_{t-1}])} \cdot \log(100pn)\right).$$

Proof Clearly, the only way the algorithm makes queries is via Lemma 6. Let x_t be the number of unknown edges in $G[S]$ in the t-th iteration. The number of queries made by Lemma 6 is thus at most

$$O\left(\frac{\max(2, x_t)}{\log \max(2, x_t)} \cdot \log(100pn)\right).$$

The function $\frac{x_t}{\log x_t}$ is concave for sufficiently large x_t ($x_t > e^2$). In order to apply Jensen's inequality, we define an auxiliary variable $y_t = e^2 + x_t$, and use

$$O\left(\frac{y_t \log(100pn)}{\log y_t}\right)$$

to upper bound the number of queries used in the iteration. By Jensen's inequality,

$$\mathbf{E}\left[\frac{y_t \log(100pn)}{\log y_t}\right] \leq \frac{\mathbf{E}[y_t] \log(100pn)}{\log \mathbf{E}[y_t]} \leq \frac{(\mathbf{E}[x_t] + e^2) \log(100pn)}{\log(\mathbf{E}[x_t] + e^2)} = \frac{(p^2\mathbf{E}[m_{t-1}] + e^2) \log(100pn)}{\log(p^2\mathbf{E}[m_{t-1}] + e^2)},$$

which is further upper bounded by

$$O\left(\frac{\max(2, p^2\mathbf{E}[m_{t-1}])}{\log\max(2, p^2\mathbf{E}[m_{t-1}])} \cdot \log(100pn)\right).$$

■

Lemma 22 *The expected number of queries of Algorithm 2 is*

$$O\left(\left(\frac{|E|}{\log\max(2, p^2|E|)} + \ell\right) \cdot \log(100pn)\right)$$

Proof By combining Lemma 18 and Lemma 21, the expected cost of the algorithm can be bounded by

$$O\left(\sum_{t=0}^{\ell-1} \frac{\max(2, p^2\mathbf{E}[m_t])}{\log\max(2, p^2\mathbf{E}[m_t])} \cdot \log(100pn)\right) \leq O\left(\sum_{t=0}^{\ell-1} \frac{\max(2, p^2(1-p^2/2)^t|E|)}{\log\max(2, p^2(1-p^2/2)^t|E|)} \cdot \log(100pn)\right).$$

We decompose the above sum based on values of $p^2(1-p^2/2)^t|E|$.

- $p^2(1-p^2/2)^t|E| < 2$. We can bound the sum from these terms by $O(\ell \cdot \log(100pn))$.
- $p^2(1-p^2/2)^t|E| \in [2^i, 2^{i+1})$ for some integer $1 \leq i \leq \log(p^2|E|)$. The number of such terms is $O(1/p^2)$, and each term contributes $O(\frac{2^i}{i} \cdot \log(100pn))$ to the total sum. Hence, in total, these terms contribute $O(\frac{1}{p^2} \cdot \frac{2^i}{i} \cdot \log(100pn))$.

The total contribution of the second case above can be written as

$$O\left(\sum_{i=1}^{\lfloor\log(p^2|E|)\rfloor} \frac{1}{p^2} \cdot \frac{2^i}{i} \cdot \log(100pn)\right).$$

It is a simple exercise to verify $\sum_{i=1}^{q} \frac{2^i}{i} = O(\frac{2^q}{q})$, so the above can be bounded by

$$O\left(\frac{|E|}{\log\max(2, p^2|E|)} \cdot \log(100pn)\right).$$

■

Proof of Lemma 11. The bound on the expected value of $|E \setminus K|$ follows by repeatedly applying Lemma 18 ℓ times, and the expected cost bound follows from Lemma 22. ■

Proof of Corollary 12. We apply Lemma 11 with parameter $\ell = \lceil \frac{2}{p^2}(\ln m + \ln \frac{1}{\delta}) \rceil$. The expected cost easily follows. To bound the error probability, note that

$$\mathbf{E}[|E \setminus K|] \leq (1-p^2/2)^\ell |E| \leq (1-p^2/2)^{2/p^2(\ln m + \ln\frac{1}{\delta})} \cdot m = \frac{\delta}{m} \cdot m = \delta.$$

Hence, by Markov's inequality, $\mathbf{Pr}[|E \setminus K| \geq 1] \leq \delta$, so $\mathbf{Pr}[E = K]$ (i.e., the algorithm finds all edges) is at least $1 - \delta$. ■

Proof of Corollary 13. We first apply Lemma 11 with parameter $\ell = \lceil \frac{2\ln(n^2/m)}{p^2} \rceil$. The expected number of undiscovered edge m_u afterwards is

$$O(m \cdot (1 - p^2/2)^{\ell}) = O(m^2/n^2),$$

and the expected cost is

$$O\left(\left(\frac{|E|}{\log\max(2, p^2|E|)} + \frac{\log(n^2/m)}{p^2}\right) \cdot \log(100pn)\right).$$

Afterwards, we run Lemma 7 to reconstruct the whole graph. The number of queries used by Lemma 7 is

$$O((n + m_u) \cdot \log \frac{n^2}{n + m_u}).$$

By Jensen's inequality, the expected number of queries is

$$O\left((n + \mathbf{E}[m_u]) \cdot \log \frac{n^2}{n + \mathbf{E}[m_u]}\right) = O\left((n + m^2/n^2) \cdot \log \frac{n^2}{n + m^2/n^2}\right).$$

■

Appendix C. Deferred Calculations from the Adaptive Algorithm

Reconstruct edges inside $S_i \cup S_{i+1}$. Here, we further consider two different cases, depending on whether $i = 1$ or not.

- Reconstruct edges inside $S_1 \cup S_2$. Here, we run Corollary 12 with parameters $V = S_1 \cup S_2, m, D = 2T_2 = \Theta(m^{0.35})$ and $\delta = 1/100$. The success rate is $1 - \delta = 0.99$, and the expected query complexity is

$$O\left(\left(\frac{|E|}{\log\max(2, p^2|E|)} + \frac{\log m + \log\frac{1}{\delta}}{p^2}\right) \cdot \log(100pn)\right)$$

for $p = \Theta((mD)^{-1/3}) = \Theta(m^{-0.45})$. The query complexity simplifies to

$$O\left(\frac{m}{\log(m^{0.1})} + \frac{\log m}{m^{-0.9}} \cdot \log(100pn)\right) = O\left(\frac{m\log n}{\log m}\right).$$

- Reconstruct edges inside $S_i \cup S_{i+1}$ for $i \geq 2$. In this case, we run Corollary 13 with parameters $V = S_i \cup S_{i+1}, m, D = 2T_{i+1}$. Note that as the minimum degree of vertices in $S_i \cup S_{i+1}$ is

$\Omega(T_{i-1})$, $|S_i \cup S_{i+1}| = O(m/T_{i-1})$. Let $e_{i,i+1}$ denote the number of edges in $G[S_i \cup S_{i+1}]$. Then, the expected number of queries used by Corollary 13 is

$$\begin{aligned}
&O\Bigg(\left(\frac{e_{i,i+1}}{\log\max(2, p^2\cdot e_{i,i+1})} + \frac{\log((m/T_{i-1})^2/m)}{p^2}\right)\cdot\log(100p\cdot(m/T_{i-1})) \\
&\quad +(m/T_{i-1} + m^2/(m/T_{i-1})^2)\cdot\log\left(\frac{(m/T_{i-1})^2}{m/T_{i-1} + m^2/(m/T_{i-1})^2}\right)\Bigg) \\
&= O\left(\left(\frac{e_{i,i+1}}{\log\max(2, p^2\cdot e_{i,i+1})} + \frac{\log(m/T_{i-1}^2)}{p^2}\right)\cdot\log(100pm/T_{i-1}) + T_{i-1}^2\cdot\log\left(\frac{m}{T_{i-1}^2}\right)\right),
\end{aligned}$$

where $p = \Theta((mT_{i+1})^{-1/3})$. Note that to bound the second term, we used the fact that $T_{i-1} = \Omega(m^{1/3})$ and so $T_{i-1}^2 = \Omega(m/T_{i-1})$. By plugging in the value of p and $T_{i-1} = \sqrt{m/\alpha_{i-1}}$, $T_{i+1} = \sqrt{m/\alpha_{i+1}}$, the above simplifies to

$$\begin{aligned}
&O\left(\left(\frac{e_{i,i+1}}{\log\max(2, \frac{\alpha_{i+1}^{1/3}}{m}\cdot e_{i,i+1})} + \frac{m\log(\alpha_{i-1})}{\alpha_{i+1}^{1/3}}\right)\cdot\log(\alpha_{i+1}^{1/6}\alpha_{i-1}^{1/2}) + m\cdot\frac{\log(\alpha_{i-1})}{\alpha_{i-1}}\right) \\
=&O\left(\left(\frac{e_{i,i+1}}{\log\max(2, \frac{\alpha_{i+1}^{1/3}}{m}\cdot e_{i,i+1})}\right)\cdot\log(\alpha_{i+1}) + m\cdot\left(\frac{\log^2(\alpha_{i+1})}{\alpha_{i+1}^{1/3}} + \frac{\log(\alpha_{i-1})}{\alpha_{i-1}}\right)\right).
\end{aligned}$$

When $e_{i,i+1} \le m/\alpha_{i+1}^{0.1}$, the first term can be bounded by $O((m/\alpha_{i+1}^{0.1})\cdot\log(\alpha_{i+1})) = O(m/\alpha_{i+1}^{0.01})$. Otherwise, the first term can be bounded by $O(e_{i,i+1})$. Therefore, the above can be upper bounded by

$$O\left(e_{i,i+1} + m\cdot\left(\frac{1}{\alpha_{i+1}^{0.01}} + \frac{\log^2(\alpha_{i+1})}{\alpha_{i+1}^{1/3}} + \frac{\log(\alpha_{i-1})}{\alpha_{i-1}}\right)\right) = O\left(e_{i,i+1} + \frac{m}{\alpha_{i+1}^{0.01}}\right).$$

Summing over all $i \ge 2$, the total expected query complexity can hence be bounded by

$$\sum_{i=2}^{\ell-1} e_{i,i+1} + m\sum_{i=3}^{\ell}\alpha_i^{-0.01} = O(m) = O\left(\frac{m\log n}{\log m}\right)$$

where we have used the fact that $\sum_{i=3}^{\ell}\alpha_i^{-0.01} = \Theta(1)$.

Reconstruct edges between S_i and S_j for $1 \le i \le j-2$. Recall the maximum degree of vertices in S_i is $2T_i$ and we have computed all edges in $G[S_i]$ by the previous case. Hence, we can apply Lemma 8 with $V = S_i \cup S_j, U = S_i, D = 2T_i$ to compute all adjacent edges between any given $v \in S_j$ and S_i using

$$O\left((T_i + \deg(v, S_i))\cdot\frac{\log(|S_i|/T_i + 1)}{\log(\max(2, \deg(v, S_i)/T_i))}\right)$$

queries. Here, we also need to consider two cases, depending on whether $i = 1$ or not.

- $i = 1$. In this case, we simply use n to upper bound $|S_i|$, and the query complexity simplifies to
$$O\left((T_1 + \deg(v, S_1)) \cdot \frac{\log n}{\log(\max(2, \deg(v, S_1)/T_1))}\right).$$
We can further use
$$O\left(T_1 \log n + \max(\deg(v, S_1), T_2) \cdot \frac{\log n}{\log(T_2/T_1))}\right)$$
to upper bound the above since the second term in the sum is an increasing function of $\deg(v, S_1)$. The above is now upper bounded by
$$O\left((T_2 + \deg(v, S_1)) \cdot \frac{\log n}{\log m}\right),$$
since $T_2/T_1 = \Theta(m^c)$ for a constant $c > 0$. Next, we sum the above bound over all $v \in \cup_{j=3}^{\ell} S_j$. Note that for every such v, $\deg(v) \geq \Omega(T_2)$, and so the number of such v is bounded by $|\cup_{j=3}^{\ell} S_j| \leq O(m/T_2)$. Hence, the sum of the above bound over all $v \in \cup_{j=3}^{\ell} S_j$ is bounded by
$$\left(\sum_{v \in \cup_{j=3}^{\ell} S_j} (T_2 + \deg(v, S_1))\right) \cdot O\left(\frac{\log n}{\log m}\right) \leq O\left(\frac{m \log n}{\log m}\right).$$

- $i \geq 2$. In this case, we use $O(m/T_{i-1})$ to bound $|S_i|$, as all vertices in S_i have degree $\Omega(T_{i-1})$. Hence, the query complexity for a fixed v simplifies to
$$O\left((T_i + \deg(v, S_i)) \cdot \frac{\log((m/T_{i-1})/T_i)}{\log(\max(2, \deg(v, S_i)/T_i))}\right).$$
We can further use
$$O\left(T_i \cdot \log((m/T_{i-1})/T_i) + \max(T_{i+1}/\alpha_i^{0.01}, \deg(v, S_i)) \cdot \frac{\log((m/T_{i-1})/T_i)}{\log((T_{i+1}/\alpha_i^{0.01})/T_i))}\right)$$
to upper bound the above (by considering whether $\deg(v, S_i) \leq T_{i+1}/\alpha_i^{0.01}$ or not). This further simplifies to
$$O\left(T_i \cdot \log(\alpha_{i-1}^{1/2}\alpha_i^{1/2}) + (T_{i+1}/\alpha_i^{0.01} + \deg(v, S_i)) \cdot \frac{\log(\alpha_{i-1}^{1/2}\alpha_i^{1/2})}{\log(\alpha_i^{1/2-0.01}/\alpha_{i+1}^{1/2})}\right)$$
$$= O\big(T_{i+1}/\alpha_i^{0.01} + \deg(v, S_i)\big).$$
since $T_i = T_{i+1}/\alpha_i^{0.05}$, implying that $T_i \cdot \log(\alpha_{i-1}^{1/2}\alpha_i^{1/2}) = o(T_{i+1}/\alpha_i^{0.01})$. Finally, note that every vertex in $\cup_{j=i+2}^{\ell} S_j$ has degree $\Omega(T_{i+1})$ and so $| \cup_{j=i+2}^{\ell} S_j| = O(m/T_{i+1})$. The total

query complexity over all $i \geq 2$ and all $v \in S_j$ for $j \geq i+2$ is thus

$$O\left(\sum_{2\leq i\leq \ell-2}\ \sum_{v\in\bigcup_{j=i+2}^{\ell} S_j} \left(T_{i+1}/\alpha_i^{0.01} + \deg(v, S_i)\right)\right)$$
$$= O\left(\sum_{2\leq i\leq \ell-2} \left|\cup_{j=i+2}^{\ell} S_j\right| \cdot T_{i+1}/\alpha_i^{0.01} + \sum_{2\leq i\leq \ell-2}\ \sum_{v\in\bigcup_{j=i+2}^{\ell} S_j} \deg(v, S_i)\right)$$
$$= O\left(\sum_{2\leq i\leq \ell-2} m/\alpha_i^{0.01} + m\right) = O(m).$$

Appendix D. Deferred Proofs for the Two Round Algorithm of Section 4

In this section we provide proofs of correctness for the two subroutines used in our two-round algorithm.

D.1. Approximating the Degree of a Vertex

Proof of Theorem 14. Our estimator works by checking how likely it is for a random set of a given size to intersect the neighborhood of u. We first observe that this event can be checked with two CC-queries.

Fact 23 *Given $u \in V$ and $S \subseteq V \setminus u$ we have*

$$\mathsf{CC}(S \cup \{u\}) - \mathsf{CC}(S) = \mathbf{1}(\deg(u, S) = 0),$$

i.e. two CC-queries suffice to check if S contains a neighbor of u.

Proof The fact follows by observing that u has a neighbor in S if and only if adding u to S does not increase the number of connected components. ■

The algorithm is described in Algorithm 3, and we now prove its correctness. The inner for-loop from Line 3 to Line 6 computes an empirical estimate for the probability of a random set of 2^p vertices being disjoint from the neighborhood of u. This probability is exactly

$$\mathbf{Pr}_{S:\ |S|=2^p}\left[\deg(u, S) = 0\right] = \left(1 - \frac{\deg_G(u)}{n-1}\right)^{2^p}. \tag{3}$$

We will argue that with probability $1-\varepsilon$, the value $p^\star$ computed in Line 8 satisfies $2^{p^\star} \in \left(\frac{n-1}{2\deg_G(u)}, \frac{2(n-1)}{\deg_G(u)}\right]$, and therefore the output in Line 9 satisfies $\widetilde{d}_u \in [\deg_G(u), 4\deg_G(u)]$ as desired. This claim holds due to the following Claim 24, which completes the proof. ■

Claim 24 *Consider the random variables Z_p and $X_j^{(p)}$ defined in Algorithm 3. With probability $1-\varepsilon$ the following hold.*

Algorithm 3: Non-adaptive degree estimator

```
Input: $n \in \mathbb{N}, \varepsilon \in (0,1)$, CC-query access to a graph $G = (V, E)$ with $n$ nodes, and $u \in V$.
Output: $\widetilde{d}_u \in \mathbb{N}$ satisfying $\widetilde{d}_u \in [\deg_G(u), 4\deg_G(u)]$ with probability $1 - \varepsilon$.
Let $\ell = \lceil 450 \ln(800/\varepsilon^2) \rceil$;
for $p = 1, 2, \dots, \lceil \log(n-1) \rceil$ do
    for $j = 1, \dots, \ell$ do
        Draw random set $S$ containing $2^p$ iid uniform samples from $V \setminus \{u\}$, and compute the
          Boolean random variable $X_j^{(p)} := \mathbf{1}(\deg(u, S) = 0)$ by making two CC-queries
          (Fact 23);
        Let $Z_p := \frac{1}{\ell} \sum_{j \le \ell} X_j^{(p)}$;
    end
end
Let $p^\star$ be the largest $p \in [\log n]$ for which $Z_p > \frac{1}{2e}$;
return $\widetilde{d}_u = \frac{2(n-1)}{2^{p^\star}}$;
```

- *For p such that $2^p \in (\frac{n-1}{2\deg_G(u)}, \frac{n-1}{\deg_G(u)}]$, we have $Z_p > \frac{1}{2e}$.*
- *For every p such that $2^p > \frac{2(n-1)}{\deg_G(u)}$ we have $Z_p < \frac{1}{2e}$.*

Proof Define μ_p to be the quantity from Equation (3) and observe that $\mathbf{E}[Z_p] = \mathbf{E}[X_j^{(p)}] = \mu_p$. We show that the two events described in the statement of the claim each fail to hold with probability at most $\varepsilon/2$ and the claim then follows by a union bound.

Consider the first bullet point. Let p be such that $2^p \in (\frac{n-1}{2\deg_G(u)}, \frac{n-1}{\deg_G(u)}]$ and observe that $\mu_p \ge 1/e$. Recall that $Z_p = \frac{1}{\ell} \sum_{j \le \ell} X_j^{(p)}$ and $X_j^{(p)} = \mathbf{1}(\deg(u, S) = 0)$. We have

$$\mathbf{Pr}_{Z_p}[Z_p \le 1/2e] \le \mathbf{Pr}_{Z_p}[|Z_p - \mu_p| > 1/2e] \le 2\exp\left(-2 \cdot \frac{\ell}{4e^2}\right) \le \varepsilon/2 \tag{4}$$

where the first inequality follows from our lower bound on μ_p, the second inequality is due to Hoeffding's bound, and the third inequality is because $\ell \ge 2e^2 \ln(4/\varepsilon)$.

We now handle the second bullet point. We first handle p for which 2^p is significantly larger than the target interval. If $2^p \ge t \cdot \frac{n-1}{\deg_G(u)}$, then $\mu_p \le \exp(-t)$ and so by Markov's inequality $\mathbf{Pr}[Z_p \ge 1/2e] \le 2e \cdot \exp(-t)$. Therefore, by a union bound over all p such that $2^p > \ln(100/\varepsilon) \cdot \frac{n-1}{\deg_G(u)}$, we have

$$\mathbf{Pr}\left[\exists p \text{ for which } 2^p > \ln(100/\varepsilon) \cdot \frac{n-1}{\deg_G(u)} \text{ and } Z_p \ge 1/2e\right] \le 2e \sum_{q\colon 2^q > \ln(100/\varepsilon)} \exp(-2^q) < \varepsilon/4. \tag{5}$$

Next we handle values of p such that $\frac{2(n-1)}{\deg_G(u)} < 2^p \le \ln(100/\varepsilon) \cdot \frac{n-1}{\deg_G(u)}$, of which there are only $\log(\ln(100/\varepsilon))$. For such a value we have $\mu_p < 1/e^2$ and so using Hoeffding's bound we have

$$\mathbf{Pr}[Z_p \ge 1/2e] \le \mathbf{Pr}[|Z_p - \mu_p| > 1/30] \le 2\exp\left(-2 \cdot \frac{\ell}{900}\right) \le \frac{\varepsilon^2}{400} \le \frac{\varepsilon}{4\log(\ln(100/\varepsilon))}, \tag{6}$$

where the first inequality used the upper bound on μ_p, the second inequality used Hoeffding's bound, and the second to last inequality used $\ell > 450 \ln(800/\varepsilon^2)$. The final inequality is simply due to $4 \log \ln(100/\varepsilon) < \frac{400}{\varepsilon}$ for all $\varepsilon > 0$. Taking a union bound over all p satisfying $\frac{2(n-1)}{\deg_G(u)} < 2^p \leq \ln(100/\varepsilon) \cdot \frac{n-1}{\deg_G(u)}$ completes the proof. ■

D.2. Learning the Neighborhood of a Vertex

Proof of Theorem 15. Consider the Boolean vector $x^u \in \{0,1\}^n$ for which $x^u_j = 1$ iff the j-th vertex in V is a neighbor of u (under an arbitrary labeling of the vertices). We reduce to the problem to recovering the support of x^u using OR queries. Given $S \subseteq V \setminus \{u\}$, such a query is defined as

$$\mathsf{OR}_S(x^u) = \bigvee_{j \in S} x^u_j$$

By Fact 23, we can use two CC-queries to the graph to simulate one OR-query to the vector x_u. In particular, given $S \subseteq V \setminus u$, we have

$$\mathsf{CC}(S \cup \{u\}) - \mathsf{CC}(S) = \mathbf{1}(\deg(u,S) = 0) = \mathsf{OR}_S(x^u).$$

Theorem 15 is now an immediate corollary of the following standard group testing lemma, whose proof can be found, for example, in Black et al. (2024). ■

Lemma 25 (Black et al. (2024), Lemma 1.9) *Let $v \in \{0,1\}^n$ and let $\mathsf{supp}(v) = \{j \in [n] \colon x_j = 1\}$. Given n, d, α, there is a non-adaptive algorithm that makes $O(d \log(n/\alpha))$ OR queries and if $|\mathsf{supp}(v)| \leq d$, returns $\mathsf{supp}(v)$ with probability $1 - \alpha$, and otherwise certifies that $|\mathsf{supp}(v)| > d$ with probability 1.*