(Quantum) complexity of testing signed graph clusterability

Kuo-Chin Chen, Simon Apers, Min-Hsiu Hsieh

(Quantum) complexity of testing signed graph clusterability

KUO-CHIN CHEN, Hon Hai Quantum Computing Research Center, Taiwan
SIMON APERS, Université de Paris, CNRS, IRIF, France
MIN-HSIU HSIEH, Hon Hai Quantum Computing Research Center, Taiwan

This study examines clusterability testing for a signed graph in the bounded-degree model. Our contributions are two-fold. First, we provide a quantum algorithm with query complexity $\tilde{O}(N^{1/3})$ for testing clusterability, which yields a polynomial speedup over the best classical clusterability tester known [1]. Second, we prove an $\tilde{\Omega}(\sqrt{N})$ classical query lower bound for testing clusterability, which nearly matches the upper bound from [1]. This settles the classical query complexity of clusterability testing, and it shows that our quantum algorithm has an advantage over any classical algorithm.

1 INTRODUCTION

Property testing [2, 3] deals with the setting where we wish to distinguish between objects, e.g. functions [4–6] or graphs [7–12], that satisfy a predetermined property and those that are far from satisfying this property. For certain properties, this relaxed setting allows for algorithms to query only a small part of (sometimes huge) data sets. Indeed, the goal in property testing is to design so-called *property testers* to solve a property testing problem within sublinear time complexity. Property testing has been studied in many settings, such as computational learning theory [13–18], quantum information theory [19–24], coding theory [25–31], and so on. This witnesses the significant attention that property testing has drawn from the academic community.

An interesting setting is that of graph property testing. In the *dense graph model*, it was shown that a constant number of queries are needed to test a wide range graph partition properties [13], including k-colorability, ρ-clique, and ρ-cut for any fixed $k \geq 2$ and $\rho > 0$. For comparison, in the *bounded-degree model* [32] similar graph properties such as bipartiteness and expansion testing require sublinear $\tilde{\Theta}(\sqrt{N})$ classical queries. Moreover, some graph properties even have a (trivial) $\Omega(N)$ query complexity, as Ref. [33] showed for 3-colorability in the bounded-degree model. While there have been numerous studies on testing graph properties, there has been little work on testing the properties of *signed* graphs.

A signed graph is a graph where each edge is assigned a positive or a negative label. They can be applied to model a variety of problems including correlation clustering problems [14, 34], modeling the ground state energy of Ising models [35], and social network problems [36–38]. Signed graphs have different properties than unsigned graphs. One of these is the important property of clusterability, which was introduced by Davis [39] to describe the correlation clustering problem. We call a signed graph clusterable if it can be decomposed into several components such that (1) the edges in each component are all positive, and (2) the edges connecting the vertices belonging to different components are all negative. This property is equivalent to not having a "bad cycle", which is a cycle with exactly one negative edge [39]. An algorithm for testing clusterability in the bounded-degree model with only $\tilde{O}(\sqrt{N})$ was proposed in [1]. The optimality of this clusterability tester was left as an open question. Here, we prove that any classical algorithm requires at least $\tilde{\Omega}(\sqrt{N})$ queries to test clusterability, showing that the tester from [1] is nearly-optimal.

As a natural extension of past studies, we are interested in whether quantum computing can provide any advantages in testing clusterability for signed graphs. To the best of our knowledge, we are not aware of any previous work on the quantum advantage for testing the properties of

Authors' addresses: Kuo-Chin Chen, Hon Hai Quantum Computing Research Center, Taipei, Taiwan, Jim.KC.Chen@foxconn.com; Simon Apers, Université de Paris, CNRS, IRIF, Paris, France, smgapers@gmail.com; Min-Hsiu Hsieh, Hon Hai Quantum Computing Research Center, Taipei, Taiwan, min-hsiu.hsieh@foxconn.com.

signed graphs. However, in work by Ambainis, Childs and Liu [40], a quantum speedup for testing bipartiteness and expansion of bounded-degree graphs was shown. We adopt these techniques to obtain a quantum algorithm for testing clusterability in signed graphs. More specifically, we combine their quantum approach with the classical property testing techniques provided by Adriaens and Apers [1] to obtain a quantum algorithm for testing the clusterability of bounded-degree graphs in time $\tilde{O}(N^{1/3})$. This outperforms the $\tilde{O}(\sqrt{N})$ query complexity of the classical tester in [1] (which is optimal by our lower bound). We leave optimality of the quantum algorithm for testing clusterability as an open question. Indeed, settling the quantum query complexity of property testing in the bounded-degree graph model has been a long open question, and even for the well-studied problem of bipartiteness testing no matching lower bound is known [40].

1.1 Overview of Main Results

Here we formally state our main results (precise definitions are deferred to Section 2). First, we prove a lower bound on the classical query complexity of clusterability testing for a signed graph.

Theorem 3.1 (Restated). *Any classical clusterability tester with error parameter $\epsilon = 0.01$ must make at least $\sqrt{N}/10$ queries.*

Up to polylogarithmic factors this matches the upper bound from [1], thus proving that their clusterability tester is optimal in the classical computing regime. However, taking inspiration from this classical clusterability tester, we reduce the clusterability testing problem to a collision finding problem which can be solved faster by quantum computing. As a result, we propose a quantum clusterability tester with a query complexity $\tilde{O}(N^{1/3})$.

Theorem 3.5 (Restated). *We propose a quantum clusterability tester with query complexity $\tilde{O}(N^{1/3})$.*

This improves over the classical lower bound, implying a quantum advantage over classical algorithms for testing clusterability.

1.2 Technical contributions

A sketch of the proof of our two results is given in this section. The first result is the classical query lower bound for testing clusterability. While the bound follows the blueprint of the lower bound for bipartiteness testing by Goldreich and Ron [32], we have to deal with a number of additional complications in the signed graph setting.

The main idea of the lower bound is to show that, with less than $\sqrt{N}/10$ queries, we cannot distinguish two families of graphs: one family $\mathcal{G}_1^N$ that is ϵ-far from clusterable, and another family $\mathcal{G}_2^N$ that is clusterable. The design of these two families of graphs must take into account two constraints. The first constraint is that the graphs in $\mathcal{G}_2^N$ cannot contain a bad cycle, while those in $\mathcal{G}_1^N$ must have at least one bad cycle, even if we remove ϵNd edges of the graph. This ensures that $\mathcal{G}_2^N$ is clusterable, while $\mathcal{G}_1^N$ is far from clusterable. The second constraint relates to the fact that both graph should be locally indistinguishable. This requires for instance that vertices in each graph in both families are incident to the same number of positive and negative edges. If in addition we can ensure that each cycle in these graphs contains many edges with a constant probability, then we can use this to show that no algorithm can distinguish the graphs in these two families with $o(\sqrt{N})$ queries. Indeed, we show that these two families of graphs are indistinguishable with less than $\sqrt{N}/10$ queries as follows. First, we propose two random processes P_1 and P_2, one generates a uniformly random graph in $\mathcal{G}_1^N$, and the other generates a uniformly random graph in $\mathcal{G}_2^N$. Specifically, P_α for $\alpha \in \{1, 2\}$ takes a query given from an algorithm as input and returns a vertex while "on-the-fly" or "lazily" constructing a graph from $\mathcal{G}_\alpha^N$. In other words, P_α simulates

how an algorithm interacts with a graph sampled uniformly in $\mathcal{G}_\alpha^N$. We observe that these random processes are statistically identical if the answer to each query is not found in the past answers or queries, which is equivalent to not finding a cycle when exploring a graph. Second, we demonstrate that the probability of these random processes being statistically identical is greater than $1/4$ within $\sqrt{N}/10$ queries. In other words, no classical algorithm can distinguish between these two families with a probability exceeding $3/4$ within $\sqrt{N}/10$ queries to the input graph.

Our second result is a quantum algorithm for clusterability testing with a better query complexity. To this end, we reduce the main procedure in the algorithm proposed by Adriaens and Apers [1] to a collision finding algorithm. This collision finding problem can then be solved using the quantum collision finding algorithm, similar to [40]. The main idea is that if we implement several random walks on the positive edges of a graph that is far from clusterable, then there exists a negative edge between the vertices belonging to distinct random walks with a constant probability. We define finding such a negative edge between random walks as finding a collision, a process that can be solved by using a quantum collision finding algorithm. This yields a quantum speedup for clusterability testing.

2 PRELIMINARIES

This section contains two parts. Section 2.1 defines some of the basic terminology associated with the graphs used in this paper. In Section 2.2, we introduce the graph clusterability testing problem.

2.1 Terminology

A graph $G = (V, E)$ is a pair of sets. The elements in $V = [N]$ are vertices, and the elements in E, denoted by edges, are paired vertices. The vertices $v \in V$ and $u \in V$ of an edge $(v, u) \in E$ are the endpoints of (v, u), and (v, u) is incident to u and v. The vertices u and v are adjacent if there exist an edge $(v, u) \in E$. The number of edges incident with v, denoted by $d(v)$, is the degree of a vertex, and the maximum degree among the vertices in G is the degree of the graph $G(V, E)$.

Given a graph $G = (V, E)$, a walk is a sequence of edges $((v_1, v_2), (v_2, v_3), \cdots, (v_{J-1}, v_J))$ where $(v_j, v_{j+1}) \in E$ for all $1 \leq j \leq J - 1$ and $v_j \in V$ for all $1 \leq j \leq J$. This walk can also be denoted as a sequence of vertices $(v_1, v_2, \ldots, v_J)$. A trail is a walk in which all edges are distinct. A cycle is a non-empty trail in which only the first and last vertices are equal. A Hamiltonian cycle is a cycle of a graph in which every vertex is visited exactly once.

A signed graph $G = (V, E, \Sigma)$ consists of the vertex set V, the edge set $E \subseteq V \times V$, and a mapping $\Sigma : E \to \{+, -\}$ that indicates the sign of each edge. We say that a signed graph $G = (V, E, \Sigma)$ is clusterable if we can partition vertices into components such that (i) every edge that connects two vertices in the same components is positive, and (ii) every edge that connects two vertices in different components is negative.

2.2 Clusterability testing for signed graphs

We can easily modify the usual graph query model to signed graphs. Given a signed graph G with maximum degree d, the bounded-degree graph model is defined as follows. A query is a tuple (v, i) where $v \in [N]$ is a vertex in the graph and $i \in [d]$. The oracle answers this query with (i) the i^{th} neighbor of the vertex v if the degree of v is larger than i (otherwise it returns an error symbol), and (ii) the sign of the corresponding edge.

Property testing in the bounded-degree model is described as follows. Given oracle access to a graph G with degree bound d and $|V| = N$, we wish to distinguish whether the graph G satisfies a certain property, or whether it is ϵ-far from any graph having that property, where $\epsilon \in (0, 1]$ is an error parameter. Here we say that two graphs G and G' are ϵ-far from each other if we have to add

or remove at least ϵNd edges to turn G into G'. The specific case of clusterability testing is defined formally as follows.

Definition 2.1. A clusterability testing algorithm is a randomized algorithm that has query access to a signed graph $G(V, E, \Sigma)$ with $|V| = N$ and maximum degrees d. Given an error parameter ϵ, the algorithm behaves as follows:

- If G is clusterable, then the algorithm should accept with probability at least $2/3$.
- If G is ϵ-far from clusterable, then the algorithm rejects with probability at least $2/3$.

3 MAIN RESULTS AND PROOFS

In this section, we give the formal statements and proofs of our two main results – a classical query lower bound for clusterability testing and a quantum clusterability tester. In Section 3.1, we first give the classical query lower bound of $\Omega(\sqrt{N})$ for clusterability testing. This result claims the optimality of the classical clusterability tester in [1]. In Section 3.2, we provide a quantum clusterability tester with query complexity $\tilde{O}(N^{1/3})$ which outperforms the classical clusterability tester in [1].

3.1 Classical query lower bound for testing clusterability

In this section, we derive a classical query lower bound for the clusterability testing problem. Specifically, we show that testing the clusterability of a signed graph with N vertices requires at least $\sqrt{N}/10$ queries.

Theorem 3.1. *Given a signed graph G with N vertices, testing clusterability of G with error parameter $\epsilon = 0.01$ requires at least $\sqrt{N}/10$ queries.*

Proof. The proof consists of three main steps. First, we construct two families of graphs denoted as $\mathcal{G}_1^N$ and $\mathcal{G}_2^N$, each possessing specific desirable properties. In particular, we require that most graphs within $\mathcal{G}_1^N$ are at least 0.01-far from being clusterable, while graphs within $\mathcal{G}_2^N$ are inherently clusterable. The construction and analysis of these families is deferred to Section 4.1.

To prove Theorem 3.1, we illustrate the interaction between an arbitrary T-query clusterability testing algorithm $\mathcal{A}$ and a graph g uniformly sampled from $\mathcal{G}_\alpha^N$ as follows:

For all $t \leq T$, each query q_t is represented as a tuple (v_t, i_t), and the answer to q_t is denoted as a_t, where $v_t, a_t \in [N]$ and $i_t \in [6]$. It is crucial to note that each query q_t corresponds to an edge in g, specifically the edge (v_t, a_t). We additionally denote a list of tuples $h = [(q_1, a_1), (q_2, a_2), \ldots, (q_t, a_t)]$ as the query-answer history. This history is generated by the interaction between $\mathcal{A}$ and g in the following manner: For each $t \leq T$, $\mathcal{A}$ maps h to q_{t+1} and ultimately to either accept or reject for $t = T$. For a given history $h = [(q_1 = (v_1, i_1), a_1), \ldots, (q_t = (v_t, i_t), a_t)]$, we say that a vertex u is in h if $u = v_{t'}$ or $u = a_{t'}$ for some $t' \in [t]$.

Secondly, we introduce two processes, denoted as P_α for $\alpha \in \{1, 2\}$, which simulate how an algorithm $\mathcal{A}$ interacts with a graph sampled uniformly from $\mathcal{G}_\alpha^N$. To be more specific, we consider that $\mathcal{A}$ interacts with a graph g sampled from $\mathcal{G}_\alpha^N$ and generates the query-answer history h. We must have that the graph g is uniformly distributed in $\mathcal{G}_{\alpha,h}^N \subseteq \mathcal{G}_\alpha^N$, where $\mathcal{G}_{\alpha,h}^N$ includes all graphs that produce the query-answer history h during interactions with $\mathcal{A}$. Therefore, if $\mathcal{A}$ makes a query $q_{t+1} \notin \{q_i\}_{i=1}^t$ to a graph uniformly sampled from $\mathcal{G}_{\alpha,h}^N$, we can determine that the answer corresponds to a certain vertex $u \in [N]$ with a specific probability denoted as p_u. The random processes P_α are precisely defined to return the answer u with the corresponding probability p_u when responding to the query q_{t+1} (initiated by $\mathcal{A}$) and considering the history h. As a result, these two random processes, P_α, interact with $\mathcal{A}$, providing responses to $\mathcal{A}$'s queries while

simultaneously constructing a graph uniformly distributed in $\mathcal{G}_\alpha^N$. The description and analysis of these random processes are deferred to Section 4.2.

In the third part, we demonstrate that no algorithm can with high probability differentiate between query-answer histories generated during the interactions of $\mathcal{A}$ with P_1 and P_2 while making less than $\sqrt{N}/10$ queries. To prove such indistinguishability, we examine the distribution of query-answer histories of length T denoted as $\mathbf{D}_\alpha^{\mathcal{A}}$, where each element in $\mathbf{D}_\alpha^{\mathcal{A}}$ is generated through the interactions of $\mathcal{A}$ and P_α. The statistical difference between $\mathbf{D}_1^{\mathcal{A}}$ and $\mathbf{D}_2^{\mathcal{A}}$ is defined as follows:

$$\frac{1}{2} \cdot \sum_x \left| \text{Prob}\left[\mathbf{D}_1^{\mathcal{A}} = x\right] - \text{Prob}\left[\mathbf{D}_2^{\mathcal{A}} = x\right] \right|,$$

where x is some query-answer history of length T. We then provide an upper bound on this statistical difference in the following lemma. The proof of this lemma is a modification of the proof of Lemma 7.4 in [32], and we defer its proof to Section 4.3.

Lemma 3.2 (based on [32], Lemma 7.4)). *Let $\delta < \frac{1}{2}, T \leq \delta\sqrt{N}$ and $N \geq 40T$. For every algorithm $\mathcal{A}$ that uses T queries, the statistical distance between $\mathbf{D}_1^{\mathcal{A}}$ and $\mathbf{D}_2^{\mathcal{A}}$ is at most $10\delta^2$.*

Finally, we establish Theorem 3.1 through a proof by contradiction. Let us assume the existence of a clusterability tester $\mathcal{A}$ that requires only $\sqrt{N}/10$ queries. Consequently, we can infer that the probability of $\mathcal{A}$ accepting a graph from $\mathcal{G}_2^N$ is at least $2/3$. By referring to Lemma 3.2, we determine that the statistical difference between $\mathbf{D}_1^{\mathcal{A}}$ and $\mathbf{D}_2^{\mathcal{A}}$ is at most $10\delta^2 = 1/10$ where δ is set $1/10$ for a $\sqrt{N}/10$-query algorithm. Hence, $\mathcal{A}$ accepts a graph distributed uniformly in $\mathcal{G}_1^N$ with a probability of at least $2/3 - 1/10 > 0.4$.

Furthermore, as indicated by Proposition 4.1, more than 99% of the graphs in $\mathcal{G}_1^N$ are at least 0.01-far from being clusterable. Consequently, by the definition of a clusterability tester, we can conclude that $\mathcal{A}$ accepts a graph distributed uniformly in $\mathcal{G}_1^N$ at most $0.99 \cdot 1/3 + 0.01 < 0.35$. This contradicts the earlier finding that $\mathcal{A}$ accepts a graph distributed uniformly in $\mathcal{G}_1^N$ with a probability of at least 0.4. Hence, we can deduce that there does not exist a clusterability tester capable of distinguishing between a graph sampled from $\mathcal{G}_1^N$ and $\mathcal{G}_2^N$ using only $\sqrt{N}/10$ queries, and the theorem follows. □

3.2 Quantum clusterability tester

In this section, we present our second result: a quantum clusterability tester (Algorithm 1) with a query complexity of $O\left(N^{1/3} \text{ poly}\left(\log N/\epsilon\right)\right)$. We begin by introducing the quantum clusterability tester, followed by the proof of its correctness in Theorem 3.5.

Algorithm 1 takes a signed graph $G(V, E, \Sigma)$ with N vertices and a bound on the maximum degree d, along with an accuracy parameter $\epsilon \in (0, 1]$, as input. The goal is to determine whether $G(V, E, \Sigma)$ is clusterable or ϵ-far from clusterable. The algorithm consists of four major steps.

First, Algorithm 1 randomly selects a vertex $s \in V$. Second, it constructs random variables that determine the direction of movement in each step of these random walks. To achieve this, we need to prepare $O(K \cdot L)$ random variables (K and L are defined in Algorithm 1); however, we can derandomize and reduce the number of random bits from $O(K \cdot L)$ to $O(L)$ because Algorithm 1 only depends on each pair of walks that are selected from K random walks. Therefore, it is sufficient to construct k-wise independent random variables b_{ij} mapping to $[2d]$ for $i \in [K]$ and $j \in [L]$, where $k = \Theta(L)$. This construction can be realized by the following proposition.

Proposition 3.3. *([41], Proposition 6.5) Let $n + 1$ be a power of 2 and k be an odd integer such that $k \leq n$. In this scenario, there exists a uniform probability space denoted as $\Omega = \{0, 1\}^m$, where*

Algorithm 1 Quantum clusterability tester

Input: Oracle access to a signed graph $G(V, E, \Sigma)$ with N vertices and degree bound d; an accuracy parameter $\epsilon \in (0, 1]$.

1: **for** $O(1/\epsilon)$ times **do**
2: Pick a vertex $s \in V$ randomly.
3: Let $K = O\left(\sqrt{N}\,\mathrm{poly}(\log N/\epsilon)\right)$, $L = \mathrm{poly}\,(\log N/\epsilon)$, $n = KL$, and $k = \Theta(L)$.
4: Adopt Proposition 3.3 to construct k-wise independent random variables b_{ij} taking values
5: in $[2d]$ for $i \in [K]$ and $j \in [L]$.
6: Run the quantum collision finding algorithm in Lemma 3.4 with the following setting:
- $X := [K] \times [L]$; Y is the set of tuples (v, v_{neb}) where $v \in V$ and v_{neb} is the set of vertices adjacent to v.
- A function f that take $(i, j) \in X$ as input, and return the endpoint of a random walk that starts at s with random coin flips $\left(b_{i1}, \ldots, b_{ij}\right)$.
- Symmetric binary relation $R \subseteq Y \times Y$ defined as follows:

$\left((v, v_{\mathrm{neb}}), (v', v'_{\mathrm{neb}})\right) \in R$ iff ($v \in v'_{\mathrm{neb}}$ and the edge between v and v' is negative).

7: **if** quantum collision finding algorithm finds a collision **then**
8: **return** false
9: **end if**
10: **end for**
11: **return** true

$m = 1 + \frac{1}{2}(k-1)\log_2(n+1)$. *Within this space, there exist k-wise independent random variables, represented as $\xi_1, \ldots, \xi_n$ over Ω, such that* $\Pr\left[\xi_j = 1\right] = \Pr\left[\xi_j = 0\right] = \frac{1}{2}$.

Moreover, an algorithm exists that, when provided with $i \in \Omega$ and $1 \le j \le n$, can compute $\xi_j(i)$ in a computational time of $O(k \log n)$.

Third, we define a function f that implements random walks according to these random variables. f returns the endpoint of a random walk and the neighborhood of this endpoint. Specifically, we let $X = \{1, \ldots, K\} \times \{1, \ldots, L\}$, and Y be the set of tuples (v, v_{neb}) where $v \in \{1, \ldots, N\}$ and v_{neb} is the set of vertices adjacent to v. Then, we define the function f as follows. f takes $(i, j) \in X$ as input, then it runs a random walk according to the random variables $\left(b_{i1}, \ldots, b_{ij}\right)$ such that (i) this walk starts at s and (ii) each edge in this walk is positive. The function f finally returns a tuple (v, v_{neb}).

Fourth, we define the symmetric binary relation $R \subseteq Y \times Y$ such that $\left((v, v_{\mathrm{neb}}), (v', v'_{\mathrm{neb}})\right) \in R$ iff (i) $v \in v'_{\mathrm{neb}}$, and (ii) the edge between v and v' is negative. In other words, detecting a collision is equivalent to detecting a bad cycle. The last step is to detect two distinct elements $x_1, x_2 \in X$ such that $(f(x_1), f(x_2))$ satisfies the symmetric binary relation R. The collision finding problem can be improved by a quantum collision finding algorithm proposed by Ambainis [40] as follow.

Lemma 3.4. *([40], Theorem 9) Given a function $f : X \to Y$, and a symmetric binary relation $R \subseteq Y \times Y$ which can be computed in* $\mathrm{poly}(\log |Y|)$ *time steps where X and Y are some finite sets, we denote a collision by a distinct pair $x, x' \in X$ such that $(f(x), f(x')) \in R$. There exists a quantum algorithm that can find a collision with a constant probability when a collision exists, and always returns false when there does not exist a collision. The running time of the quantum algorithm is* $O\left(|X|^{2/3} \cdot \mathrm{poly}(\log |Y|)\right)$.

By this lemma we can identify a bad cycle within K random walks, with a query complexity of $O(|X|^{2/3}) = O((K \cdot L)^{2/3}) = O\left((\sqrt{N}\,\mathrm{poly}(\log N/\epsilon))^{2/3}\right) = O(N^{1/3}\,\mathrm{poly}(\log N/\epsilon))$. Next, we

establish the correctness of this algorithm and present its time complexity in the following theorem.

Theorem 3.5. *Algorithm 1 is a quantum algorithm that tests the clusterability of a signed graph with query complexity and running time* $O(N^{1/3}\,\mathrm{poly}(\log N/\epsilon))$.

Following our first result, we conclude that our quantum clusterability tester outperforms any classical clusterability tester.

Proof. First we prove that Algorithm 1 is indeed a clusterability tester. When G is clusterable, signifying the absence of one bad cycle, Algorithm 1 fails to discover a collision. Consequently, it returns true. On the contrary, when G is ϵ-far from clusterable, the assertion in Claim 14 from [1] suggests that the algorithm can pinpoint a bad cycle within the sampled random walks with a constant probability. This leads Algorithm 1 to return false with a constant probability.

To bound the time complexity (and hence query complexity), we need to bound the following quantities:

- The time required to evaluate the k-wise independent random variables.
- The time required to evaluate f.
- The number of queries required to find a collision.

For the first requirement, it takes $O\left(\mathrm{poly}(\log N/\epsilon)\right)$ time to evaluate a k-wise independent random variable, as indicated by Proposition 3.3. Moving to the second requirement, it is evident that each evaluation of f consumes time $\mathrm{poly}(\log N/\epsilon)$ since f is a procedure implementing a random walk, and the length of the walk is $L \in \mathrm{poly}(\log N/\epsilon)$. Concerning the last requirement, we are aware that detecting a collision requires $O\left(N^{1/3}\,\mathrm{poly}\left(\log N/\epsilon\right)\right)$ time, as derived from Lemma 3.4. In conclusion, the query and time complexity of Algorithm 1 is $O\left(N^{1/3}\,\mathrm{poly}\left(\log N/\epsilon\right)\right)$. □

4 PROOF DETAILS

In this section, we detail the construction and lemmas in Theorem 3.1. In Section 4.1, we generate two distinct families of graphs, each exhibiting different property of clusterability. In Section 4.2, we introduce two random processes that interact with an arbitrary algorithm $\mathcal{A}$ during the generation of graphs selected uniformly from the aforementioned families. In Section 4.3, we demonstrate that the statistical difference of query answer histories produced by $\mathcal{A}$ and these two random processes is bounded by the number of queries.

4.1 Graph construction

Here we detail the construction and analysis of the graph families $\mathcal{G}_1^N$ and $\mathcal{G}_2^N$.

4.1.1 Construction of two families of signed graphs $\mathcal{G}_\alpha^N$. We detail the construction of two families of signed graphs denoted as $\mathcal{G}_1^N$ and $\mathcal{G}_2^N$. In both families, each signed graph consists of N vertices, where N is a multiple of 10. Each vertex v is assigned a label p_v chosen from the set $\{0, 1, \ldots, 9\}$ in such a way that there are exactly $N/10$ vertices for each possible label.

For the edge set, we embed them in a manner such that each vertex is incident to precisely 6 edges. We construct edge sets based on cycles associated to a permutation $\sigma = (r_1\ r_2\ \ldots\ r_L)$, where $0 \le L \le 9$ and $r_l \in \{0, 1, \ldots, 9\}$ are distinct for $1 \le l \le L$, With some abuse of notation, we also denote by σ the bijective function $\sigma : \{r_1, r_2, \ldots, r_L\} \to \{r_1, r_2, \ldots, r_L\}$ defined as

$$\sigma(r_l) = \begin{cases} r_{l+1} & \text{if } l < L. \\ r_1 & \text{if } l = L. \end{cases}$$

With this notation, we define a family $\mathcal{D}^{\sigma}$ such that each member of this family is a union of cycles satisfying two properties: (i) the union of cycles contains all vertices in $[N]$ labeled with values from r_0 to r_L, and (ii) for each cycle $(v_1, v_2, \ldots, v_J)$ in the union of cycles, the label for each vertex must satisfy $p_{v_{j+1}} = \sigma(p_{v_j})$ for $1 \leq j \leq J$ (where we set $v_{J+1} = v_1$). We then employ these cycles to define the edge sets for the graphs in the family $\mathcal{G}_{\alpha}^{N}$. See Fig. 1 for an illustration for $\mathcal{G}_1^{40}$.

For $\mathcal{G}_1^N$: Each graph in $\mathcal{G}_1^N$ consists of one Hamiltonian cycle and two unions of cycles (we later comment on the particular choice of σ's):

- The Hamiltonian cycle $\in \mathcal{D}^{\sigma^{1\text{st}}}$ with $\sigma^{1\text{st}} = (0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9)$. We call this the arc cycle. All of its edges are positive, and we refer to these edges as arc edges.
- One union of cycles $\in \mathcal{D}^{\sigma^{2\text{nd}}}$ with $\sigma^{2\text{nd}} = (2\ 4\ 6\ 0\ 8\ 1\ 3\ 7\ 9\ 5)$,[1] with each of its edges being positively signed. We call these edges connecting edges.
- A second union of cycles $\in \mathcal{D}^{\sigma^{3\text{rd}}}$ with $\sigma^{3\text{rd}} = (1\ 6\ 3\ 8\ 5\ 0\ 7\ 2\ 9\ 4)$, and all edges are negatively signed.

For $\mathcal{G}_2^N$: In the family of graphs $\mathcal{G}_2^N$, each graph contains one Hamiltonian cycle and 12 unions of cycles:

- The Hamiltonian cycle $\in \mathcal{D}^{\sigma^{1\text{st}}}$ with each of its edges negatively signed.
- There are ten additional unions of cycles $\in \mathcal{D}^{\sigma^s}$ with $\sigma^s = (s)$ for s taking values in the set $\{0, 1, \ldots, 9\}$. These edges are positive.
- The last two unions of cycles are disjoint. One belongs to $\mathcal{D}^{\sigma^{10}}$ with $\sigma^{10} = (0\ 2\ 4\ 6\ 8)$. The other belongs to $\mathcal{D}^{\sigma^{11}}$ with $\sigma^{11} = (1\ 3\ 5\ 7\ 9)$. These edges are positive.

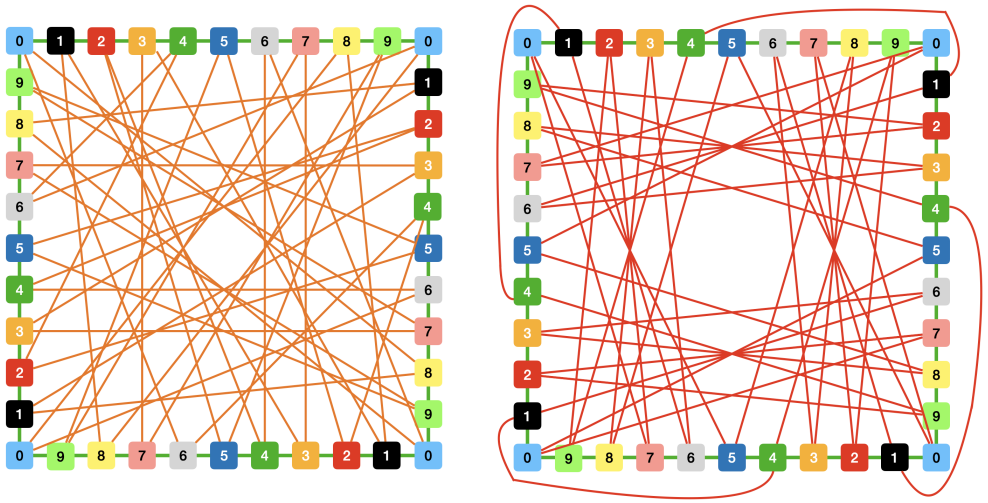

Fig. 1. An instance of $\mathcal{G}_1^{40}$. The green lines indicate the edges in one Hamiltonian cycle belonging to $\mathcal{D}^{\sigma^{1\text{st}}}$, the orange lines indicate the edges in one union of cycles belonging to $\mathcal{D}^{\sigma^{2\text{nd}}}$, and the red lines indicate the edges in one union of cycles belonging to $\mathcal{D}^{\sigma^{3\text{rd}}}$.

[1]The choice of $\sigma^{2\text{nd}}$ and $\sigma^{3\text{rd}}$ is not unique; we only require that these edge sets are disjoint when we fix the label of each vertex. This forbids picking for example $\sigma^{2\text{nd}} = (0\ 2\ 4\ 6\ 8\ 1\ 3\ 5\ 7\ 9)$, since the edges connecting vertices labeled 9 and 0 can be found in both $\mathcal{D}^{\sigma^{1\text{st}}}$ and $\mathcal{D}^{\sigma^{2\text{nd}}}$, meaning they are not disjoint. However, we could replace $\sigma^{2\text{nd}}$ with $(2\ 6\ 4\ 0\ 8\ 1\ 3\ 7\ 9\ 5)$, where exchanging 6 and 4 would not violate the disjoint property.

In every graph within $\mathcal{G}_\alpha^N$, each vertex is incident to precisely six edges, and these edges are labeled according to the following convention: For a pair of adjacent vertices, represented as v_j and v_{j+1} for $1 \le j \le J-1$, within any cycle $(v_1, v_2, \ldots, v_J)$, we label the edge connecting them as k for v_m and as $k+1$ for v_{m+1}, for some $k \in \mathbb{N}$. This labeling effectively associates an orientation to the cycle. More specifically, in a graph from $\mathcal{G}_1^N$:

- The edges in the Hamiltonian cycle from $\mathcal{D}^{\sigma^{1\text{st}}}$ are labeled with 1 and 2.
- For the edges in the union of cycles $\in \mathcal{D}^{\sigma^{2\text{nd}}}$, we use labels 3 and 4.
- For the edges in the union of cycles $\in \mathcal{D}^{\sigma^{3\text{rd}}}$, we use labels 5 and 6.

In the case of a graph from $\mathcal{G}_2^N$:

- The edges in the Hamiltonian cycle corresponding to $\mathcal{D}^{\sigma^{1\text{st}}}$ are labeled as 5 and 6.
- For the edges in the union of cycles within $\mathcal{D}^{\sigma^s}$ for s ranging from 0 to 9, we assign labels 1 and 2.
- For the edges in the union of cycles $\in \mathcal{D}^{\sigma^{10}}$ and $\mathcal{D}^{\sigma^{11}}$, we label them with 3 and 4.

4.1.2 Clusterability of $\mathcal{G}_\alpha^N$. We initially observe that all graphs within $\mathcal{G}_2^N$ are clusterable with the following rationale. All vertices with even (odd, respectively) labeling are interconnected via positive edges. Consequently, two connected components emerge: one component comprises all vertices with even labeling, and the other includes all vertices with odd labeling, each with positive edges. We further note that these two components can only be connected through negative edges. Hence, any graph within $\mathcal{G}_2^N$ satisfies the clusterability definition. As a result, all graphs in the second family are clusterable.

Regarding the graphs in $\mathcal{G}_1^N$, we will demonstrate that they are at least 0.01-far from being clusterable with probability at least $1 - \exp(-\Omega(N))$ in the following Proposition 4.1.

Proposition 4.1. *The graphs in $\mathcal{G}_1^N$ are 0.01-far from clusterable with probability at least $1 - \exp(-\Omega(N))$.*

Proof. We commence our proof by providing a description of the random process used to uniformly generate a graph denoted as g from $\mathcal{G}_1^N$. We begin by constructing the set of vertices $[N]$ and refer to the resulting (empty) graph as sg. The graph sg is equipped with its edges set through a three-step process:

(1) **(One Hamiltonian cycle):** In the first step, we uniformly select a Hamiltonian cycle from all possible Hamiltonian cycles on the vertices set $[N]$ and assign each vertex a label from the set $\{0, 1, \ldots, 9\}$, based on the rule of cycles $\in \mathcal{D}^{\sigma^{1\text{st}}}$. All edges constructed in this step are positive.
(2) **(Second edge set from $\mathcal{D}^{\sigma^{2\text{nd}}}$):** In the second step, we repeat the following processes N times:
   (a) Select an arbitrary vertex u_i that lacks an edge labeled as 3 where the index $i \in [N]$ represents the label of iteration.
   (b) Uniformly select a vertex v_i from a set that includes all vertices labeled as $\sigma^{2\text{nd}}(p_{u_i})$ and that lack an edge labeled as 4.
   (c) Add the edge (u_i, v_i).

   This adds an edge set from $\mathcal{D}^{\sigma^{2\text{nd}}}$. We make these edges positive.
(3) **(Third edge set from $\mathcal{D}^{\sigma^{3\text{rd}}}$):** Similar to the previous procedure, we add an edge set from $\mathcal{D}^{\sigma^{3\text{rd}}}$. We make these edges negative.

We call the resulting graph g, and note that g is a uniformly random element from $\mathcal{G}_1^N$. We proceed to observe that each graph g in $\mathcal{G}_1^N$ is inherently non-clusterable. Indeed, unless we remove

arc edges from the Hamiltonian cycle, every negative edge of a cycle in $\mathcal{D}^{\sigma^{3\text{rd}}}$ contributes to a bad cycle. We show that, with high probability over the random graph in $\mathcal{G}_1^N$, removing less than $0.01dN = 0.06N$ edges cannot make the graph clusterable.

More precisely, we will establish that after removing less than $0.06N$ arc edges, with high probability, all vertices remain connected through (positive) connecting edges in $\mathcal{D}^{\sigma^{2\text{nd}}}$, and so the graph cannot be clustered. To prove this, let us delve into a more detailed description of the random process used to generate a graph g.

In the first step, we construct a Hamiltonian cycle and eliminate $x < 0.06N$ arc edges, resulting in a graph with x components. There are $C_x^N = \binom{N}{x}$ possible possibilities for these x components.

During the first iteration of the second step, we select the arbitrary vertex u_0 from the component with the fewest vertices and designate this component as C. It becomes evident that, in the first iteration of step 2(c), the edge (u_0, v_0) connects component C to another component, with a probability exceeding $1/2$. Consequently, the number of components in the graph sg decreases by 1, and the number of vertices in C increases with a probability greater than $1/2$.

In the subsequent iterations, we select the vertex u_i for $2 \leq i \leq N$ based on the following rule: If the number of vertices labeled as $\sigma^{2\text{nd}}(p_{v_{i-1}})$ and lacking edges labeled as 4 within the component C is fewer than the number of vertices labeled as $\sigma^{2\text{nd}}(p_{v_{i-1}})$ not in C, then we set the vertex u_i equal to v_{i-1}. Subsequently, in 2(b) and 2(c), the process embeds an edge connecting u_i to some vertex v_i that is not a resident in C with a probability greater than $1/2$. Otherwise, we choose u_i from any arbitrary vertex labeled as $\sigma^{2\text{nd}}(p_{v_{i-1}})$ and not in C. Subsequently, in 2(b), the process selects v_i in C with a probability greater than $1/2$, as C has more vertices capable of connecting with u_i than the set of vertices not in C.

Consequently, the probability of the graph having more than one component can be bounded by the probability of obtaining fewer than x heads when flipping N unbiased coins. This probability can be bounded as follows:

$$\sum_{i=2}^{x} C_i^N \left(\frac{1}{2}\right)^{N-i} < 2^{N\cdot H(0.06)} 2^{-N} 2^{x} < 2^{N(-1+0.06+H(0.06))},$$

where $H(p) = -p\log(p) - (1-p)\log(1-p)$ is the (binary) entropy function.

At this point we removed only $x < 0.06N$ edges and we are permitted to remove an additional $0.06N - x$ connecting edges from sg. This corresponds to $0.06N - x$ tests where the coin flips tails (thus reducing the number of components by 1) can be taken into account for flips resulting in heads. In other words, the condition of having fewer than x heads can be extended to having fewer than $x + 0.06N - x$ heads when flipping N unbiased coins. Consequently, the probability that the resulting graph has more than one component, when $0.06N - x$ connecting edges are removed, can be bounded as:

$$\sum_{i=2}^{x+(0.06N-x)} C_i^N \left(\frac{1}{2}\right)^{N-i} < 2^{N(-1+0.06+H(0.06))}.$$

Given that there are $C_x^N < 2^{NH(0.06)}$ possible ways to construct x components in the first step, we can confidently assert that, after implementing step (2), all vertices in sg are interconnected by positive edges with a probability of at least $1 - \exp^{-\Omega(N)}$, even in cases where $0.06N$ positive edges (comprising of x arc edges and $0.06N - x$ connecting edges) were removed. The negative edges are present in the edge set in $\mathcal{D}^{\sigma^{3\text{rd}}}$, and each of them generates a bad cycle under the condition that only one component (only positive edges inside) is left after completing the second step in

the process. In other words, under this condition, we must remove all negative edges to make this graph clusterable. Consequently, the lemma follows. □

4.2 Random processes

Here we construct and analyze the random processes that play a key role in our lower bound. The first part describes the interaction of a random process P_α with an algorithm $\mathcal{A}$. The second part proves that P_α indeed generates a graph uniformly within $\mathcal{G}_\alpha^N$, as further elucidated in Proposition 4.2.

We will begin by defining the random process P_1, which involves two stages. The first stage will explain how P_1 interacts with an arbitrary T-query algorithm $\mathcal{A}$. The second stage will elaborate on how P_1 constructs a graph uniformly sampled from $\mathcal{G}_1^N$.

First stage of P_1: Given a query-answer history $h = [(q_1, a_1), (q_2, a_2), \ldots, (q_{t-1}, a_{t-1})]$ for $t \leq T$, we define a set of vertices $X_{p,i}$, which contains all vertices labeled with p in the history and lacking edge i. We also use the notation n_p to represent the count of vertices in this history that are labeled p. For each query $q_t = (v_t, i_t)$ made by $\mathcal{A}$, the actions of P_1 are defined as follows:

(1) If v_t is not in h, then P_1 labels v_t with a number $p \in \{0, 1, \cdots, 9\}$ with a probability of $\frac{(N/10)-n_p}{N-\left(\sum_{p=0}^{9} n_p\right)}$. Subsequently, P_1 answers this query as described in (2) below.

(2) If v_t belongs to h, there are two possible scenarios:

(a) If we can find the edge corresponding to q_t in h, then P_1 responds with the vertex connected to this edge. In other words, there exists an edge (v_t, u) in h such that (v_t, u) is labeled as i_t for vertex v_t, and P_1 responds with u. The query-answer history remains unchanged in this case.

(b) If the edge corresponding to $q_t = (v_t, i_t)$ does not exist in h, we follow these steps: Suppose, without loss of generality, that $i_t = 1$. We set the label $\sigma^{1\text{st}}(p_{v_t})$ as $\overline{p}$ and $\overline{i} = i_t + 1 = 2$. P_1 decides whether to uniformly select a vertex from $X_{\overline{p},\overline{i}}$ by flipping a coin with bias $\frac{|X_{\overline{p},\overline{i}}|}{N/10 - n_{\overline{p}} + |X_{\overline{p},\overline{i}}|}$ or to uniformly select a vertex not present in h, and assigns the label $\overline{p}$ to it. In either case, P_1 responds with the selected vertex u, and the edge (v_t, u) is signed positively. Subsequently, this edge (v_t, u) is added to the query-answer history h.
For the other case ($i_t = 2, 3, 4, 5, 6$), P_1 acts in a similar manner as described above, except for the assignment for $\overline{p}$, the assignment for $\overline{i}$, and the sign of the added edge. The added edge is positively signed for $i_t = 2, 3, 4$, and negatively signed for $i_t = 5, 6$. For $\overline{i}$, it is set to $i_t + 1$ for $i_t = 3, 5$ and to $i_t - 1$ for $i_t = 2, 4, 6$. The assignment for $\overline{p}$ is as follows:

$$\begin{cases} \overline{p} \leftarrow (\sigma^{1\text{st}})^{-1}(p_{v_t}) & \text{for } i_t = 2 \\ \overline{p} \leftarrow \sigma^{2\text{nd}}(p_{v_t}) & \text{for } i_t = 3 \\ \overline{p} \leftarrow (\sigma^{2\text{nd}})^{-1}(p_{v_t}) & \text{for } i_t = 4 \\ \overline{p} \leftarrow \sigma^{3\text{rd}}(p_{v_t}) & \text{for } i_t = 5 \\ \overline{p} \leftarrow (\sigma^{3\text{rd}})^{-1}(p_{v_t}) & \text{for } i_t = 6 \end{cases}$$

First stage of P_2: P_2 follows a similar process to P_1, with the only differences being the assignment for $\overline{p}$ as follows:

$$\begin{cases} \overline{p} \leftarrow p_{v_t} & \text{for } i_t = 1 \\ \overline{p} \leftarrow p_{v_t} & \text{for } i_t = 2 \\ \overline{p} \leftarrow p_{v_t} + 2 \pmod{10} & \text{for } i_t = 3 \\ \overline{p} \leftarrow p_{v_t} - 2 \pmod{10} & \text{for } i_t = 4 \\ \overline{p} \leftarrow \sigma^{1\text{st}}(p_{v_t}) & \text{for } i_t = 5 \\ \overline{p} \leftarrow (\sigma^{1\text{st}})^{-1}(p_{v_t}) & \text{for } i_t = 6 \end{cases}$$

Second stage of P_1: After answering all of these queries and generating a query-answer history $[(q_1, a_1), \ldots, (q_T, a_T)]$, P_1 proceeds with the following processes:

(1) Uniformly selecting a feasible way to embed the edges in h on a cycle. The embedding of these edges adhere to the following conditions: Each vertex is assigned a cycle position, i.e., an integer in $\{0, \ldots, N-1\}$, in a manner that ensures each vertex labeled with $p \in \{0, \ldots, 9\}$ is positioned at a position x such that $p \equiv x \pmod{10}$. This assignment implies that all acr edges (labeled 1, 2) are placed on the cycle, and edges labeled 3, 4, 5, 6 are excluded from the cycle.

(2) Randomly positioning all other vertices on the cycle, ensuring that each vertex v with label p_v is positioned at a position x such that $p_v \equiv x \pmod{10}$. Subsequently, all cycle edges are assigned a positive sign.

(3) In the end, uniformly selecting a feasible way to embed the edges sets in $\mathcal{D}^{\sigma^{2\text{nd}}}$ and $\mathcal{D}^{\sigma^{3\text{rd}}}$. All edges in the edges sets $\in \mathcal{D}^{\sigma^{2\text{nd}}}$ assigned a positive sign, while all edges in the edges sets $\in \mathcal{D}^{\sigma^{3\text{rd}}}$ are assigned a negative sign.

Second stage of P_2: P_2 follows a process similar to that of P_1, with few distinctions: In (2), we assign each cycle edge a negative sign. In (3), P_2 uniformly selects a feasible way to embed the edges sets $\in \mathcal{D}^{\sigma^s}$ for $s \in \{0, 1, \cdots, 11\}$, and assigns positive signs to these edges.

We will show that the above two processes uniformly generate a graph in the corresponding family in the next lemma.

Proposition 4.2. *For every algorithm $\mathcal{A}$ that uses T queries and for each $\alpha \in \{1, 2\}$, the process P_α uniformly generates graphs in $\mathcal{G}_\alpha^N$ when interacting with $\mathcal{A}$.*

Proof. We will use induction to prove this lemma. Consider that every probabilistic algorithm can be viewed as a distribution of deterministic algorithms. Therefore, it is sufficient to prove this lemma for any deterministic algorithm $\mathcal{A}$. The base case (i.e., $T = 0$) is correct because the query-answer history is empty, and the second stage in the process P_α uniformly generates a graph in $\mathcal{G}_\alpha^N$. We assume that the claim is true for $T-1$, and we will prove that the claim is also true for T. Let $\mathcal{A}'$ be the algorithm defined by stopping $\mathcal{A}$ before it asks the T^{th} query. By the inductive assumption, we know that P_α uniformly generates graphs in $\mathcal{G}_\alpha^N$ when P_α interacts with $\mathcal{A}'$. We will show that after P_α interacts with $\mathcal{A}$ and answers the T^{th} query, the second stage of P_α also uniformly generates graphs in $\mathcal{G}_\alpha^N$.

Assuming, without loss of generality, that the answer to the T^{th} query cannot be obtained from the query-answer history because this query does not provide additional information. Denote the T^{th} query of $\mathcal{A}$ as $q_T = (v_T, i_T)$ and consider all actions of the process P_1:

- **(Case 1) $\mathbf{i_T} \in \{3, 4, 5, 6\}$, and $\mathbf{v_T}$ in h:**
  Assume, without loss of generality, that $i_T = 3$ and denote $\overline{p} = \sigma^{2\text{nd}}(p_{v_T})$. The probability of P_1 connecting v_T to any vertex is independent of the specific order of vertices on the cycle

but depends on the labeling of the vertices. After considering all possible connecting edges carried out in the second stage following the interaction with $\mathcal{A}'$, it becomes evident that the only vertices in h to which v_T can connect are those in $X_{\overline{p},4}$. In any potential arrangement of the vertices on the cycle, there will be exactly $(N/10) - n_{\overline{p}}$ vertices labeled $\overline{p}$ and available for connection to v_T. This implies that the probability of v_T being connected to a vertex in $X_{\overline{p},4}$ is $\frac{|X_{\overline{p},4}|}{|X_{\overline{p},4}|+(N/10)-n_{\overline{p}}}$. Furthermore, when v_T is connected to a vertex in $X_{\overline{p},4}$, this vertex is uniformly distributed within $X_{\overline{p},4}$. Similarly, when connected to a vertex not in h, this vertex is uniformly distributed among the vertices not in h. These probabilities align with the definitions in P_1. Therefore, in Case 1, the induction step holds for P_1.

- **(Case 2) $\mathbf{i_T} \in \{1, 2\}$, and $\mathbf{v_T}$ in h:**
  Assume, without loss of generality, that $i_T = 1$ and denote $\overline{p}$ as $\sigma^{1\text{st}}(p_{v_T})$. In any valid embedding of the edges in h onto the cycle, it is evident that v_T can be adjacent with another vertex u in h only if u belongs to $X_{\overline{p},2}$. Moreover, when v_T is adjacent to a vertex in h, this vertex is uniformly distributed within $X_{\overline{p},2}$. If v_T is adjacency with another vertex u not in h, it is evident that the number of vertices labeled $\overline{p}$ but not in h is $(N/10) - n_{\overline{p}}$. Consequently, the probability of v_T being adjacent to some $u \in X_{\overline{p},2}$ is $\frac{|X_{\overline{p},2}|}{|X_{\overline{p},2}|+(N/10)-n_{\overline{p}}}$, and the probability of it being adjacent to a vertex not in h is $\frac{(N/10)-n_{\overline{p}}}{|X_{\overline{p},2}|+(N/10)-n_{\overline{p}}}$. These probabilities align with the definitions in P_1. Therefore, in this case, the induction step holds for P_1.
- **(Case 3) $\mathbf{v_T}$ is not in h:**
  We can reduce this case to case 1 and 2, provided that the label of v_T is selected with the appropriate probability. In the second stage, each vertex is randomly assigned label based on the proportion of missing vertices with that label. This essentially follows the assignment rule outlined in case (1) in the first stage of P_1.

For P_2, we omit the proof since it is similar to the argument in P_1, and this lemma follows.

□

4.3 Proof of Lemma 3.2

We may assume that $\mathcal{A}$ does not make a query whose answer can be obtained from its query answer history h since such a query does not update the h. Then, we begin the proof by proving the following proposition.

Proposition 4.3. *([32], Claim in lemma 7.4) Both in $\mathbf{D}_1^{\mathcal{A}}$ and in $\mathbf{D}_2^{\mathcal{A}}$, the total probability mass assigned to query-answer histories in which for some $t \leq T$ a vertex in h is returned as an answer to the t^{th} query is at most $10\delta^2$.*

Proof. We begin the proof by claiming that the probability of the event that the answer in the t^{th} query is a vertex in h is at most $20(t-1)/N$ for every $t \leq T$. The statement can be derived by observing that there are at most $2(t-1)$ vertices in h, and uses the definition of both processes. Then, the probability that the event occurs in an arbitrary query-answer history of length T is at most $\sum_{t=1}^{\delta\sqrt{N}} \frac{20(t-1)}{N} < 10\delta^2$. The proposition follows. □

From the proposition, we know that the edges in h will not form a cycle with probability at least $1 - 10\delta^2$. This event implies that for each query, these two processes pick a random vertex uniformly among the vertices, not in h. In addition, $\mathcal{A}$'s queries can only depend on the previous query-answer histories. Therefore, the distributions of the query-answer histories for these two processes are identical, except if we found a cycle, which happens with probability at most $10\delta^2$. Lemma 3.2 follows.

REFERENCES

[1] Florian Adriaens and Simon Apers. Testing cluster properties of signed graphs. In *Proceedings of the ACM Web Conference 2023*, pages 49–59, 2023.

[2] Oded Goldreich. Property testing. *Lecture Notes in Comput. Sci*, 6390, 2010.

[3] Dana Ron et al. Algorithmic and analysis techniques in property testing. *Foundations and Trends® in Theoretical Computer Science*, 5(2):73–205, 2010.

[4] Nir Ailon, Bernard Chazelle, Seshadhri Comandur, and Ding Liu. Estimating the distance to a monotone function. *Random Structures & Algorithms*, 31(3):371–383, 2007.

[5] Noga Alon and Asaf Shapira. Testing satisfiability. *Journal of Algorithms*, 47(2):87–103, 2003.

[6] Tali Kaufman and Madhu Sudan. Algebraic property testing: the role of invariance. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 403–412, 2008.

[7] Noga Alon, Seannie Dar, Michal Parnas, and Dana Ron. Testing of clustering. *SIAM Journal on Discrete Mathematics*, 16(3):393–417, 2003.

[8] Noga Alon and Michael Krivelevich. Testing k-colorability. *SIAM Journal on Discrete Mathematics*, 15(2):211–227, 2002.

[9] Noga Alon, Tali Kaufman, Michael Krivelevich, and Dana Ron. Testing triangle-freeness in general graphs. *SIAM Journal on Discrete Mathematics*, 22(2):786–819, 2008.

[10] Noga Alon and Asaf Shapira. Every monotone graph property is testable. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 128–137, 2005.

[11] Simon Apers and Alain Sarlette. Quantum fast-forwarding: Markov chains and graph property testing. *arXiv preprint arXiv:1804.02321*, 2018.

[12] Noga Alon. Testing subgraphs in large graphs. *Random Structures & Algorithms*, 21(3-4):359–370, 2002.

[13] Oded Goldreich, Shari Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *Journal of the ACM (JACM)*, 45(4):653–750, 1998.

[14] Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. *Machine learning*, 56(1):89–113, 2004.

[15] Dana Ron et al. Property testing: A learning theory perspective. *Foundations and Trends® in Machine Learning*, 1(3):307–402, 2008.

[16] Tom Gur, Min-Hsiu Hsieh, and Sathyawageeswar Subramanian. Sublinear quantum algorithms for estimating von neumann entropy. *arXiv preprint arXiv:2111.11139*, 2021.

[17] Ilias Diakonikolas, Homin K Lee, Kevin Matulef, Krzysztof Onak, Ronitt Rubinfeld, Rocco A Servedio, and Andrew Wan. Testing for concise representations. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*, pages 549–558. IEEE, 2007.

[18] Eldar Fischer, Guy Kindler, Dana Ron, Shmuel Safra, and Alex Samorodnitsky. Testing juntas. *Journal of Computer and System Sciences*, 68(4):753–787, 2004.

[19] Ashley Montanaro and Ronald de Wolf. A survey of quantum property testing. *arXiv preprint arXiv:1310.2035*, 2013.

[20] Harry Buhrman, Lance Fortnow, Ilan Newman, and Hein Röhrig. Quantum property testing. *SIAM Journal on Computing*, 37(5):1387–1400, 2008.

[21] Nilanjana Datta, Milan Mosonyi, Min-Hsiu Hsieh, and Fernando GSL Brandao. A smooth entropy approach to quantum hypothesis testing and the classical capacity of quantum channels. *IEEE transactions on information theory*, 59(12):8014–8026, 2013.

[22] Sebastien Bubeck, Sitan Chen, and Jerry Li. Entanglement is necessary for optimal quantum property testing. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 692–703. IEEE, 2020.

[23] Roland Nagy, Matthias Widmann, Matthias Niethammer, Durga BR Dasari, Ilja Gerhardt, Öney O Soykal, Marina Radulaski, Takeshi Ohshima, Jelena Vučković, Nguyen Tien Son, et al. Quantum properties of dichroic silicon vacancies in silicon carbide. *Physical Review Applied*, 9(3):034022, 2018.

[24] Andris Ambainis, Aleksandrs Belovs, Oded Regev, and Ronald de Wolf. Efficient quantum algorithms for (gapped) group testing and junta testing. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 903–922. SIAM, 2016.

[25] Oded Goldreich. Short locally testable codes and proofs (survey). In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 14, 2005.

[26] Luca Trevisan. Some applications of coding theory in computational complexity. *arXiv preprint cs/0409044*, 2004.

[27] Charanjit S Jutla, Anindya C Patthak, Atri Rudra, and David Zuckerman. Testing low-degree polynomials over prime fields. In *45th Annual IEEE Symposium on Foundations of Computer Science*, pages 423–432. IEEE, 2004.

[28] Tali Kaufman and Dana Ron. Testing polynomials over general fields. *SIAM Journal on Computing*, 36(3):779–802, 2006.

[29] Ting-Chun Lin and Min-Hsiu Hsieh. c 3-locally testable codes from lossless expanders. In *2022 IEEE International Symposium on Information Theory (ISIT)*, pages 1175–1180. IEEE, 2022.

[30] Pavel Panteleev and Gleb Kalachev. Asymptotically good quantum and locally testable classical ldpc codes. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 375–388, 2022.

[31] Irit Dinur, Shai Evra, Ron Livne, Alexander Lubotzky, and Shahar Mozes. Locally testable codes with constant rate, distance, and locality. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 357–374, 2022.

[32] Oded Goldreich and Dana Ron. Property testing in bounded degree graphs. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 406–415, 1997.

[33] Andrej Bogdanov, Kenji Obata, and Luca Trevisan. A lower bound for testing 3-colorability in bounded-degree graphs. In *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, pages 93–102. IEEE, 2002.

[34] Erik D Demaine, Dotan Emanuel, Amos Fiat, and Nicole Immorlica. Correlation clustering in general weighted graphs. *Theoretical Computer Science*, 361(2-3):172–187, 2006.

[35] Pieter W Kasteleyn. Dimer statistics and phase transitions. *Journal of Mathematical Physics*, 4(2):287–293, 1963.

[36] Frank Harary. On the notion of balance of a signed graph. *Michigan Mathematical Journal*, 2(2):143–146, 1953.

[37] Jure Leskovec, Daniel Huttenlocher, and Jon Kleinberg. Signed networks in social media. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 1361–1370, 2010.

[38] Jiliang Tang, Yi Chang, Charu Aggarwal, and Huan Liu. A survey of signed network mining in social media. *ACM Computing Surveys (CSUR)*, 49(3):1–37, 2016.

[39] James A Davis. Clustering and structural balance in graphs. *Human relations*, 20(2):181–187, 1967.

[40] Andris Ambainis, Andrew M Childs, and Yi-Kai Liu. Quantum property testing for bounded-degree graphs. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 365–376. Springer, 2011.

[41] Noga Alon, László Babai, and Alon Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *Journal of algorithms*, 7(4):567–583, 1986.