Graph Reconstruction with a Connected Components Oracle

Juha Harviainen
Department of Computer Science, University of Helsinki, Finland

Pekka Parviainen
Department of Informatics, University of Bergen, Norway

Abstract

In the Graph Reconstruction (GR) problem, the goal is to recover a hidden graph by utilizing some oracle that provides limited access to the structure of the graph. The interest is in characterizing how strong different oracles are when the complexity of an algorithm is measured in the number of performed queries. We study a novel oracle that returns the set of connected components (CC) on the subgraph induced by the queried subset of vertices. Our main contributions are as follows:

1. For a hidden graph with n vertices, m edges, maximum degree Δ, and treewidth k, GR can be solved in $\mathcal{O}(\min\{m/\log m, \Delta^2, k^2\} \cdot \log n)$ CC queries by an adaptive randomized algorithm.
2. For a hidden graph with n vertices and degeneracy d, GR can be solved in $\mathcal{O}(d^2 \log^2 n)$ CC queries by an adaptive randomized algorithm.
3. There are hidden graphs with n vertices, m edges, maximum degree Δ, treewidth k, and degeneracy d such that $\Omega(m)$, $\Omega(\Delta^2)$, $\Omega(k^2)$, and $\Omega(d^2)$ CC queries are required for solving GR.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases graph reconstruction, parameterized complexity, query complexity

Digital Object Identifier 10.4230/LIPIcs.WG.2026.24

Related Version *Extended Version*: https://arxiv.org/abs/2509.05002

Funding *Juha Harviainen*: Co-funded by the Research Council of Finland, Grant 351156. Co-funded by the European Union (ERC, SCALEBIO, 101169716). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.
Pekka Parviainen: This work was supported by L. Meltzers Høyskolefond.

1 Introduction

The Graph Reconstruction (GR) problem of discovering the structure of an unknown graph has been investigated extensively from both theoretical [6, 18] and practical [11, 16, 17, 19] point of view with applications to, for example, bioinformatics [24]. There, we are given the set of vertices of a *hidden graph* and access to an *oracle* that can be *queried* to obtain information about the structure of the graph.

The GR problem has also been studied within the machine learning community. Specifically, structure learning of probabilistic graphical models [17] is a special case of GR. Probabilistic graphical models are representations of multivariate probability distributions. They use a graph structure to express dependencies among variables. The vertices of the graph correspond to the random variables of the model, and the relationships between variables are encoded by the edge structure of a graph. In structure learning, one is given samples from the (unknown) distribution and the goal is to reconstruct the graph structure of the data-generating distribution. One approach for learning this structure is then to utilize statistical conditional independence tests that state whether two variables are conditionally

independent of each other when conditioned by some subset of other variables. Essentially, these tests can be seen as calls to a specific oracle depending on which particular graphical model is in question. For Markov networks, where the graph is undirected, one gives the oracle a triple (u, v, S) and the oracle outputs whether the corresponding two vertices u and v would be in different connected components if the vertices for the variables S were to be removed. With Bayesian networks, whose structure is a directed acyclic graph, the independencies are captured by a more involved structural notion of d-separation. Ideally, one would like to discover the structure of a model using only a moderate number of queries, possibly under constraints on the size of the queries or the structure of the hidden graph [11, 16, 17, 19].

The query complexity of GR has been studied for varying oracles and families of graphs. One main line of research has focused on the independent set (IS) oracle that only reports whether the queried subset is an independent set. This oracle has been applied to learning matchings [4, 8], cycles [13], stars [3], cliques [3], and general graphs [1, 6]. Other studied oracles include betweenness [2, 26], distance [7, 15, 21, 25], edge counting [10, 14, 24], shortest path oracles [24, 28], and connected component counting oracle [9].

From an information-theoretical point of view, many of the previous oracles give only few bits of information per query, and thus struggle in reconstructing larger families of graphs in a small number of queries. Konrad et al. [18] therefore recently proposed a more powerful oracle that returns a maximal independent set. Notably, their algorithms are able to reconstruct hidden graphs with bounded maximum degree in a logarithmic number of queries in the number of vertices because of the increased amount of information the oracle provides. The lower bounds for the query complexity with the oracle were later refined by Michel and Scott [22].

In this paper, we study the connected components (CC) oracle, which returns the sets of vertices in each connected component of the subgraph induced by the queried set. This is thus a stronger version of the connected component counting (#CC) oracle of Black et al. [9], which only returns the number of the connected components. One can also see the CC oracle as a stronger version of the separation oracle used in learning Markov networks[1]. We then investigate how powerful this CC oracle is as a function of classical graph parameters such as the maximum degree, the *treewidth*, and the *degeneracy* of the hidden graph, when the complexity is measured in terms of the number of performed queries.

Desired properties of an algorithm. There are several desirable properties for the algorithms, of which we focus on *determinism* and *non-adaptivity* similarly to previous works. An algorithm is deterministic if it does not utilize randomness, and otherwise it is *randomized*. A deterministic algorithm should always give the correct output whereas a randomized algorithm has to succeed with *high probability*, that is, the probability should tend to 1 as the input size increases. Randomized algorithms can often be made deterministic by *derandomization* at the expense of slightly increased complexity by, for example, deterministically constructing a family of query sets with some combinatorial properties.

Adaptivity refers on whether the performed queries can depend on the outcomes of the previous queries; a non-adaptive algorithm thus first constructs a set of (possibly randomized) queries, then performs all of them, and finally tries to recover the hidden graph based on the outcomes. The notion of having r *rounds of adaptivity* gives a more granular measure

[1] To answer a $\mathrm{Sep}(a, b, S)$ query, one just conducts a $\mathrm{CC}(V \setminus S)$ query and check whether a and b are in the same connected component. Essentially, this means that a $\mathrm{CC}(V \setminus S)$ query answers to $\mathrm{Sep}(a, b, S)$ queries for all pairs $a, b \in V \setminus S$.

Table 1 Summary of our upper bounds. The first corollary follows directly from Black et al. [9].

Theorem	Complexity	Randomized	Adaptive	Comments
Corollary 2	$\mathcal{O}\big(m\log(n)/\log(m)\big)$	no	yes	-
Theorem 3	$\mathcal{O}(\Delta^2\log n)$	yes	no	known Δ
Theorem 10	$\mathcal{O}\big(\Delta^3\log(n/\Delta)\big)$	no	no	known Δ, existence result
Theorem 11	$\mathcal{O}\big(\Delta^2\log n\big)$	yes	yes	-
Theorem 11	$\mathcal{O}\big(\Delta^3\log(n/\Delta)\big)$	no	yes	existence result
Theorem 19	$\mathcal{O}(k^2\log n)$	yes	yes	-
Theorem 19	$\mathcal{O}(k^3\log n)$	no	yes	existence result
Theorem 23	$\mathcal{O}(d^2\log^2 n)$	yes	yes	-
Theorem 23	$\mathcal{O}\big(d^3\log n\log(n/d)\big)$	no	yes	existence result

of adaptivity, where we are allowed to construct r sets of queries such that the queries in the ith set can depend on the outcomes of the queries in the previous sets. Without any prior knowledge about the properties of the graph, non-adaptive algorithms tend to need a polynomial number of queries in the number of vertices, since the constructed set of queries needs to work correctly (with high probability) for all possible graphs.

Our contributions. We start by devising several algorithms for the GR problem for graphs of n vertices with a CC oracle. A summary of the upper bounds is shown in Table 1. We present results parameterized by four graph parameters: number of edges m, maximum degree Δ, treewidth k, and degeneracy d. With all these parameters, we provide adaptive algorithms (both deterministic and randomized) whose query complexity is polynomial with respect to the parameter and (poly)logarithmic with respect to the number of vertices. When the maximum degree of the hidden graph is known, we also provide a non-adaptive algorithm. The algorithms for m, Δ, and k can be combined, giving a randomized adaptive algorithm that uses at most $\mathcal{O}(\min\{m/\log m, \Delta^2, k^2\}\cdot\log n)$ queries (Theorem 20). Our algorithm for graphs with bounded degeneracy uses $\mathcal{O}(d^2\log^2 n)$ queries, so it is mildly worse in terms of n. However, the degeneracy of a graph is always the smallest one of all the considered parameters, and so this algorithm is applicable to many graph families for which the other algorithms require polynomially many queries in n. For example, planar graphs have degeneracy at most five [20] but can have unbounded maximum degree and treewidth.

We also study lower bounds; a summary is shown in Table 2. Black et al. [9] showed that it is not possible to design efficient non-adaptive algorithms for the #CC queries, and we note that their result applies also for CC queries. In other words, non-adaptive algorithms require $\Omega(n^2)$ CC queries in the worst case if no additional information is given (Corollary 25), even if we know that the graph has treewidth 2 and thereby has at most $\mathcal{O}(n)$ edges. There is a peculiar observation that very little adaptivity can help. Algorithm 4 has two rounds of adaptivity and query complexity $\mathcal{O}(k^2\log n)$ implying that adding one round of adaptivity reduces the dependence on n from n^2 to $\log n$. Furthermore, we show that no adaptive algorithm can solve the problem for all instances by performing $o(m)$, $o(\Delta^2)$, $o(k^2)$, or $o(d^2)$ CC queries (Theorem 24). Thus, our upper bounds are within a (poly)logarithmic factor in n from the lower bound.

Finally, we provide a comparison of the CC oracle with other oracles. First, we show that getting explicit information about the connected components rather than just their number [9] leads to more efficient algorithms (Theorem 26). Then, we consider the maximal

Table 2 Summary of our lower bounds, where the Complexity column states that there is an instance on which $\Omega(\cdot)$ CC queries are needed. A lower bound for randomized algorithms holds for deterministic algorithms, and a lower bound for adaptive algorithms holds for non-adaptive algorithms. The proof of Corollary 25 is identical to the lower bound proof for the #CC oracle [9].

Theorem	Complexity	Randomized	Adaptive	Comments
Theorem 24	$\Omega(m)$	yes	yes	-
Theorem 24	$\Omega(k^2)$	yes	yes	-
Theorem 24	$\Omega(\Delta^2)$	yes	yes	-
Theorem 24	$\Omega(d^2)$	yes	yes	-
Corollary 25	$\Omega(n^2)$	yes	no	even if $k = 2$ and $m = \mathcal{O}(n)$

independent set (MIS) oracle introduced by Konrad et al. [18]. While there are instances where the MIS oracle needs $\mathcal{O}(n)$ queries but $\mathcal{O}(\log n)$ queries are sufficient for the CC oracle (Theorem 27), these two oracles seem to be roughly equally powerful in the sense that both upper and lower bounds for m and Δ are nearly the same for both types of oracles.

2 Notation and Preliminaries

An *undirected graph* is a pair $G = (V(G), E(G))$ where $V(G)$ is the vertex set of G and $E(G)$ is the edge set of G. If there is no ambiguity about which graph we refer to, we simply write $G = (V, E)$. An edge between vertices u and v is denoted by uv; note that edges are undirected and ordering does not matter, that is, uv and vu are the same edge. We use $n = |V(G)|$ to denote the number of vertices and $m = |E(G)|$ to denote the number of edges in a graph. If there is no edge between vertices u and v, we call uv a *non-edge*.

Vertex u is a neighbor of v in G if there is an edge $uv \in E(G)$. We denote the set of neighbors in G by $N_G(v)$ or simply by $N(v)$ if the graph G is clear from the context. The degree of a vertex v is the number of its neighbors. The maximum degree Δ of a graph is the maximum out of the degrees of its vertices.

A graph H is a *subgraph* of G if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. A graph J is a *supergraph* of G if $V(G) \subseteq V(J)$ and $E(G) \subseteq E(J)$. In this paper, all used supergraphs J of G have $V(J) = V(G)$.

The subgraph of G *induced by* $S \subseteq V$, denoted by $G[S]$, has the vertex set S and the edge set $\{uv \in E(G) \colon u, v \in S\}$. A subset of vertices is an *independent set* if the subgraph induced by it has no edges. An independent set is *maximal* if none of its proper supersets are independent. The *degeneracy* of a graph $d = d(G)$ is the smallest integer such that every induced subgraph has at least one vertex with degree at most d.

A *connected components* (CC) *oracle* is a black box subroutine that is given a subset S of vertices and returns the set of connected components of $G[S]$, that is, a partition of S such that each subset is a connected component.

In the Graph reconstruction problem, we are given a vertex set and an oracle, and asked to perform queries on the oracle to reconstruct the edge structure of the graph:

Graph reconstruction (GR)
Input: A vertex set V, an oracle O
Question: What is the edge set E of the hidden graph $G = (V, E)$?

In this paper, we focus on using a CC oracle as the oracle and use the number of performed queries to measure the complexity of the algorithms for the GR problem.

Throughout the paper, we will denote by G^+ the supergraph of G, which is initialized to a complete graph and from which edges are removed as the queries reveal more information.

Treewidth. A *tree* is a connected graph with no cycles. A *tree decomposition* of a graph G is a pair $(T, \mathcal{X})$, where T is a tree and $\mathcal{X} = \{X_t : t \in V(T)\}$ a family of subsets of $V(G)$ called *bags* with the following properties:

1. For every vertex $v \in V(G)$, there is a bag X_t with $v \in X_t$;
2. For every edge $uv \in E(G)$, there is a bag X_t with $u, v \in X_t$; and
3. For every vertex $v \in V(G)$, the subgraph of T induced by its vertices t with $v \in X_t$ is connected.

The *width of a tree decomposition* is the size of its largest bag minus one. The *treewidth of a graph* is the smallest width among its tree decompositions. We will denote the treewidth of the hidden graph G by k. A maximal graph with treewidth k is called a *k-tree*.

A pair of sets of vertices (A, B) with $A \cup B = V(G)$ is called a *separation* if there are no edges between $A \setminus B$ and $B \setminus A$. The set $A \cap B$ is then called a *separator* and its *order* is $|A \cap B|$. In other words, removal of the separator $A \cap B$ would partition the graph into two separate connected components $G[A \setminus B]$ and $G[B \setminus A]$. A separator S is *α-balanced* if the size of every connected component in $G[V \setminus S]$ is at most $\alpha \cdot |V(G)|$.

Let $f(n)$ be a function from positive integers to nonnegative real numbers. Recall that $f(n) = o(g(n))$ for some function g if $f(n)/g(n)$ tends to 0 as n tends to infinity, and $f(n) = \omega(g(n))$ if $f(n)/g(n)$ tends to infinity.

The following lemma will be useful later. Various versions of it have appeared in the literature for constructing sets that include and exclude subsets of elements.

▶ **Lemma 1.** *Let V be a set and let $U, W \subseteq V$ its disjoint subsets of size q and p, respectively. Suppose that we sample a set $Q \subseteq V$ such that we independently include each $v \in V$ in Q with probability $1/(ap+b)$, where $a > 0$ and $b \geq 0$ are constants with $ap + b > 1$ for all $p \geq 1$. Then, the probability that all elements of U are in Q and none of the elements of W are in Q is at least $\left(\frac{1}{ap+b}\right)^q \cdot \min\{\mathrm{e}^{-1/a}, \mathrm{e}^{-1/(a+b-1)}\}$. In particular, the probability is $\Omega(1/p^q)$ for constant q.*

Proof. We have that

$$\begin{aligned}
\Pr(U \subseteq Q \text{ and } W \cap Q = \emptyset) &= \left(\frac{1}{ap+b}\right)^q \left(1 - \frac{1}{ap+b}\right)^p \\
&\geq \left(\frac{1}{ap+b}\right)^q \cdot \left(\mathrm{e}^{-1/(ap+b-1)}\right)^p \\
&= \left(\frac{1}{ap+b}\right)^q \cdot \mathrm{e}^{-p/(ap+b-1)},
\end{aligned}$$

where the inequality follows from the fact that $1 - 1/x \geq \mathrm{e}^{-1/(x-1)}$ for every $x \geq 1$. The derivative of $-p/(ap+b-1)$ with respect to p is $(1-b)/(ap+b-1)^2$, which is zero if and only if $b = 1$, in which case $\mathrm{e}^{-p/(ap+b-1)}$ is e^{-a} everywhere. For other values of b, $\mathrm{e}^{-p/(ap+b-1)}$ is minimized either when $p = 1$ or p tends to infinity, corresponding to values $\mathrm{e}^{-1/(a+b-1)}$ and $\mathrm{e}^{-1/a}$, respectively. ◀

3 Algorithms

In this section, we will provide upper bounds for GR with a CC oracle.

3.1 Bounded Number of Edges

Recently, Black et al. [9] showed an algorithm utilizing an oracle that returns the number of connected components for the subgraph induced by the query set S can reconstruct any hidden graph on n vertices and m edges with $\mathcal{O}\big(m \log(n)/\log(m)\big)$ queries. Since our oracle can trivially simulate theirs, the same upper bound follows.

▶ **Corollary 2.** *There is an adaptive deterministic algorithm that solves* GR *with* $\mathcal{O}\left(\frac{m \log(n)}{\log(m)}\right)$ *CC queries.*

3.2 Bounded Maximum Degree

For graphs with a bounded maximum degree Δ, we follow the strategy of Konrad et al. [18] of performing queries on randomized subsets of vertices. The main idea is that if there is no edge between u and v, both vertices are included in a query set, and none of the neighbors of u are included, then the CC query outputs them in different components.

Algorithm 1 An algorithm for GR for graphs with bounded maximum degree.

Input: A vertex set V, a CC oracle $\mathrm{CC}(\cdot)$, maximum degree Δ
Output: Hidden graph G with high probability

```
Initialize G⁺ to a complete graph on V;
for t = O(Δ² log n) times do
    Sample Q ⊆ V s.t. v ∈ Q with probability 1/(Δ+1) for all v ∈ V independently;
    C ← CC(Q);
    Remove from G⁺ edges uv for which v and u are in different components in C;
return G⁺;
```

▶ **Theorem 3.** *Algorithm 1 is a non-adaptive randomized algorithm that solves* GR *with* $\mathcal{O}(\Delta^2 \log n)$ *CC queries when* Δ *is known. The output of the algorithm is correct with high probability.*

Proof. Consider an arbitrary non-edge uv of G. By Lemma 1, the probability that Q contains u and v but none of the neighbors of u is at least $1/((\Delta+1)^2 \cdot \mathrm{e})$. If this occurs, then the output of the CC query must have v and u in distinct connected components. Let X_i be a random Boolean variable that is true if this occurs at least once for the ith edge over t independently chosen subsets Q, and otherwise false. By the union bound, we have that

$$\Pr\left(\bigwedge_{i=1}^{m} X_i\right) \geq 1 - \sum_{i=1}^{m} \Pr(\overline{X_i}) \geq 1 - m \cdot \left(1 - \frac{1}{(\Delta+1)^2 \cdot \mathrm{e}}\right)^t .$$

We want to choose a value for t such that $m \cdot \big(1 - 1/((\Delta+1)^2 \cdot \mathrm{e})\big)^t \leq 1/n$, which holds if $\big(1 - 1/((\Delta+1)^2 \cdot \mathrm{e})\big)^t \leq 1/n^3$. Since $(1 - 1/x)^x \leq 1/\mathrm{e}$ for $x \geq 1$, choosing $t = \mathcal{O}(\Delta^2 \log n)$ suffices for finding all edges with high probability. ◀

3.2.1 Derandomization

Next, we will derandomize Algorithm 1. Our schemes follow straightforwardly from the one presented by Konrad et al. [18].

In Algorithm 1, we remove an edge uv if u and v are in different connected components returned by the query $\mathrm{CC}(Q)$ for some Q. If u and v are not connected by an edge in G, then there exists a set $\{w_1, \ldots, w_p\}$ such that u and v are in different connected components in $\mathrm{CC}(Q)$ for any Q where $u, v \in Q$ and $w_i \notin Q$ for all i. Following Konrad et al. [18], we define a *witness* as follows.

▶ **Definition 4** (Witness). *Let V be a set of n vertices and $0 \leq p \leq n-2$ be an integer. Then, the tuple $(\{u, v\}, \{w_1, \ldots, w_p\})$ with $u, v, w_1, \ldots, w_p \in V$ being distinct vertices is called a witness, and we denote the set of all witnesses by $\mathcal{W}$.*

The following lemma gives an upper bound for the number of witnesses.

▶ **Lemma 5.** *The number of witnesses $|\mathcal{W}|$ is bounded by*

$$|\mathcal{W}| \quad \leq \quad n^2 \cdot \left(\mathrm{e} \cdot \frac{n-2}{p}\right)^p$$

Proof. We use the bound $\binom{a}{b} \leq (\mathrm{e} \cdot \frac{a}{b})^b$ and get

$$\binom{n}{2}\binom{n-2}{p} \quad \leq \quad n^2 \cdot \left(\mathrm{e} \cdot \frac{n-2}{p}\right)^p. \qquad \blacktriangleleft$$

▶ **Definition 6** (p-Query-Scheme). *Let V be a set of n vertices and $0 \leq p \leq n-2$ be an integer. The set $\mathcal{Q} = \{Q_1, \ldots, Q_\ell\}$ is a p-Query-Scheme of size ℓ if, for every witness $(\{u, v\}, \{w_1, \ldots, w_p\}) \in \mathcal{W}$, there exists a query $Q_i \in \mathcal{Q}$ such that*

1. *$u, v \in Q_i$ and*
2. *$\{w_1, \ldots, w_p\} \cap Q_i = \emptyset$.*

The following lemma gives an upper bound for the size of a p-Query-Scheme.

▶ **Lemma 7.** *There exists a p-Query-Scheme of size $\mathcal{O}(p^3 \log n/p)$.*

Proof. Suppose we have a collection of ℓ sets $\mathcal{Q} = \{Q_1, \ldots, Q_\ell\}$ with $Q_i \subseteq V$ for all i. Suppose further that each vertex has probability $1/(p+1)$ to be in the set Q_i independently for each set. That is, $\Pr(v \in Q_i) = 1/(p+1)$ for all v and i.

Next, we will show that if $\ell = \mathcal{O}(p^3 \log(n/p))$, then the probability that $\mathcal{Q}$ is a p-Query-Scheme is larger than 0.

By Lemma 1, the probability that a set Q_i contains a witness $(\{u, v\}, \{w_1, \ldots, w_p\})$ is at least $1/((p+1)^2 \cdot \mathrm{e})$. Then, the probability that none of the sets in $\mathcal{Q}$ contains the witness $(\{u, v\}, \{w_1, \ldots, w_p\})$ is

$$\begin{aligned} \Pr(\{u, v\} \not\subseteq Q_i \text{ or } \{w_1, \ldots w_p\} \cap Q_i \neq \emptyset \text{ for all } i) \quad &\leq \quad \left(1 - \frac{1}{(p+1)^2} \cdot \frac{1}{\mathrm{e}}\right)^\ell \\ &\leq \quad \exp\left(-\frac{\ell}{(p+1)^2 \cdot \mathrm{e}}\right), \end{aligned}$$

where the latter inequality follows from the fact $1 + x \leq \mathrm{e}^x$.

By union bound, the probability that there exists a witness that is not considered for some pair u and v is at most

$$n^2 \left(\mathrm{e} \cdot \frac{n-2}{p}\right)^p \exp\left(-\frac{\ell}{(p+1)^2 \cdot \mathrm{e}}\right). \tag{1}$$

To guarantee that the probability is less than 1, it suffices to choose $\ell = \Theta(p^3 \log n/p)$. ◀

We are not aware of any efficient explicit constructions for p-Query-Schemes of size $\mathcal{O}(p^3 \log n/p)$, but there is rich literature on closely related pseudorandom objects like *group testing schemes* and *strongly selective families* – see, for example, the work of Porat and Rothschild [23]. At the cost of increased size, (n, κ, κ^2)-*splitters* can be utilized to construct a p-Query-Scheme in polynomial time. Recall that an (n, κ, κ^2)-splitter for the vertices is a family $\mathcal{F}$ of functions from the n vertices to κ^2 colors $\{1, \ldots, \kappa^2\}$ such that for any subset of vertices S of size κ, there is a function that colors the vertices of S with distinct colors.

▶ **Lemma 8** ([5]). *An (n, κ, κ^2)-splitter of size $\kappa^{\mathcal{O}(1)} \log n$ can be constructed in time $\kappa^{\mathcal{O}(1)} n \log n$.*

▶ **Theorem 9.** *A p-Query-Scheme of size $p^{\mathcal{O}(1)} \log n$ can be constructed in time $p^{\mathcal{O}(1)} n \log n$.*

Proof. Let $\kappa = p + 2$ and construct an (n, κ, κ^2)-splitter $\mathcal{F}$ of size $p^{\mathcal{O}(1)} \log n$ in time $p^{\mathcal{O}(1)} n \log n$ by applying Lemma 8. Construct now in time $p^{\mathcal{O}(1)} n \log n$ a family of sets

$$\mathcal{Q} = \left\{ f^{-1}(i) \cup f^{-1}(j) \colon f \in \mathcal{F}, i, j \in \{1, \ldots, \kappa^2\} \right\}$$

by considering each function and each pair of colors, and then including a set of vertices colored in either of the colors to $\mathcal{Q}$. We claim that $\mathcal{Q}$ is a p-Query-Scheme.

Consider any witness $(\{u, v\}, \{w_1, \ldots, w_p\})$. The splitter has a function f that assigns a distinct color to each of the vertices of the witness. Letting, $i = f(u)$ and $j = f(v)$, the set $f^{-1}(i) \cup f^{-1}(j)$ contains the vertices u and v but none of the vertices $w_1, \ldots, w_p$. Since this holds for all witnesses, $\mathcal{Q}$ is a p-Query-Scheme. ◀

By utilizing Δ-Query-Schemes similarly to Konrad et al. [18], we get the following derandomized algorithm.

▶ **Theorem 10.** *There exists a non-adaptive deterministic algorithm that solves* GR *with $\mathcal{O}\big(\Delta^3 \log(n/\Delta)\big)$ CC queries when Δ is known.*

Proof. We claim that it suffices to construct a Δ-Query-Scheme $\mathcal{Q}$ and to perform those $t = \mathcal{O}\big(\Delta^3 \log(n/\Delta)\big)$ queries in Algorithm 1 instead of the randomized queries.

Consider any non-edge uv. By the definition of a Δ-Query-Scheme, there exists a query Q_i that contains both u and v, but none of the neighbors of u. Thus, the oracle returns u and v in a separate component and the edge uv gets removed from the supergraph G^+. ◀

3.2.2 Unknown Δ

By making the algorithms adaptive, we can solve GR with the asymptotically same number of queries even when Δ is not given to us beforehand by following the strategy of Konrad et al. [18]. Essentially, we guess an upper bound for Δ, and when we get evidence that it is incorrect, we double the value of the upper bound.

▶ **Theorem 11.** *There exist an adaptive randomized algorithm and an adaptive deterministic algorithm that solve* GR *with $\mathcal{O}\big(\Delta^2 \log n\big)$ and $\mathcal{O}\big(\Delta^3 \log(n/\Delta)\big)$ CC queries, respectively. The output of the randomized algorithm is correct with high probability.*

Proof. We start by guessing that $D = 1$ is an upper bound for Δ, and then run Algorithm 1 or the algorithm from Theorem 10 as if the known value of Δ were $D + 1$. If D is smaller than Δ, then the graph outputted by that algorithm has a vertex whose degree exceeds D by the construction of the query sets. In this case, we double the value of D and run the algorithm again as if Δ were $D + 1$, repeating the process $\mathcal{O}(\log \Delta)$ times.

Each round of adaptivity performs $\mathcal{O}\big(D^3 \log(n/D)\big)$ queries. Since D is doubled each round, the total number of queries is dominated by the last round, resulting in the total number of queries being $\mathcal{O}\big(\Delta^3 \log(n/\Delta)\big)$.

For the randomized algorithm, it remains to show that the output is correct with high probability. Recall that Algorithm 1 succeeds with probability at least $1 - 1/n$ and note that the output is correct if it succeeds on all $\mathcal{O}(\log \Delta)$ rounds of adaptivity. The probability of this is at least $(1 - 1/n)^{\mathcal{O}(\log n)} \approx (1/\mathrm{e})^{\mathcal{O}(\log(n)/n)}$, which tends to 1 as n increases. ◀

Note that the type of adaptivity we used is rather weak: in a sense, we have a fixed (possibly randomized) sequence of queries where our guess for Δ varies, and the adaptivity is only used to decide when to stop performing more queries from the sequence.

Our algorithms parameterized by the maximum degree can be straightforwardly generalized for a recently introduced parameter *maximum pairwise connectivity* λ, which is the maximum number of vertex-disjoint paths with at least one internal vertex between any pair of vertices [19]. Clearly, λ is bounded by the maximum degree, since each of the disjoint paths has to use one of the edges adjacent to the starting vertex of the path. By Menger's theorem, any two non-adjacent vertices u and v can be separated by removing at most λ vertices. Thus, including the vertices to the queried sets with probability $1/(\lambda + 1)$ yields a tighter upper bound $\mathcal{O}(\lambda^2 \log n)$ for the needed number of queries such that the algorithm succeeds with high probability. Similarly, we can consider λ-Query-Schemes for the derandomized version.

3.3 Bounded Treewidth

Next, we will provide algorithms for GR in bounded treewidth graphs. In what follows, we assume that the hidden graph G has treewidth k. We will use the following well-known fact.

▶ **Fact 12** ([27]). Let $G = (V, E)$ be a k-tree. Suppose $u, v \in V$ such that $uv \notin E$. Then there exists a set $S \subseteq V \setminus \{u, v\}$ with $|S| = k$ such that u and v are in different connected components in $G[V \setminus S]$.

We give an adaptive algorithm that uses 2 rounds of adaptivity when the treewidth k of the hidden graph is known. First, the algorithm uses Fact 12 to construct a supergraph G^+ of the hidden graph G such that the treewidth of G^+ is at most k. Then, the algorithm removes edges from G^+ to find G.

Algorithm 2 performs the first step: It outputs a supergraph of the hidden graph. Note that, to get a correct result, one does not actually need to know the treewidth of the hidden graph G but an upper bound of it. However, the number of queries conducted increases if the bound is not tight. Next, we will prove the correctness and complexity of Algorithm 2.

▶ **Lemma 13.** *Algorithm 2 is a non-adaptive randomized algorithm that executes $\mathcal{O}(k^2 \log n)$ CC queries and returns a graph G^+ which is a supergraph of the hidden graph G and, with high probability, has treewidth at most k.*

Proof. The proof is a straightforward adaptation from Konrad et al. [18]. Let $\bar{G} = (V, \bar{E})$ be any k-tree which is a supergraph of G. Let $\bar{E}^- = V \times V \setminus \bar{E}$ be the set of non-edges in $\bar{G}$. Next, we will prove that Algorithm 2 detects all non-edges $\bar{E}^-$ of $\bar{G}$ with high probability.

By Fact 12, if u and v are not neighbors in $\bar{G}$ then there exists a separator U of size k. Furthermore, as $\bar{G}$ is a supergraph of G, U separates u and v in G. To detect the non-edge uv it is sufficient to conduct query $\mathrm{CC}(Q)$ for some Q subject to $u, v \in Q$ and $w \notin Q$ for all

Algorithm 2 An algorithm for GR for learning a supergraph of treewidth k.

Input: A vertex set V, a CC oracle $\mathrm{CC}(\cdot)$, upper bound for treewidth k
Output: A supergraph G^+ of the hidden graph G with treewidth at most k with high probability

```
Initialize G⁺ to complete graph on V;
for t = O(k² log n) times do
    Sample Q ⊆ V s.t. v ∈ Q with probability 1/(k+1) for all v ∈ V independently;
    C ← CC(Q);
    Remove from G⁺ edges uv for which v and u are in different components in C;
return G⁺;
```

$w \in U$. Again, by Lemma 1, the probability of this occurring is at least $1/((k+1)^2 \cdot \mathrm{e})$. We conduct $t \coloneqq c \cdot (k+1)^2 \cdot \lceil \ln n \rceil$ queries. The probability that we fail to identify a non-edge is

$$\begin{aligned}
\Pr(\text{non-edge } uv \text{ not detected}) &= \prod_{i=1}^{t} \left(1 - \Pr(u, v \in Q \text{ and } w \notin Q \text{ for all } w \in U)\right) \\
&\leq \left(1 - \frac{1}{(k+1)^2 \cdot \mathrm{e}}\right)^{c \cdot (k+1)^2 \cdot \lceil \ln n \rceil} \\
&\leq \left(\mathrm{e}^{-\frac{1}{(k+1)^2 \cdot \mathrm{e}}}\right)^{c \cdot (k+1)^2 \cdot \ln n} \\
&= \mathrm{e}^{-c \ln(n)/\mathrm{e}}
\end{aligned}$$

where the second inequality follows from the fact that $1 + x \leq \mathrm{e}^x$. If $c \geq 3\mathrm{e}$ then $\Pr(\text{non-edge } uv \text{ not detected}) \leq 1/n^3$. There are at most $n(n-1)/2$ potential non-edges, and the probability of not detecting at least one of them is at most $\mathcal{O}(1/n)$ by union bound. ◀

The graph found by Algorithm 2 may contain edges that are not present in the hidden graph. Next, we show how to prune a supergraph to find the hidden graph.

We take advantage of the following fact about existence of small balanced separators in bounded treewidth graphs (see, e.g., Lemma 7.19 of the textbook of Cygan et al. [12]):

▶ **Fact 14** (folklore)**.** Let G be a graph with treewidth k. Then there exists a 1/2-balanced separator of order at most $k+1$.

A *coloring* of a graph G is a mapping from the vertex set to some number of colors such that neighboring vertices have always different color. Graphs with small treewidth can be colored with few colors (see, e.g., Sopena [29]):

▶ **Fact 15** (folklore)**.** Every graph with treewidth at most k can be colored with $k+1$ colors.

The core idea of the algorithm is to find small separators whose removal splits the graph into smaller connected components and then recurse to split them into even smaller components. At each stage, we find all the neighbors of the vertices belonging to the separator. Since we know the neighbors of the separators, we exclude them from the future queries, allowing us to conduct tests in parallel for the connected components resulting from the removal of the separators.

▶ **Lemma 16.** *Suppose that we are given a graph G^+ whose treewidth is k and which is a supergraph of the hidden graph G. Then Algorithm 3 is a deterministic, non-adaptive algorithm that finds the hidden graph G with $\mathcal{O}(k^2 \log n)$ CC queries.*

Algorithm 3 An algorithm for GR given a supergraph of the hidden graph.

Input: A vertex set V, a CC oracle $\mathrm{CC}(\cdot)$, a supergraph of the hidden graph G^+
Output: The hidden graph G

$G^* \leftarrow G^+$;
Find a tree decomposition (X, T) of G^+;
Color vertices in G^+ arbitrarily using $k+1$ colors;
$R \leftarrow V$ // Vertices that have not been processed yet;
$\mathcal{Q} \leftarrow \emptyset$ // Queries that will be conducted;
while $R \neq \emptyset$ **do**
  $\mathcal{S} = \{\}$ // Collection of sets;
  for C that is a connected component of $G^+[R]$ **do**
    Find a balanced separator X' of C;
    Append X' to $\mathcal{S}$;
  $h = \max_{X' \in \mathcal{S}} |X'|$;
  Create a collection K_1 of $h(h-1)/2$ sets such that every set in K_1 contains at most two elements from X' for all $X' \in \mathcal{S}$ and every pair $u, v \in X'$, $u \neq v$ appears together in at least one set in K_1;
  $S' \leftarrow \bigcup_{U \in \mathcal{S}} U$;
  Create a collection K_2 of $h(k+1)$ sets such that each set in K_2 contains one element from each $X' \in \mathcal{S}$ and all vertices of one color in $R \setminus S'$ such that every vertex in S' appears at least once in a set with each color;
  Add all sets in K_1 and K_2 to $\mathcal{Q}$;
  $R \leftarrow R \setminus S'$;
for $Q \in \mathcal{Q}$ **do**
  $\mathcal{C} \leftarrow \mathrm{CC}(Q)$;
  Remove from G^* edges uv for which u and v are in different components in $\mathcal{C}$;
return G^*

Proof. We show that, by conducting all queries in $\mathcal{Q}$ after an iteration of the `while` loop, we find all the neighbors of the vertices in $V \setminus R$. The algorithm terminates when $R = \emptyset$. Thus, the algorithm constructs the hidden graph G.

At each iteration of the `while` loop, we find all the neighbors to vertices in S'. To find out whether u is a neighbor of v, it suffices to conduct a query $\mathrm{CC}(Q)$ such that $u, v \in Q$ and none of the other neighbors of v are in Q.

Let us consider an arbitrary vertex $v \in S'$ which belongs to a balanced separator X' in a connected component C in $G^+[R]$. First, suppose $u \in X'$. We want to conduct a query with set Q such that $u, v \in Q$ and none of the other neighbors of v are in Q. The set K_1 contains such a set Q as the collection has at least one set with that contains both u and v and all the other vertices in that set are in different connected components in $G^+[R]$.

Next, suppose that $u \notin X'$. Now the collection K_2 contains a desired set Q. The vertices in outside S' included in the set Q are of the same color. Thus, they do not have neighbors that is not in S' is included in the set Q. Thus, if there is no edge between u and v in G, then u and v have to be in different connected components in $\mathrm{CC}(Q)$.

At each iteration of the `while` loop, each of the connected components has size at most half of the size of the component before removing the balanced separator. Therefore, the procedure terminates after $\mathcal{O}(\log n)$ iterations. We conduct at most $\mathcal{O}(k^2)$ queries on each round. Thus, in total, we conduct at most $\mathcal{O}(k^2 \log n)$ queries. ◀

To solve GR for graphs with bounded treewidth, we simply run Algorithms 2 and 3 resulting in Algorithm 4.

Algorithm 4 An algorithm for GR for graphs with bounded treewidth.

Input: A vertex set V, a CC oracle $\mathrm{CC}(\cdot)$, treewidth k
Output: The hidden graph G with high probability
1 $G^+ \leftarrow$ Algorithm 2$(V, \mathrm{CC}(\cdot), k)$;
2 $G \leftarrow$ Algorithm 3$(V, \mathrm{CC}(\cdot), G^+)$;
3 **return** G

▶ **Theorem 17.** *Algorithm 4 is an adaptive randomized algorithm that solves* GR *with* $\mathcal{O}(k^2 \log n)$ *CC queries when* k *is known. The output of the algorithm is correct with high probability.*

Proof. By Lemma 13, Algorithm 2 returns with high probability a supergraph of G with treewidth at most k using $\mathcal{O}(k^2 \log n)$ queries. By Lemma 16, Algorithm 3 always returns the hidden graph G using $\mathcal{O}(k^2 \log n)$ queries, assuming that G^* has treewidth at most k. Combining these two results, we get that Algorithm 4 solves GR with high probability and uses $\mathcal{O}(k^2 \log n)$ queries. ◀

We can derandomize Algorithm 4 similarly to how Algorithm 1 is derandomized. That is, we replace the randomized subsets of Algorithm 2 by a k-Query-Scheme, resulting in a deterministic construction of a supergraph with treewidth at most k.

▶ **Theorem 18.** *There exists an adaptive deterministic algorithm that solves* GR *with* $\mathcal{O}(k^3 \log n)$ *CC queries when* k *is known.*

Proof. By Lemma 7, the k-Query-Scheme consists of $\mathcal{O}(k^3 \log n/k)$ sets that are used as CC queries. Furthermore, by Lemma 16, Algorithm 3 conducts $\mathcal{O}(k^2 \log n)$ CC queries. By combining these results, we conclude that the algorithm conducts $\mathcal{O}(k^3 \log n)$ CC queries. ◀

Algorithm 4 assumes that the treewidth k of the hidden graph is known. Next, we show how to adapt it to work when k is not known.

▶ **Theorem 19.** *There exists an adaptive randomized algorithm and an adaptive deterministic algorithm that solve* GR *with* $\mathcal{O}(k^2 \log n)$ *and* $\mathcal{O}(k^3 \log n)$ *CC queries, respectively. The algorithms use* $\mathcal{O}(\log k)$ *rounds of adaptivity. The output of the randomized algorithm is correct with high probability.*

Proof. The proof is analogous to the doubling approach of Theorem 11. ◀

Combining the adaptive algorithms for m, Δ, and k yields the following result:

▶ **Theorem 20.** *There exists an adaptive randomized algorithm that solves* GR *with* $\mathcal{O}(\min\{m/\log m, \Delta^2, k^2\} \cdot \log n)$ *CC queries. The output of the algorithm is correct with high probability.*

Proof. We use the adaptive randomized algorithms from Corollary 2, Theorem 11, and Theorem 19. If ran separately, they would stop after performing some number of queries – say, t_1, t_2, and t_3, respectively. Consider instead running the algorithms in parallel, such that in the ith iteration we perform the ith query of each of the three algorithms. If any of them would stop after the query, we output the graph computed by that algorithm. In total, we thus perform $3 \cdot \min\{t_1, t_2, t_3\} = \mathcal{O}(\min\{m/\log m, \Delta^2, k^2\} \cdot \log n)$ CC queries. The output of each algorithm is correct with high probability, and so is the output of this algorithm. ◀

3.4 Bounded Degeneracy

Next, we prove a polylogarithmic upper bound for the query complexity for graphs of bounded degeneracy. The bound holds also for graphs with bounded number of edges, maximum degree, and treewidth, since such graphs also have bounded degeneracy. For planar graphs, the degeneracy d is at most five [20] but the number of edges, the maximum degree and the treewidth can be unbounded in terms of d, so the polylogarithmic upper bound holds for a strictly larger family of graphs than with the previous algorithms. However, this comes at the expense of an additional $\mathcal{O}(\log n)$ factor in the query complexity.

Note that if the degeneracy is d, then the average degree of every induced subgraph is at most $2d$. Consequently, if the average degree of a graph is at most $2d$, then at least half of its vertices have degree at most $4d$. We thus repeatedly remove all vertices of degree at most $4d$ from the copy of G, at least halving the number of vertices in each iteration.

▶ **Theorem 21.** *Algorithm 5 is an adaptive randomized algorithm that solves* GR *with* $\mathcal{O}(d^2 \log^2 n)$ *CC queries when* d *is known. The output of the algorithm is correct with high probability.*

Proof. Analogously to the proof of Theorem 3, one can show that after running Algorithm 1 with a possibly incorrect value $4d$ for the maximum degree, the outputted graph G' has an edge uv if and only if the hidden graph has that edge with high probability for all vertices v with degree at most $4d$ in G. In other words, with high probability, we correctly identify all neighbors of vertices with degree at most $4d$ in G. Denote this set of vertices by L. It then remains to identify the structure of the subgraph of the hidden graph induced by $V \setminus L$. This is precisely what Algorithm 5 does, repeating the process until we have identified the neighbors for all vertices. Since at least half of the vertices of each induced subgraph have degree at most $4d$, the process is repeated at most $1 + \log_2 n$ times. Taking into account the query complexity of Algorithm 1, we perform $\mathcal{O}(d^2 \log^2 n)$ CC queries in total. Finally, recall from Theorem 3 that Algorithm 1 works correctly with probability at least $1 - 1/n$. Since $(1 - 1/n)^{1+\log_2 n}$ tends to 1 as n increases, the output is correct with high probability. ◀

Algorithm 5 An algorithm for GR for graphs with bounded degeneracy.

Input: A vertex set V, a CC oracle $\mathrm{CC}(\cdot)$, degeneracy d
Output: The hidden graph G with high probability

```
Initialize G⁺ to a complete graph on V;
V' ← V;
while V' ≠ ∅ do
    Let G' be the output of Algorithm 1 on the vertex set V' with (incorrect)
      maximum degree 4d;
    Remove edges uv from G⁺ for which edge uv is not in G' and u, v ∈ V';
    Let L be the set of vertices in G' with degree at most 4d;
    V' ← V' \ L;
return G⁺;
```

If we instead apply the non-adaptive deterministic algorithm from Theorem 10 instead of the randomized algorithm for bounded degree graphs, we get an adaptive deterministic algorithm for graphs with bounded degeneracy.

▶ **Theorem 22.** *There exists an adaptive deterministic algorithm that solves* GR *with* $\mathcal{O}\big(d^3 \log(n) \log(n/d)\big)$ *CC queries when* d *is known.*

Proof. The proof is analogous to the proof of Theorem 21 except that we use a d-Query-Scheme similarly to Theorem 10 to eliminate randomness. ◀

The idea for adjusting the algorithm for graphs of unknown degeneracy works similarly to before, but some care is needed to identify that our guess for the degeneracy was too low.

▶ **Theorem 23.** *There exist an adaptive randomized algorithm and an adaptive deterministic algorithm that solve* GR *with* $\mathcal{O}\big(d^2 \log^2 n\big)$ *and* $\mathcal{O}\big(d^3 \log(n) \log(n/d)\big)$ *CC queries, respectively. The output of the randomized algorithm is correct with high probability.*

Proof. We start by guessing that $D = 1$ is an upper bound for d, and then run Algorithm 5 or the algorithm from Theorem 22 as if the known value of d were D. If we use the former algorithm and it runs correctly or we use the latter algorithm, then we identify that at least half of the vertices have degree at most $4(D + 1)$ in each iteration if $D \geq d$. This may also occur if $D < d$, but if does not occur, then we know that $D < d$ and we double the value of D and restart the algorithm. The last round with $D < 2d$ dominates the query complexity.

It remains to show that the randomized algorithm succeeds with high probability. Recall that the probability that Algorithm 5 succeeds is at least $(1 - 1/n)^{1+\log_2 n}$. If the algorithm succeeds on all $\mathcal{O}(\log d)$ rounds of adaptivity, then the output is correct. The probability of this is at least $(1 - 1/n)^{\mathcal{O}(\log^2 n)} \approx (1/\mathrm{e})^{\mathcal{O}(\log^2(n)/n)}$, which tends to 1 as n increases. ◀

4 Lower Bounds

In this section, we show that the query complexity of our randomized algorithms cannot be improved by a factor larger than $\mathcal{O}(\log n)$ with a proof similar to the lower bound proof of Konrad et al. [18]. Whilst Black et al. [9] have showed a tighter upper bound with respect to m for the #CC oracle, their information-theoretical argument does not work here because our oracle yields more information than theirs. However, we confirm that their non-adaptive lower bound is applicable for the CC oracle, showing that non-adaptive algorithms require $\Omega(n^2)$ CC queries if no additional information about the structure is given.

▶ **Theorem 24.** *No algorithm can solve* GR *with* $o(m)$, $o(k^2)$, $o(\Delta^2)$, *or* $o(d^2)$ *CC queries with probability greater than* $1/2$.

Proof. Fix two disjoint subsets of vertices $C_1, C_2 \subseteq V$ of some size η and consider the family of graphs $\mathcal{G}_\eta$ where both C_1 and C_2 induce a clique and there is an arbitrary subset of edges between C_1 and C_2. Clearly, $|\mathcal{G}_\eta| = 2^{\eta^2}$.

Suppose now that there is a randomized algorithm that outputs the correct graph for each hidden graph $G \in \mathcal{G}_\eta$ with probability greater than $1/2$ while performing at most $\eta^2 - 1$ queries. By Yao's lemma, there is then a deterministic algorithm that performs $\eta^2 - 1$ queries and outputs the correct graph for more than half of the hidden graphs $G \in \mathcal{G}_\eta$. However, observe that for any CC query on a set $S \subseteq V$, there are exactly two possible outputs over the graphs $G \in \mathcal{G}_\eta$: the vertices in $S \setminus (C_1 \cup C_2)$ are always in connected components of size 1 and the remaining connected components induced by S are either $S \cap C_1$ and $S \cap C_2$ or just S. Consequently, the deterministic algorithm can identify in $\eta^2 - 1$ queries only $2^{\eta^2-1} \leq |\mathcal{G}_\eta|/2$ hidden graphs, resulting in a contradiction. Thus, $\Omega(\eta^2)$ queries are needed. We have $k, \Delta, d \leq 2\eta$ and $m \leq 2\eta^2$ for this family of hidden graphs, completing the proof. ◀

▶ **Corollary 25** ([9])**.** *No non-adaptive algorithm can solve* GR *with* $o(n^2)$ *CC queries with probability greater than* $1/2$ *even if* $k = 2$ *and* $m = \mathcal{O}(n)$.

Proof. Let G_{uv} denote a graph on n vertices such that we start by having the vertices u and v, and then add $n-2$ vertices that neighbor u and v. Denote by G_{uv}^+ the graph obtained from G_{uv} by adding the edge uv. Both G_{uv} and G_{uv}^+ have treewidth 2 and $m = \mathcal{O}(n)$.

Suppose that the hidden graph is either G_{uv} or G_{uv}^+ but the non-adaptive algorithm does not perform the query $\{u, v\}$. Then, we cannot distinguish between them since u and v would not be in distinct connected components for any of the performed CC queries. After performing all the queries, the best one can do is to guess between G_{uv} and G_{uv}^+, resulting in a success probability $1/2$. For the algorithm to work correctly with probability greater than $1/2$ for any graph G_{uv} or G_{uv}^+, the non-adaptively constructed set of queries $\mathcal{Q}$ has to include the query $\{u, v\}$ for all distinct vertices $u, v \in V$ with probability greater than $1/2$. By the linearity of expectation, the expected size of $\mathcal{Q}$ would then have to be at least $\binom{n}{2}/2$. In other words, for any algorithm that always performs $o(n^2)$ queries, there is an instance on which it fails with probability at least $1/2$. ◀

5 Relationship with Other Oracles

In this section, we compare our CC oracle to the connected component counting oracle and the maximal independent set oracle.

5.1 Relationship with the Connected Component Counting Oracle

We start by proving that it is not sufficient to use the connected component counting (#CC) *oracle* of Black et al. [9] for obtaining our results, since $\Omega(n/\log n)$ #CC queries are needed for already very constrained graph families. The main observation for this result is that the #CC oracle has $n+1$ possible distinct values it can output, and so in t queries we can correctly identify only $(n+1)^t$ graphs. Constructing a family of graphs larger than that shows that more than t queries are needed. Similar combinatorial arguments were used by Michel and Scott [22] and Black et al. [9] to prove lower bounds for the maximal independent set oracle in bounded degree graphs and for the #CC oracle in graphs with m edges.

▶ **Theorem 26.** *No algorithm can solve all instances of* GR *with maximum degree, treewidth, or degeneracy at most p with $\Omega(np/\log n)$ #CC queries with probability greater than $1/2$.*

Proof. Without loss of generality, assume that p divides n. Let G be a graph on n vertices $v_1, v_2, \ldots, v_n$ where there is an edge $v_i v_j$ if and only if $\lfloor (i-1)/p \rfloor = \lfloor (j-1)/p \rfloor$ or $i = j-1$. The maximum degree, treewidth, and degeneracy of G are all at most p.

We can now construct a family of graphs $\mathcal{G}$ by considering all subgraphs of G that retain all edges of the form $v_i v_{i+1}$. The size of the family is then $|\mathcal{G}| = 2^{\binom{p-1}{2} \cdot \frac{n}{p}} = (n+1)^{\binom{p-1}{2} \cdot \frac{n}{p} \cdot \log_{n+1}(2)}$.

Suppose that there is a randomized algorithm that outputs the correct graph for each hidden graph $G' \in \mathcal{G}$ with probability greater than $1/2$ and performs at most $t := \binom{p-1}{2} \cdot \frac{n}{p} \cdot \log_{n+1}(2) - 1$ #CC queries. Then, by Yao's lemma, there is a deterministic algorithm that performs t and outputs the correct graph for more than half of the hidden graphs $G' \in \mathcal{G}$. However, a deterministic algorithm can only have $(n+1)^t \leq |\mathcal{G}|/(n+1)$ distinct outputs if it performs t queries, and so the output is incorrect for more than half of the hidden graphs. ◀

5.2 Relationship with the Maximal Independent Set Oracle

We next compare the CC oracle against the *maximal independent set* (MIS) *oracle* of Konrad et al. [18], which returns a maximal independent set of $G[S]$. Note that the MIS oracle can be adversarial, that is, it may return the "least useful" maximal independent set. Here, we demonstrate that CC queries are more powerful than MIS queries on some instances.

▶ **Theorem 27.** *There are instances of* GR *that take* $\Omega(n)$ *MIS queries to solve but* $\mathcal{O}(\log n)$ *CC queries.*

Proof. Take a star graph on n vertices centered around the vertex v and consider query sets $Q_1, Q_2, \ldots, Q_\ell$. Suppose our adversarial MIS oracle returns the set $Q_i \setminus \{v\}$ for the query set Q_i for all $i \in \{1, 2, \ldots, \ell\}$. Then, the results are also consistent with a graph with edges only between v and vertices $\max_{u \in Q_i \setminus \{v\}} u$ for all i, under some arbitrary ordering of the vertices. Thus, we need ℓ to be at least $n - 1$ to recover the original graph. Meanwhile, the graph is a tree, and thus the instance is solvable by $\mathcal{O}(\log n)$ CC queries by Theorem 18. ◀

6 Conclusions

We proposed a novel connected components oracle and showed that it requires only a (poly)logarithmic number of queries to solve graph reconstruction in graphs of bounded number of edges, maximum degree, treewidth, or degeneracy. These results were complemented by unconditional lower bounds and comparisons against the #CC oracle and the MIS oracle.

It remains open whether the gap between our lower and upper bounds can be tightened. Michel and Scott [22] recently improved the lower bound for the MIS oracle from $\Omega(\Delta^2)$ to $\omega(\Delta^2)$, but their techniques do not appear to directly generalize for the CC oracle.

Another direction is to study how much weaker the CC oracle could be made such that we would still obtain similar upper bounds. For example, an oracle that might be weaker than the CC oracle but stronger than the #CC oracle would be one that could answer the following query: Given a set Q, output one vertex from each connected component of $G[Q]$.

In the context of Markov networks where the separation oracle is used, the query complexity of graph reconstruction has been recently studied under constraints on the size of the queried sets [19]. For example, $n^{\Omega(\lambda)}$ separation queries of size λ are needed when the maximum pairwise connectivity is λ. Here, we were able to reconstruct the graph in $\mathcal{O}(\lambda^2 \log n)$ queries, but the queried sets were larger. A natural follow-up question would thus be how the connected components oracle performs with query sets of bounded size.

References

1 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. URL: `http://proceedings.mlr.press/v98/abasi19a.html`.

2 Mikkel Abrahamsen, Greg Bodwin, Eva Rotenberg, and Morten Stöckel. Graph reconstruction with a betweenness oracle. In *Proceedings of the 33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016*, volume 47 of *LIPIcs*, pages 5:1–5:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. `doi:10.4230/LIPIcs.STACS.2016.5`.

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

4 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`.

5 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995. `doi:10.1145/210332.210337`.

6 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`.

7 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`.

8 Richard Beigel, Noga Alon, Simon Kasif, Mehmet Serkan Apaydin, and Lance Fortnow. An optimal procedure for gap closing in whole genome shotgun sequencing. In *Proceedings of the Fifth Annual International Conference on Computational Biology, RECOMB 2001*, pages 22–30. ACM, 2001. doi:10.1145/369133.369152.

9 Hadley Black, Arya Mazumdar, Barna Saha, and Yinzhan Xu. Optimal graph reconstruction by counting connected components in induced subgraphs. In *The Thirty Eighth Annual Conference on Learning Theory, COLT 2025*, volume 291 of *Proceedings of Machine Learning Research*, pages 315–343. PMLR, 2025. URL: https://proceedings.mlr.press/v291/black25a.html.

10 Mathilde Bouvel, Vladimir Grebinski, and Gregory Kucherov. Combinatorial search on graphs motivated by bioinformatics applications: A brief survey. In *Proceedings of the 31st International Workshop Graph-Theoretic Concepts in Computer Science, WG 2005*, volume 3787 of *Lecture Notes in Computer Science*, pages 16–27. Springer, 2005. doi:10.1007/11604686_2.

11 C. K. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Trans. Inf. Theory*, 14(3):462–467, 1968. doi:10.1109/TIT.1968.1054142.

12 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

13 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.

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

15 Sampath Kannan, Claire Mathieu, and Hang Zhou. Graph reconstruction and verification. *ACM Trans. Algorithms*, 14(4):40:1–40:30, 2018. doi:10.1145/3199606.

16 David R. Karger and Nathan Srebro. Learning Markov networks: maximum bounded tree-width graphs. In *Proceedings of the Twelfth Annual Symposium on Discrete Algorithms, SODA 2001*, pages 392–401. ACM/SIAM, 2001. URL: http://dl.acm.org/citation.cfm?id=365411.365486.

17 Daphne Koller and Nir Friedman. *Probabilistic Graphical Models - Principles and Techniques*. MIT Press, 2009.

18 Christian Konrad, Conor O'Sullivan, and Victor Traistaru. Graph reconstruction via MIS queries. In *Proceedings of the 16th Conference on Innovations in Theoretical Computer Science, ITCS 2025*, volume 325 of *LIPIcs*, pages 66:1–66:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.ITCS.2025.66.

19 Tuukka Korhonen, Fedor V. Fomin, and Pekka Parviainen. Structural perspective on constraint-based learning of Markov networks. In *International Conference on Artificial Intelligence and Statistics, AISTATS 2024*, volume 238 of *Proceedings of Machine Learning Research*, pages 1855–1863. PMLR, 2024. URL: https://proceedings.mlr.press/v238/korhonen24a.html.

20 Don R. Lick and Arthur T. White. k-degenerate graphs. *Canadian Journal of Mathematics*, 22(5):1082–1096, 1970.

21 Claire Mathieu and Hang Zhou. A simple algorithm for graph reconstruction. *Random Struct. Algorithms*, 63(2):512–532, 2023. doi:10.1002/RSA.21143.

22 Lukas Michel and Alex D. Scott. Lower bounds for graph reconstruction with maximal independent set queries. *Theor. Comput. Sci.*, 1034:115121, 2025. doi:10.1016/J.TCS.2025.115121.

23 Ely Porat and Amir Rothschild. Explicit nonadaptive combinatorial group testing schemes. *IEEE Transactions on Information Theory*, 57(12):7982–7989, 2011. doi:10.1109/TIT.2011.2163296.

24 Lev Reyzin and Nikhil Srivastava. Learning and verifying graphs using queries with a focus on edge counting. In *Proceedings of the Eighteenth International Conference on Algorithmic Learning Theory, ALT 2007*, volume 4754 of *Lecture Notes in Computer Science*, pages 285–297. Springer, 2007. doi:10.1007/978-3-540-75225-7_24.

25 Guozhen Rong, Wenjun Li, Yongjie Yang, and Jianxin Wang. Reconstruction and verification of chordal graphs with a distance oracle. *Theor. Comput. Sci.*, 859:48–56, 2021. doi:10.1016/J.TCS.2021.01.006.

26 Guozhen Rong, Yongjie Yang, Wenjun Li, and Jianxin Wang. A divide-and-conquer approach for reconstruction of $\{C_{\geq 5}\}$-free graphs via betweenness queries. *Theor. Comput. Sci.*, 917:1–11, 2022. doi:10.1016/J.TCS.2022.03.008.

27 Donald J. Rose. On simple characterizations of k-trees. *Discret. Math.*, 7(3-4):317–322, 1974. doi:10.1016/0012-365X(74)90042-9.

28 Sandeep Sen and V. N. Muralidhara. The covert set-cover problem with application to network discovery. In *Proceedings of the 4th International Workshop on Algorithms and Computation, WALCOM 2010*, volume 5942 of *Lecture Notes in Computer Science*, pages 228–239. Springer, 2010. doi:10.1007/978-3-642-11440-3_21.

29 Éric Sopena. The chromatic number of oriented graphs. *J. Graph Theory*, 25(3):191–205, 1997. doi:10.1002/(SICI)1097-0118(199707)25:3\%3C191::AID-JGT3\%3E3.0.CO;2-G.