Thesis for the degree
Master of Science

Submitted to the Scientific Council of the
Weizmann Institute of Science
Rehovot, Israel

עבודת גמר (תזה) לתואר
מוסמך למדעים

מוגשת למועצה המדעית של
מכון ויצמן למדע
רחובות, ישראל

By
Yevgeny Levanzov

מאת
יבגני לבנזוב

על מציאת קליקות גדולות בגרפים מקריים ומקריים למחצה
On Finding Large Cliques in Random and Semi-Random Graphs

Advisor:
Prof. Robert Krauthgamer

מנחה:
פרופ' רוברט קראוטגמר

January 2018

טבת ה'תשע"ח

Abstract

A *clique* in a graph $G = (V, E)$ is a subset of the vertices, every two of which are connected by an edge. We study and analyze algorithms for finding large cliques in random and semi-random graph models. Let $G(n, p)$ denote the Erdős-Rényi random graphs model, where graphs have n vertices, and every possible edge occurs independently with probability p.

The first model we study is the well-known *Hidden Clique* problem, suggested independently by Jerrum and by Kučera. In this problem, the input graph G is constructed as follows. First, a random graph G' is drawn from $G(n, \frac{1}{2})$, and then a set K of $k = k(n)$ vertices of G' is chosen uniformly at random, and the subgraph induced on K is made into a clique. The goal is to find K in polynomial time. We study several variations of this problem, where the hidden object is slightly different, or the algorithm has access to additional information. In particular, we present the problem of finding a hidden *biclique* in a random graph; we prove that this problem is at least as hard as the Hidden Clique problem, and present a polynomial-time algorithm for finding a hidden biclique of size $k \geq \Omega(\sqrt{n})$. In another variant we study, two cliques are planted in a random graph, instead of just one, and the goal is to find at least one of the hidden cliques in polynomial time. We then analyze the Hidden Clique problem in several models of communication complexity. We present one-way protocols that use polylogarithmic communication, and we also show a communication lower bound. Finally, we study the Hidden Clique problem in a model of *statistical queries*, where the algorithm can make queries on subsets of the vertices and obtain aggregate information that may help in recovering the hidden clique.

The second model we study is semi-random graphs, where the input graph is constructed by a combination of random and adversarial decisions. We analyze a semi-random model, where the input graph H is constructed as follows. First, a random graph G is drawn from $G(n, \frac{1}{2})$, and then an adversary either adds or removes edges from $E(G)$ (in a restricted manner). We present a polynomial-time algorithm for finding, with high probability, the largest clique in such semi-random graph H whenever it has a clique of size $\Omega(\sqrt{n \log n})$, and we also show an impossibility result.

Acknowledgements

First and foremost, I would like to express my deep and sincere gratitude to my adviser, Prof. Robert Krauthgamer, for the unique opportunity to work with a scientist of the highest caliber, for always being available to help, for sharing his rich mathematical knowledge and creative ideas, and for emphasizing the importance of proper writing. Robi, it was a great pleasure to work with you!

I wish to thank my friends from the faculty, Yosef Pogrow, Ben Berger, Renan Gross, Yotam Hacohen, Ori Sberlo, Havana Rika, David Reitblat, Alon Ivtsan, Rani Izsak, and many others. You were a great source of inspiration!

I would also like to thank my friends Uri David, Oded Ross, Shmulik Itzhacki, Eyal Yablonko, Yuval Yaniv, Roee Zemach, and Max Rodshtein, for the great support during tough periods of my research journey.

Finally, I wish to thank my family for their great support.

Contents

1 Introduction

Graphs are fundamental objects in theoretical computer science. They are generally used to model relationships between pairs of objects, e.g., a friendship between members in a social network. A *clique* in a graph $G = (V, E)$ is a subset of the vertices, every two of which are connected by an edge. Let $\omega(G)$ denote the maximum number of vertices in a clique of G. The problems of computing or estimating $\omega(G)$ and of finding a clique of maximum size in an input graph G are fundamental problems in theoretical computer science. In particular, the problem of computing $\omega(G)$ exactly, called *Maximum Clique*, is well known to be NP-hard [Kar72]. Furthermore, approximating $\omega(G)$ for a graph G on n vertices within a factor of $n^{1-\varepsilon}$ is NP-hard, for every fixed $\varepsilon > 0$ [Hås96, Zuc06], and the best polynomial-time approximation ratio known for $\omega(G)$ is $O(\frac{n(\log\log n)^2}{(\log n)^3})$ [Fei04]. These facts suggest that an efficient algorithm for finding the largest clique in a general graph (i.e., worst-case instances) is unlikely to exist. Therefore, it is natural to study the performance of algorithms for this problem on average-case instances. A possible way to describe average-case instances is by probabilistic models, that is, randomly generated graphs. One major motivation for studying random graph models is that in real-life applications the input graphs often have certain random properties.

Let $G(n, p)$ denote the Erdős-Rényi random graphs model, where graphs have n vertices, and every possible edge occurs independently with probability p. It is known that $\omega(G)$ in a graph G drawn from $G(n, \frac{1}{2})$ is, *almost surely* (that is, with probability that approaches 1 as n tends to infinity), either r or $r + 1$, where $r = r(n) \approx 2\log_2 n$ (see Section 4.5 in [AS92]). There are a few polynomial-time algorithms that find in such random graph G (see, e.g., [KS98]), with high probability, a clique of size $(1 + o(1))\log_2 n$ (i.e., roughly half the size of the largest one). However, there is no known polynomial-time algorithm (not even a randomized one) that finds, almost surely, a clique of size $(1 + \varepsilon)\log_2 n$, for any fixed $\varepsilon > 0$. This is a long-standing open problem, suggested by Karp [Kar76].

Motivated by these facts, we study a few similar random models. The first model we have studied is the well-known *Hidden Clique* (also known as *Planted Clique*) problem, suggested independently by Jerrum [Jer92] and by Kučera [Kuč95]. In this problem, the input graph G is constructed as follows. First, a random graph G' is drawn from $G(n, \frac{1}{2})$, and then a set K of $k = k(n) \gg 2\log_2 n$ vertices of G' is chosen uniformly at random, and the subgraph induced on K is made into a clique, by connecting every pair of vertices in K by an edge. The goal is to find K in polynomial time. We denote by $G(n, \frac{1}{2}, k)$ the distribution of G. We actually study several variations of this problem, where the hidden object is slightly different,

e.g., a biclique or two cliques, or the algorithm has access to additional information.

The second model we study is semi-random graphs, where the input graph is constructed by a combination of random and adversarial decisions. A common variant of the semi-random model, introduced by Blum and Spencer [BS95], is when a random graph is chosen first, and then an adversary can modify this graph, but only in some bounded way, as otherwise he/she can convert the graph into a worst-case instance. A major motivation for studying such models is that they capture better real-life applications, where the input instances often have some non-random structures hidden inside, along with certain random properties. We analyze a semi-random model, where the input graph H is constructed as follows. First, a random graph G is drawn from $G(n, \frac{1}{2})$, and then an adversary either adds or removes edges from $E(G)$ (in a restricted manner). We present an efficient algorithm for finding, with high probability, a large clique in such semi-random graph H, and we also show an impossibility result.

Remark: Throughout, we use the notation $\omega(\cdot)$ in two different contexts, as follows. For a graph G, $\omega(G)$ denotes the maximum number of vertices in a clique of G, while for a function $f : \mathbb{N} \to \mathbb{R}_{\geq 0}$, $\omega(f(n))$ denotes the family of functions that dominate $f(n)$ asymptotically.

1.1 Variations of the Hidden Clique Problem

Let G be a graph drawn from $G(n, \frac{1}{2}, k)$. It is known that for $k = k(n) \gg 2\log_2 n$ the hidden clique K will almost surely be the largest clique in G.

Definition 1.1. (Solving the Hidden Clique problem). We say that algorithm A *solves* the Hidden Clique problem for $k = k(n) \gg 2\log_2 n$ if the following holds:

1. A is a randomized polynomial-time algorithm.
2. A finds the hidden clique in a graph G drawn from $G(n, \frac{1}{2}, k)$ with probability at least q, where $q > 0$ is some constant. The probability is taken over the choice of the input graph G and the randomness of the algorithm.

There are several polynomial-time algorithms that solve the Hidden Clique problem for all $k \geq \Omega(\sqrt{n})$ [AKS98, FK00, McS01, FR10, AV11, DGP14, DM15a], but no such algorithm is known for any $k = o(\sqrt{n})$. Moreover, it was shown that certain algorithmic approaches to the problem fail when $k = o(\sqrt{n})$, see [Jer92] regarding the Metropolis process, [FK03] regarding Lovász-Schrijver SDP hierarchies, and [DM15b, MPW15, HKP$^+$16, BHK$^+$16] regarding the Sum-of-Squares hierarchy. Motivated by these facts, we study and analyze several variations

of the Hidden Clique problem, in an attempt to shed more light on the hardness of the problem (both in positive and negative directions).

1.1.1 The Hidden Biclique Problem

One natural variation of the Hidden Clique problem is planting (in a random graph) a k-vertex graph, that is slightly sparser than a clique and has less structure. We focus on a natural candidate for such a graph - a *balanced biclique*.

Definition 1.2. Let $G = (V, E)$ be an undirected graph. Two disjoint sets of vertices $S_1, S_2 \subseteq V$ are called a *biclique* if $\{u, v\} \in E$ for every $u \in S_1, v \in S_2$ (i.e., a complete bipartite subgraph of G). The size of the biclique S_1, S_2 is defined as $|S_1| + |S_2|$. It is called *balanced* if $|S_1| = |S_2|$.

In the *Hidden Biclique* problem, the input graph $G = (V, E)$ is constructed as follows. First, a random graph G' is drawn from $G(n, \frac{1}{2})$. Then, two disjoint sets of vertices $S_1, S_2 \subseteq V$ of the same size $k/2$ are chosen independently at random, and for every $i \in S_1, j \in S_2$, the edge $\{i, j\}$ is added to G' (if not already there). The goal is to find the hidden biclique $S_1 \cup S_2$ in polynomial time (with high probability over the choice of G). We denote by $G(n, \frac{1}{2}, k/2, k/2)$ the distribution of G. In the *Bipartite Hidden Biclique* problem, the input (bipartite) graph $G = (V_1 \cup V_2, E)$ is constructed as follows. First, a random bipartite graph G' is drawn from $G(n, n, \frac{1}{2})$ (the distribution over bipartite graphs that have n vertices on each side, and every possible edge between the sides occurs independently with probability $\frac{1}{2}$). Then, two sets of vertices $S_1 \subseteq V_1, S_2 \subseteq V_2$ of the same size $k = k(n)$ are chosen independently at random, and for every $i \in S_1, j \in S_2$, the edge $\{i, j\}$ is added to G' (if not already there). The goal is to find the hidden biclique $S_1 \cup S_2$ in polynomial time (with high probability over the choice of G). We denote by $G(n, n, \frac{1}{2}, k, k)$ the distribution of G.

We point out that the problem of finding a *maximum-edge biclique* in a bipartite graph whenever it is unique was studied by Ames and Vavasis [AV11]. In particular, they present a polynomial-time algorithm that finds, with high probability, a hidden biclique in a random bipartite graph whenever each of the sides of the biclique is of size $\geq \Omega(\sqrt{n})$.

Our results. In Section 2 we compare the Hidden Clique problem and the Hidden Biclique problems. In particular, we show, by presenting polynomial-time reductions, that the Hidden Biclique problem is at least as hard as the Bipartite Hidden Biclique problem (with same k), and that the latter problem is at least as hard as the Hidden Clique problem (with same k). By combining the two results, we get that the Hidden Biclique problem is at least as hard

as the Hidden Clique problem (with same k). We have tried to show also reductions in the opposite direction, which would imply that the problems are computationally equivalent (up to polynomial factors), but unfortunately we did not manage to make any progress on that. We believe that this question is worth investigating further.

On the positive side, we present a linear-time algorithm that solves the Hidden Biclique problem for $k = \Omega(\sqrt{n})$, which we call algorithm FindBiclique. Formally, we prove the following theorem in Section 2.3.

Theorem 1.3. *For a sufficiently large constant $c > 0$, Algorithm FindBiclique finds, with probability at least $2/3$, the hidden biclique in a graph G drawn from $G(n, \frac{1}{2}, k/2, k/2)$ for $k = c\sqrt{n}$, and has running time $O(n^2)$.*

We note that algorithm FindBiclique can be extended to work for an arbitrary constant $c > 0$ using a standard technique presented in Section 2.3 of [AKS98], at the cost of a larger (polynomial) running time.

1.1.2 Planting Two Cliques in a Random Graph

We study a variation of the Hidden Clique problem, in which two cliques are planted in a random graph, instead of just one. Formally, the input graph $G = (V, E)$ is constructed as follows. First, a random graph G' is drawn from $G(n, \frac{1}{2})$. Then, two disjoint sets of vertices $K_1, K_2 \subseteq V$ of the same size $k = k(n) \gg 2\log_2 n$ are chosen independently at random, and each of them is made into a clique. We denote by $G(n, \frac{1}{2}, k + k)$ the distribution of G. The goal is to find at least one of the two hidden cliques in polynomial time (with high probability over the choice of G).

We point out that there exists a polynomial-time algorithm that finds, with high probability, both hidden cliques in a graph G drawn from $G(n, \frac{1}{2}, k+k)$ for all $k \geq \Omega(\sqrt{n})$ [McS01], but no such algorithm is known for any $k = o(\sqrt{n})$.

Our results. In Section 3 we show that if there exists a polynomial-time algorithm that finds, with high probability, at least one hidden clique in a graph G drawn from $G(n, \frac{1}{2}, k+k)$, then we can solve the Hidden Clique problem for the same $k = k(n) \gg 2\log_2 n$, with probability at least $1/2 - o(1)$ (which is not known how to be done for $k = o(\sqrt{n})$). This basically shows that planting two cliques instead of one just makes the problem harder. We have tried to show also a reduction in the opposite direction, which would imply that the problems are computationally equivalent (up to polynomial factors), but unfortunately we did not manage to make any progress on that. We believe that this question is worth investigating further.

1.1.3 The Hidden Clique Problem in a Communication Complexity Setting

We analyze and compare the communication complexity of the Hidden Clique problem in two models. As usual, there are two parties, Alice and Bob, where Bob's input is a graph G drawn from $G(n, \frac{1}{2}, k)$ for $k = k(n) \gg 2\log_2 n$, whereas Alice will have a different input in each of the models. The goal of Bob is to eventually recover the hidden clique in G. The communication between the parties is restricted to be one-way (i.e., Alice sends a single message to Bob) and the two parties have a shared randomness. We also note that unlike in the conventional models of communication complexity, where both parties are computationally unbounded, here only Alice is all-powerful, while Bob is polynomially bounded, as otherwise he could recover the hidden clique by himself (in time $n^{O(\log n)}$).

In the first model, Alice's input is the underlying random graph of G (i.e., before the clique was planted). Observe that Alice does not know where is the hidden clique in Bob's graph (information-theoretically), and thus her unbounded computational power is not important here. However, the difference between her graph and Bob's graph may give a valuable information that can help Bob to recover the hidden clique.

In the second model, Alice's input is a graph obtained by randomly permuting the labels of the vertices of G. Observe that Alice can find the hidden clique in her graph, as she is computationally unbounded, and if the vertex labels were not permuted, she could send the labels of the hidden clique vertices to Bob, using $O(\log^2 n)$ bits of communication (as it is enough to have $3\log_2 n$ random vertices from the hidden clique to be able to recover it, by Lemma 4.5).

We argue that in both models, Alice is able to send Bob a single and relatively short message, consisting of a certain information about her input graph, which will enable Bob to find the hidden clique in G in polynomial time.

Our results. In Section 4 we present two different protocols in the first model. The first one is based on sums of degrees and uses $O(\log^3 n)$ bits of communication, whereas the second is based on ℓ_0-samplers and uses $O(\log^5 n)$ bits of communication. Then, we present a protocol in the second model, which is based on the degree sequences of neighbors and uses $O(\log^2 n)$ bits of communication. Finally, we show a lower bound of $\omega(\log n)$ on the communication complexity of protocols where Bob's only input is a graph G drawn from $G(n, \frac{1}{2}, k)$ for $k = k(n) \gg 2\log_2 n$, regardless of the input of Alice, assuming hardness of the Hidden Clique problem for $k = o(\sqrt{n})$ (i.e., assuming that there is no algorithm that solves the Hidden Clique problem for $k = o(\sqrt{n})$).

1.1.4 Finding a Hidden Clique Using Statistical Queries

We study the Hidden Clique problem in a model of *statistical queries*, where the algorithm can make queries on subsets of the vertices and obtain aggregate information that may help in recovering the hidden clique. The queries can be performed either on the underlying random graph (i.e., before the clique was planted) or on the input graph itself. The motivation for this model is to examine the following question: Can statistical information of a certain type help to recover the hidden clique? And if so, how much such information do we need?

In Section 5 we study the following type of statistical queries. For an input graph G drawn from $G(n, \frac{1}{2}, k)$ for $k = k(n) \gg 2\log_2 n$, an algorithm can choose a subset $S \subseteq V$, and make a query for the sum of degrees of the vertices in S, in the underlying random graph G' (i.e., before the clique was planted).

Our results. In Section 5 we show a lower bound of $\omega(\frac{\log n}{\log k})$ on the query complexity, assuming hardness of the Hidden Clique problem for $k = o(\sqrt{n})$ (i.e., assuming that there is no algorithm that solves the Hidden Clique problem for $k = o(\sqrt{n})$). In addition, we present a polynomial-time algorithm for finding the hidden clique, which has query complexity $O(\log^2 n)$.

1.2 Finding Large Cliques in Semi-Random Graphs

We focus on studying semi-random graphs that are constructed as follows. First, a random graph G is drawn from $G(n, \frac{1}{2})$. Then, an adversary adds to it a certain amount of edges, resulting in a semi-random graph we denote by H, in such a way that $\omega(G) \ll \omega(H)$. Furthermore, an additional limitation is enforced on the adversary: every vertex outside the largest clique in H must have a degree that is smaller than the degree of any vertex inside the largest clique in H, unless it was bigger initially (i.e., in G).

Our results. In Section 6 we show that if H is a semi-random graph (constructed as above) with $\omega(H) \geq c\sqrt{n \log n}$ (for a sufficiently large constant $c > 0$), then we can find the largest clique in H in polynomial time. In addition, we show that there is no polynomial-time algorithm that approximates the largest clique in H within a factor of $2 - \varepsilon$, for every fixed $\varepsilon > 0$, assuming hardness of the Hidden Clique problem for $k = o(\sqrt{n})$ (i.e., assuming that there is no algorithm that solves the Hidden Clique problem for $k = o(\sqrt{n})$).

2 The Hidden Biclique Problem

In this section we study a variation of the Hidden Clique problem, in which instead of planting a clique in a random graph, we plant a *balanced biclique* (recall Definition 1.2).

We start by pointing out that the problem of finding the maximum balanced biclique in a graph, even a balanced bipartite one (i.e., a bipartite graph in which both parts, which are given, are of the same size) is NP-hard [Joh87]. Furthermore, it was proven that approximating the maximum balanced biclique in a graph on n vertices within a factor of $2^{(\log n)^{\delta}}$ for some $\delta > 0$ is NP-hard, under the assumption that 3-SAT cannot be solved in time $O(2^{n^{3/4+\varepsilon}})$ for some $\varepsilon > 0$ [FK04]. These facts suggest that an efficient algorithm for finding the largest balanced biclique in a general graph is unlikely to exist, which gives an additional motivation to study the particular variations of the Hidden Clique problem, which are presented in Section 1.1.1 - the Hidden Biclique and the Bipartite Hidden Biclique problems.

Previously, a similar problem was studied by Ames and Vavasis [AV11]. Specifically, they analyze the problem of finding a maximum-edge biclique (i.e., a complete bipartite subgraph $K_{m,n}$ that maximizes the product mn) in a bipartite graph whenever it is unique. In particular, they present a polynomial-time algorithm that finds, with high probability, a hidden biclique in a random bipartite graph (not necessarily balanced) whenever each of the sides of the biclique is of size $\geq \Omega(\sqrt{n})$ (and they differ only by a constant factor). Therefore, as a special case, their algorithm finds the hidden biclique in a graph G drawn from $G(n, n, \frac{1}{2}, k, k)$ for all $k \geq \Omega(\sqrt{n})$. The algorithm uses a convex relaxation that is based on a minimization of the *nuclear norm* of a matrix.

It is well known that, with high probability, the size of the maximum clique in a graph G drawn from $G(n, \frac{1}{2})$ is roughly $2\log_2 n$ (i.e., $\omega(G) \approx 2\log_2 n$), see Section 4.5 in [AS92]. It is also known that, with high probability, the size of the maximum balanced biclique both in a graph G drawn from $G(n, n, \frac{1}{2})$ and in a graph G drawn from $G(n, \frac{1}{2})$ is at most $4\log_2 n$ (i.e., the number of vertices on each side is at most $2\log_2 n$), see [DKST01]. Thus, if we plant a clique of size $r = r(n) \gg 2\log_2 n$ in a random graph $G \in G(2n, \frac{1}{2})$, then it can be shown that, with high probability, it will be the unique maximum clique in the resulting graph. Similarly, if we plant a biclique of size $r + r$ with $r = r(n) \gg 2\log_2 n$ in a random bipartite graph $G \in G(n, n, \frac{1}{2})$ or in a random graph $G \in G(2n, \frac{1}{2})$, then it can be shown that, with high probability, it will be the unique maximum biclique in the resulting graphs.

In Section 2.1 we show, by presenting a polynomial-time reduction, that the Hidden Biclique problem is at least as hard as the Bipartite Hidden Biclique problem (with same k), and in Section 2.2 we show, by presenting a polynomial-time reduction, that the latter

problem is at least as hard as the Hidden Clique problem (with same k). By combining the two results, we get that the Hidden Biclique problem is at least as hard as the Hidden Clique problem (with same k), which is formalized in Corollary 2.3. On the positive side, in Section 2.3 we present a linear-time algorithm that solves the Hidden Biclique problem for $k = \Omega(\sqrt{n})$, which we call algorithm FindBiclique.

2.1 Hardness of the Hidden Biclique Problem

In this section, we present a polynomial-time reduction from the Bipartite Hidden Biclique problem to the Hidden Biclique problem, with the same size of the planted biclique $k = k(n)$. Formally, we prove the following theorem.

Theorem 2.1. *If there exists a randomized polynomial-time algorithm that finds, with high probability, the hidden biclique in a graph G drawn from $G(2n, \frac{1}{2}, k, k)$, then there exists a randomized polynomial-time algorithm that finds, with high probability, the hidden biclique in a (bipartite) graph H drawn from $G(n, n, \frac{1}{2}, k, k)$, for all $k = k(n) \gg 2\log_2 n$.*

Proof: We assume that there exists a randomized polynomial-time algorithm, denoted by A, that finds, with high probability, the hidden biclique in a graph G drawn from $G(2n, \frac{1}{2}, k, k)$ (we assume that if A fails to find the hidden biclique, then it returns an empty set), and present a randomized polynomial-time algorithm that finds, with high probability, the hidden biclique in a graph H drawn from $G(n, n, \frac{1}{2}, k, k)$, which we call algorithm FindBC.

Algorithm 1 FINDBC (H, k)

Input: A graph $H = (V_1 \cup V_2, E)$ drawn from $G(n, n, \frac{1}{2}, k, k)$, with $k = k(n) \gg 2\log_2 n$. We assume that $V_1 = [n]$ and $V_2 = \{n+1, \ldots, 2n\}$.
Output: The hidden biclique in H (w.h.p.).

1. Set $G = H$.
2. For $i \in \{1, 2\}$, choose $H_i \leftarrow G(n, \frac{1}{2})$ and set $G[V_i] = H_i$ (note that initially $G[V_1], G[V_2]$ have no edges).
3. Choose a random permutation $\sigma : [2n] \to [2n]$ (note that $V_1 \cup V_2 = [2n]$), and rename the vertices of G according to σ (formally, the set of edges of G will now be $E'(G) = \{\{\sigma(i), \sigma(j)\} \mid i, j \in [2n] \wedge \{i, j\} \in E(G)\}$).
4. Execute $Q \leftarrow A(G)$.
5. If $Q \neq \emptyset$, **return** Q; Else, **return** "FAIL"

Analysis:
The core of the analysis will be to show that the distribution of the graph G, generated in steps 1-3 by a series of random choices, is identical to the distribution $G(2n, \frac{1}{2}, k, k)$. We observe that a graph G' drawn from $G(2n, \frac{1}{2}, k, k)$ is basically generated in two phases:

1. Choosing $G'' \leftarrow G(2n, \frac{1}{2})$, which is a series of independent choices for every pair of vertices $u \neq v$, whether the edge $\{u, v\}$ will occur or not (with equal probability).

2. Choosing two disjoint sets of vertices of size k from $V(G'')$ independently at random, and making them into a biclique.

As the random choices made in each of the phases are independent of one another, they can be done in any order, preserving the same distribution. Thus, we can generate $G' \in G(2n, \frac{1}{2}, k, k)$ in a series of the following steps:

1. Start with an empty graph G'' on the vertex set $[2n]$.

2. Select two arbitrary disjoint sets of vertices of size k, and make them into a biclique.

3. For each pair of vertices $u \neq v$ which is not an edge of the chosen biclique, make $\{u, v\}$ to be an edge with probability $1/2$, independently of all other pairs.

4. Choose a random permutation $\sigma : [2n] \to [2n]$, and rename the vertices of G'' according to σ.

5. Return the resulting graph, denoted by G'.

We note that renaming the vertices according to σ in step 3 is equivalent to choosing the vertices of the hidden biclique randomly and independently (i.e., identical distributions), as a random permutation σ provides that the labels of the vertices of the already chosen (arbitrarily) hidden biclique are uniformly distributed over all the subsets of labels of size $2k$ from $[2n]$. Clearly, permuting the labels of the vertices does not change the distribution of the randomly chosen edges. Thus, the distribution of G' is identical to the distribution $G(2n, \frac{1}{2}, k, k)$.

Looking upon algorithm FindBC, we have that after step 2, all the edges (and non-edges) of G apart from those of the hidden biclique (in H) were chosen at random. Thus, the first three generation steps described above are completed. Finally, permuting the labels of the vertices in step 3 completes the fourth generation step (we note that permuting the labels of the vertices is essential, so that the vertices of the planted biclique in G will be uniformly

distributed over all subsets of labels of size $2k$ from $[2n]$). Therefore, the distribution of the graph G, produced by the algorithm, is identical to the distribution $G(2n, \frac{1}{2}, k, k)$, and thus, with high probability, algorithm A will output (in step 4) the hidden biclique in G. As the vertices of the hidden biclique in G are the same vertices of the hidden biclique in H, algorithm FindBC will output, with high probability, the hidden biclique in H.

We note that algorithm FindBC succeeds with high probability over the choice of the input graph H, and also with high probability over its randomness. Thus, the total failure probability is $o(1)$, by a union bound.

Running time:
Let $T(n) = \text{poly}(n)$ be the running time of algorithm A. Clearly, steps 1-3 take $O(n^2)$ time, and step 4 takes $T(n)$ time. Thus, the total running time of algorithm FindBC is $O(n^2 + T(n)) = \text{poly}(n)$.
This completes the proof. ■

2.2 Hardness of the Bipartite Hidden Biclique Problem

In this section, we present a polynomial-time reduction from the Hidden Clique problem to the Bipartite Hidden Biclique problem, with the same size of the planted clique/biclique $k = k(n)$. Formally, we prove the following theorem.

Theorem 2.2. *If there exists a randomized polynomial-time algorithm that finds, with high probability, the hidden biclique in a (bipartite) graph G drawn from $G(n, n, \frac{1}{2}, k, k)$, then there exists a randomized polynomial-time algorithm that finds, with high probability, the hidden clique in a graph H drawn from $G(2n, \frac{1}{2}, 2k)$, for all $k = k(n) \gg 2\log_2 n$.*

Proof: As there exist polynomial-time algorithms that find, with high probability, a hidden clique of size $k \geq \Omega(\sqrt{n})$ in a graph G drawn from $G(2n, \frac{1}{2})$, we can assume that $2\log_2 n \ll k = k(n) = o(\sqrt{n})$.

We assume that there exists a randomized polynomial-time algorithm, denoted by A, that finds, with high probability, the hidden biclique in a graph G drawn from $G(n, n, \frac{1}{2}, k, k)$ (we assume that if A fails to find the hidden biclique, then it returns an empty set), and present a randomized polynomial-time algorithm that finds, with high probability, the hidden clique in a graph H drawn from $G(2n, \frac{1}{2}, 2k)$, which we call algorithm FindClique.

Algorithm 2 FindClique (H, k)

Input: A graph $H = (V, E)$ drawn from $G(2n, \frac{1}{2}, 2k)$, with $2\log_2 n \ll k = k(n) = o(\sqrt{n})$.
Output: The hidden clique in H (w.h.p.).

1. Set $G = H$.
2. Repeat the following $\lfloor \sqrt{n} \rfloor$ times (independently):
   2.1. Choose a random subset of vertices $S \subseteq V(G)$ of size n, and let $T = V(G) \backslash S$ (i.e., (S, T) is a random equipartition of the vertices of G).
   2.2. Remove all edges with both endpoints either in S or in T, resulting in a (balanced) bipartite graph (on the same vertex set).
   2.3. Execute $Q \leftarrow A(G)$.
   2.4. If $Q \neq \emptyset$, **return** Q.
3. **return** "FAIL".

Analysis:

Let $K \subseteq V$ be the set of vertices of the planted clique in H. We denote $K_1 = S \cap K$ and $K_2 = T \cap K$. Clearly, the induced subgraph $G[K_1 \cup K_2]$ is a biclique in G (not necessarily balanced). We observe that the random variables $|K_i|$ for $i = 1, 2$ have a hypergeometric distribution with parameters $\mathrm{HG}(2n, 2k, n)$. Thus, we have that $\mathbb{E}[|K_i|] = k$ and $\sigma(|K_i|) \leq \sqrt{k/2}$ for $i = 1, 2$. We now show a lower bound on the probability of having a balanced biclique in G of size $2k$ (i.e., $|K_1| = |K_2| = k$). We have that:

$$\Pr[|K_1| = k] = \Pr[|K_2| = k] = \frac{\binom{2k}{k}\binom{2n-2k}{n-k}}{\binom{2n}{n}} = \frac{\binom{2k}{k}\binom{2(n-k)}{n-k}}{\binom{2n}{n}}.$$

Using Stirling's approximation formula, $m! \sim \sqrt{2\pi m}\left(\frac{m}{e}\right)^m$, we get the following approximation for the largest binomial coefficient:

$$\binom{m}{m/2} = \Theta(2^m/\sqrt{m}). \tag{1}$$

Applying equation (1), we have that:

$$\Pr[|K_1| = k] = \Pr[|K_2| = k] = \frac{\binom{2k}{k}\binom{2(n-k)}{n-k}}{\binom{2n}{n}} \geq c \cdot \frac{\sqrt{n}}{\sqrt{k}\sqrt{n-k}} \geq \frac{c}{\sqrt{k}},$$

where $c > 0$ is some constant.

Therefore, with probability at least $c/\sqrt{k}$ we succeed to produce (in steps 2.1-2.2) a graph G with a balanced biclique of size $2k$. We observe that in such case, G is a random bipartite graph with a randomly planted balanced biclique of size $2k$. This is true, as the choice of the equipartition (S,T) is independent of the choice of the edges (of H, and thus of G). We can think of it as if it was chosen before the edges of H, and now examine the edges of G. We observe that all the edges (and non-edges) of G (which are between S and T) with at least one endpoint outside K were chosen uniformly at random (in H, and thus in G), and also that the vertices of the planted biclique are uniformly distributed on each side, as the choice of the vertices of the planted clique in H was uniform, and so is the choice of the random equipartition. To conclude, in each iteration, with probability at least $c/\sqrt{k}$ we produce a graph G whose distribution is identical to the distribution $G(n,n,\frac{1}{2},k,k)$ (where the K_1,K_2 is the hidden biclique). Thus, we have that:

$$\Pr[\text{alg. FindClique fails in all iterations}] \leq \left(1-\tfrac{c}{\sqrt{k}}\right)^{\sqrt{n}} \underset{k=o(\sqrt{n})}{\leq} \left(1-\tfrac{c}{n^{1/4}}\right)^{\sqrt{n}} \leq e^{-cn^{1/4}} = o(1).$$

Therefore, with high probability, algorithm FindClique will succeed to produce a graph G which is distributed according to $G(n,n,\frac{1}{2},k,k)$, and thus, with high probability, algorithm A will output (in step 2.3) the hidden biclique in G. As the vertices of the hidden biclique in G are the same vertices of the hidden clique in H, algorithm FindClique will output, with high probability, the hidden clique in H.

We note that algorithm FindClique succeeds with high probability over the choice of the input graph H, and also with high probability over its randomness. Thus, the total failure probability is $o(1)$, by a union bound.

Running time:
Let $T(n) = \text{poly}(n)$ be the running time of algorithm A. Clearly, steps 2.1-2.2 take $O(n^2)$ time, and step 2.3 takes $T(n)$ time. Thus, the total running time of algorithm FindClique is $O(\sqrt{n}\cdot(n^2+T(n))) = \text{poly}(n)$.
This completes the proof. ■

Corollary 2.3. *If there exists a randomized polynomial-time algorithm that finds, with high probability, the hidden biclique in a graph G drawn from $G(2n,\frac{1}{2},k,k)$, then there exists a randomized polynomial-time algorithm that finds, with high probability, the hidden clique in a graph H drawn from $G(2n,\frac{1}{2},2k)$, for all $k = k(n) \gg 2\log_2 n$.*

Proof: Follows from combining Theorem 2.1 and Theorem 2.2. ■

2.3 Finding a Hidden Biclique in a Random Graph

In this section we present a linear-time algorithm for finding, with high probability, a hidden biclique of size $k \geq c\sqrt{n}$ (for a sufficiently large constant $c > 0$) in a random graph drawn from $G(n, \frac{1}{2})$, which we call algorithm FindBiclique. This algorithm is a modification of the LDR (Low Degree Removal) algorithm of Feige and Ron [FR10], which finds a hidden clique of size $k \geq c\sqrt{n}$ (for a sufficiently large constant $c > 0$) in a random graph drawn from $G(n, \frac{1}{2})$. Our modification of the LDR algorithm does not directly address the fact that we are trying to find a hidden biclique, but rather performs a logic that is purely based on the degrees of the vertices, exploiting the fact that the degree of every vertex of the hidden biclique increases roughly by $k/4$ when the biclique is planted. We also observe that for $k \geq c\sqrt{n \log n}$ (for a sufficiently large constant $c > 0$), the vertices of the hidden biclique would almost surely be the ones with the largest degrees in the graph (as the standard deviation of a degree of a vertex is $O(\sqrt{n})$), and hence it is easy to find them efficiently, without using our algorithm (similar observation was presented by Kučera [Kuč95] regarding the Hidden Clique problem).

We point out that a polynomial-time algorithm that finds a hidden biclique of size $k \geq c\sqrt{n}$ in a random bipartite graph was presented by Ames and Vavasis [AV11]. However, our algorithm finds a hidden biclique of size $k \geq c\sqrt{n}$ in a general random graph, which can be only a harder problem (see Theorem 2.1).

Theorem 2.4. (Theorem 1.3 restated). *For a sufficiently large constant $c > 0$, Algorithm FindBiclique finds, with probability at least $2/3$, the hidden biclique in a graph G drawn from $G(n, \frac{1}{2}, k/2, k/2)$ for $k = c\sqrt{n}$, where the probability is taken over the choice of the input graph G. The running time of the algorithm is $O(n^2)$.*

Remark: Algorithm FindBiclique can be extended to work for an arbitrary constant $c' > 0$ using a standard technique presented in Section 2.3 of [AKS98], as follows. Let $c > 0$ be a constant from Theorem 2.4. In order to find a hidden biclique of size $k = c'\sqrt{n}$ in a graph G drawn from $G(n, \frac{1}{2}, k/2, k/2)$, guess a set T of $t \approx \log_2(c/c')$ vertices from the hidden biclique by exhaustive search, and denote by S the set of their common neighbors. Then, execute algorithm FindBiclique on $G[S]$, getting an output Q, and return $Q \cup T$ (if it is a biclique of size $k = c'\sqrt{n}$). The total running time is $n^{O(\log(c/c'))}$.

2.3.1 The FindBiclique Algorithm

We now present the FindBiclique algorithm, which works in two phases.

Algorithm 3 FINDBICLIQUE (G, k)

Input: A graph $G = (V, E)$ drawn from $G(n, \frac{1}{2}, k/2, k/2)$, with $k = c\sqrt{n}$ ($c > 0$ is a sufficiently large constant).
Output: The hidden biclique in G (w.h.p.).

The removal phase.

1. Set $r = 0$ and $G_0 = G$.
2. **while** $(|V(G_r)| > 4k/5)$ **do**
   2.1. Remove from G_r the vertex of lowest degree (breaking ties in favor of vertex with lower index). Call the remaining graph G_{r+1}.
   2.2. Increment r.

The inclusion phase.

3. Rename those vertices of G that are not in the output G_r of the removal phase to $v_r, \ldots, v_1$. Set $t = r$ and let Q_r be the set of vertices of G_r.
4. **while** $(t > 0)$ **do**
   4.1. If v_t is connected (in G) to at least $21k/40$ vertices of Q_r, then $Q_{t-1} = Q_t \cup \{v_t\}$; Else, $Q_{t-1} = Q_t$.
   4.2. Decrement t.
5. **return** Q_0.

Remark: Given the set of vertices of the hidden biclique (outputted by algorithm Find-Biclique), denoted by Q, one can find their bipartition efficiently by simply computing the connected components of the graph $\overline{G[Q]}$ (i.e., the complement graph of the induced subgraph $G[Q]$). With high probability, there are exactly two connected components (of the same size), and they are the two parts of the hidden biclique.

2.3.2 Analysis of the Algorithm

Here we prove Theorem 2.4. We will use the following notation (similar to the one used for the LDR algorithm). Let $S_1 \cup S_2 \subseteq V$ be the set of vertices of the hidden biclique; let $n_r = n - r$ denote the number of vertices in G_r; let K_r denote the set of vertices from $S_1 \cup S_2$

in G_r, and let k_r denote their number; let F_r denote the set of other vertices in G_r, and let $f_r = n_r - k_r$ denote their number. Lastly, we use $F = F_0$ to denote the set of vertices which are not part of the hidden biclique (in G).

In addition, we note that all probabilities are taken over the choice of G (the algorithm is deterministic).

Proposition 2.5. *Let $c_0 > 0$ be a sufficiently large constant, and let $i \in \{1, 2\}$. Then, with high probability, for every vertex $v \in V$, its degree into S_i in G' (the underlying random graph) is in the range $[k/4 - c_0\sqrt{k \log n}, k/4 + c_0\sqrt{k \log n}]$.*

Proof: Standard combination of Chernoff-Hoeffding bounds and union bound. ■

Corollary 2.6. *Let $c_0' > 0$ be a sufficiently large constant. Then, with high probability, for every vertex $v \in V$, its degree into $S_1 \cup S_2$ in G' (the underlying random graph) is in the range $[k/2 - c_0'\sqrt{k \log n}, k/2 + c_0'\sqrt{k \log n}]$.*

Lemma 2.7. *Let $c_1 > 0$ be a sufficiently large constant. Then, with high probability, at every step r in which $f_r \geq \sqrt{n}$, the vertex of lowest degree in G_r has degree at most $n_r/2 + c_1\sqrt{n}$.*

Proof: An equivalent method for generating G is by choosing the sets $S_1 \subseteq V_1, S_2 \subseteq V_2$ first (randomly and independently), making then into a biclique, and then every remaining pair of vertices $u \neq v \in V$ (i.e., $(u, v) \notin (S_1 \times S_2) \cup (S_2 \times S_1)$) are connected by an edge independently with probability $1/2$. Consider the situation after choosing S_1, S_2. For arbitrary values f_r in the range $\sqrt{n} \leq f_r \leq n - k$ and k_r in the range $0 \leq k_r \leq k$, pick an arbitrary set of f_r vertices from F and an arbitrary set of k_r vertices from $S_1 \cup S_2$. Now, the expected number of endpoints of edges touching F_r is exactly $f_r(n_r - 1)/2$, the standard deviation is $O(\sqrt{f_r n_r})$, and the probability of deviating from the expectation by more than $\alpha\sqrt{n}\sqrt{f_r n_r}$ is at most 2^{-2n}, by Chernoff-Hoeffding bounds (when $\alpha > 0$ is a sufficiently large constant). To compute the average degree in F_r, one needs to divide the number of endpoints by f_r, and hence, with probability at least $1 - 2^{-2n}$, there is some vertex in F_r of degree at most $(n_r - 1)/2 + \alpha\sqrt{n}\sqrt{n_r/f_r} \leq n_r/2 + \alpha\sqrt{(c+1)n}$ (we use here the fact that $f_r \geq \sqrt{n}$ and $k_r \leq c\sqrt{n}$). Taking a union bound over all permissible values of f_r and k_r, and then over all possible sets F_r and K_r (regardless of whether they actually appear in the algorithm), the above holds simultaneously for all such choices, with probability at least $1 - 2^{-n}$. Setting $c_1 \geq \alpha\sqrt{c+1}$, completes the proof of the lemma. ■

For the next lemma, we will use the following definitions. Consider an arbitrary vertex $v \in S_1 \cup S_2$. Let x_r be a random variable indicating whether in step r of the removal phase, the vertex removed was from F_r. Let y_j be the random variable indicating whether in the first step r at which $\sum_{i=1}^{r} x_i = j$, the vertex removed (that must be from F_r since $x_r = 1$ at this point) is a neighbor of v in G' (and hence in G). Let r_v denote the step in which v itself was removed (or the last step of the removal phase if v was not removed). Let $s_p(v) = \sum_{j=1}^{p} y_j - p/2$ denote the *surplus* of neighbors of v removed compared to non-neighbors up to the point at which p vertices from F were removed (defined only for values of p for which $p \leq \sum_{i=1}^{r_v} x_i$).

Lemma 2.8. *Let $c_2 > 0$ be a sufficiently large constant. Then, for every $v \in S_1 \cup S_2$,*

$$\Pr[\max_{p \leq \sum_{i=1}^{r_v} x_i} s_p(v) \geq c_2\sqrt{n}] \leq 1/100.$$

In other words, for every $v \in S_1 \cup S_2$, the surplus for v is unlikely to ever exceed $O(\sqrt{n})$.

Proof: Let $v \in S_1 \cup S_2$. Switch the order in which the random events that generate G are revealed to the algorithm. First, pick all edges of G' at random (i.e., with probability $1/2$) except those edges connected to v. Then, pick two disjoint sets of vertices S_1 and S_2 (of size $k/2$ each) independently at random, such that one of them will include v (with equal probability), and denote by i the index of the part which was chosen to include v. Complete S_1, S_2 into a biclique. Now, generate a *tentative run* of the removal phase, without yet knowing which vertices of F and S_i are neighbors of v, and not allowing v to be removed during the tentative run. For the tentative run, one will need to know whether v is a neighbor of vertices (of F and S_i) that are candidates for removal. Only at the point in which the need arises, we will decide at random whether the candidate vertex u is a neighbor of v or not. At that point we say that $\{u, v\}$ is *exposed* (either as being an edge or not). Hence, the tentative run proceeds as follows.

1. If $|V(G_r)| \leq 4k/5$, then for every vertex $u \neq v$ from G_r, if $\{u, v\}$ was not previously exposed, then expose it. After finishing this procedure, stop.

2. Else, pick the vertex $u \neq v$ from G_r that has lowest degree (counting edges to v only in cases that they were already exposed, and breaking ties in favor of the vertex of lower index). Now, in order to make the tentative run consistent with the true run, there are several cases to consider:

2.1. If $\{u, v\}$ was previously exposed (one such case is when $u \in S_{3-i}$, and then we know that $\{u, v\}$ is an edge), remove u.

2.2. If $\{u, v\}$ was not previously exposed, then expose it. If after that u remains of lowest degree (in particular, this must happen when $\{u, v\}$ is not an edge), remove u. Otherwise, go to step 1.

When the tentative run stops, all edges to v have been exposed. Hence, now one is in a position to perform a true run of the removal phase. If v is not removed in the true run, then the true run is identical to the tentative run (this is a consequence of using a fixed rule for breaking ties, in particular, the rule of breaking them in favor of the vertex of lower index). If v is removed in the true run, then the true run and the tentative run are identical up to the point in which v is removed. Beyond that point, we no longer care about either of these two runs (for the analysis of the surplus).

Now, consider the development of the surplus of v with respect to the tentative run, up to step r_v (up to that point, the tentative run and true run are identical, and the lemma is not concerned with the surplus after point r_v). Vertices in S_{3-i} do not contribute to the surplus (by definition). As for vertices $u \in F$, up to step r we exposed the status of $\{u, v\}$ for many candidates, but not all of them have been removed. Note, however, that each exposed $\{u, v\}$ has probability exactly $1/2$ of being an edge (independent of all other events) and all those exposed and not removed have $\{u, v\}$ as an edge. It follows that the process of building up surplus is stochastically dominated by a random sequence of ± 1. Moreover, the length of this sequence is at most n. By standard bounds (that follow from the Optional Stopping Theorem for Martingales), the probability that a random sequence ever builds up a surplus of $c_2\sqrt{n}$ (for sufficiently large $c_2 > 0$) is at most $1/100$. ■

Corollary 2.9. *With probability at least $9/10$, there are at most $k/10$ vertices from $S_1 \cup S_2$ that ever reach a surplus of $c_2\sqrt{n}$ during the removal phase of the algorithm.*

Proof: For $w \in S_1 \cup S_2$, let X_w be an indicator random variable for the event that w reaches a surplus of $c_2\sqrt{n}$, and let $X = \sum_{w \in S_1 \cup S_2} X_w$. By Lemma 2.8 and using Markov's inequality, we get that:

$$\Pr[X \geq k/10] \leq \frac{\mathbb{E}[X]}{k/10} \leq \frac{k/100}{k/10} = \frac{1}{10}. \blacksquare$$

We note that Corollary 2.9, with its simple proof that only uses Markov's inequality, is the reason why in Theorem 2.4 the probability is a fixed constant. Possibly, the statement of Corollary 2.9 can be strengthened.

Lemma 2.10. *Let $U \subseteq S_1 \cup S_2$ be an arbitrary set of vertices of size $4k/5$. Then, with high probability, for every vertex $v \in V$, the degree of v into U in G is at least $21k/40$ if and only if $v \in S_1 \cup S_2$.*

Proof: We denote $U_1 = S_1 \cap U$ and $U_2 = S_2 \cap U$. As $|U| = 4k/5$, we have that $3k/10 \leq |U_1|, |U_2| \leq k/2$. Let $s \in [3k/10, k/2]$ such that $|U_1| = s$, and so $|U_2| = 4k/5 - s$, and let $v \in V$. First, if $v \notin S_1 \cup S_2$, then, with high probability, its degree into U is at most $k/2 + c_0'\sqrt{k \log n}$, by Corollary 2.6 (as $U \subseteq S_1 \cup S_2$), which is (strictly) less than $21k/40$, for large enough n. Second, without loss of generality we can assume that $v \in S_1$ (due to symmetry between S_1 and S_2). Then, its degree into U is at least $|U_2| = 4k/5 - s$ plus its degree into U_1, which we denote by x. By Proposition 2.5, with high probability, the degree of v into S_1 is at least $k/4 - c_0\sqrt{k \log n}$. Thus, at least $k/4 - c_0\sqrt{k \log n} - |S_1 \backslash U_1| = s - k/4 - c_0\sqrt{k \log n}$ of v's neighbors from S_1 are in U_1, and so $x \geq s - k/4 - c_0\sqrt{k \log n}$. Therefore, the degree of v into U is at least $22k/40 - c_0\sqrt{k \log n} \gg 21k/40$, for large enough n. The lemma follows. ■

We can now prove Theorem 2.4.

Proof of Theorem 2.4: Follow the (probable) evolution of the removal phase until step $\hat{r}$ satisfying $f_{\hat{r}} = \sqrt{n}$ (as we will see, the removal phase cannot possibly end before $f_r = \sqrt{n}$, but with a small probability). Let us analyze the value of $k_{\hat{r}}$, by upper bounding the number of vertices from $S_1 \cup S_2$ that are not in $K_{\hat{r}}$. Recall that $k = c\sqrt{n}$, for some sufficiently large constant c (larger than all other constants appearing in the proof, to be chosen later). In our proof we shall assume that all events that happen with high probability actually happen (this assumption can be made, as we shall only consider a constant number of such events).

Let L be the set of vertices from $S_1 \cup S_2$ that in G' have degree not more than $n/2 - c_3\sqrt{n}$ (where $c_3 > 0$ is a sufficiently large constant). The expected number of such vertices is at most $k/100$ (e.g., by Chebyshev's inequality), and hence, with probability at least $9/10$, there are at most $k/10$ such vertices (by Markov's inequality). We do not assume that any of the vertices of L get to reach $K_{\hat{r}}$.

By Proposition 2.5, the degree of every vertex of $S_1 \cup S_2$ increased by at least $k/4 - c_0\sqrt{k \log n}$ when $S_1 \cup S_2$ was made into a biclique. For sufficiently large n we have that $k/4 \gg c_0\sqrt{k \log n}$, and for simplicity of notation (and with only negligible effect on the bounds) we shall assume that the increase in degree is $k/4$. Hence, all vertices of $(S_1 \cup S_2) \backslash L$ have degree at least $n/2 + (c/4 - c_3)\sqrt{n}$ in G. Let T be the set of vertices of $S_1 \cup S_2$ that "suffered" a surplus of $c_2\sqrt{n}$ or more in the removal phase. By Corollary 2.9, with probability at least $9/10$ we have that $|T| \leq k/10$. We do not assume that any of the vertices of T get to reach $K_{\hat{r}}$.

For all vertices of $(S_1 \cup S_2) \backslash (L \cup T)$, their initial degree in G was at least $n/2 + (c/4 - c_3)\sqrt{n}$, their surplus is at most $c_2\sqrt{n}$, and hence at every step r their degree in G_r is at least $n_r/2 + (c/4 - c_2 - c_3 - c/5)\sqrt{n}$ (where the term $(c/5)\sqrt{n} = k/5 = k/10 + k/10$ is an upper bound on the effect of L and T not reaching $K_{\hat{r}}$). If this degree is at least $n_r/2 + c_1\sqrt{n}$, then these vertices cannot be removed (at every step r in which $f_r \geq \sqrt{n}$, the vertex removed from G_r has degree at most $n_r/2 + c_1\sqrt{n}$, by Lemma 2.7). It follows that if $c/20 \geq c_1 + c_2 + c_3$, then $k_{\hat{r}} \geq 4k/5$. Recall that Lemma 2.7 required that $c_1 \geq \alpha\sqrt{c+1}$. Hence, if c is large enough so as to satisfy $c \geq 20(\alpha\sqrt{c+1} + c_2 + c_3)$, all required relations between the various parameters can be made to hold. As the removal phase stops only when $|V(G_r)| \leq 4k/5$, this shows that the it cannot possibly end before $f_r = \sqrt{n}$, but with a small probability, as was claimed above.

Let us now analyze what happens in the removal phase after step $\hat{r}$. At this point, all vertices of K_r have degree at least $21k/40 \gg k/2$ (follows from Lemma 2.10, taking $U = K_{\hat{r}}$ and $v \in U$). All vertices of F_r have degree at most $f_r + k/2 + c_0'\sqrt{k \log n}$ (by Corollary 2.6), which is smaller than $21k/40$ (when c is sufficiently large, using also the fact that $f_r \leq \sqrt{n}$). Hence, no vertex of $K_{\hat{r}}$ will be removed (from now on), and all vertices of $F_{\hat{r}}$ will be removed. This establishes that the removal phase ends with $4k/5$ vertices from $S_1 \cup S_2$ and no vertex from F.

As to the inclusion phase, the fact that $k_r = 4k/5$ together with Lemma 2.10 and the check at step 4.1, imply that no vertex of F will be included. On the other hand, all vertices of $(S_1 \cup S_2) \backslash K_r$ will be included, and hence the output of algorithm FindBiclique will be the hidden biclique $S_1 \cup S_2$.

The probability that the assumptions of the analysis fail to hold is (by a union bound) at most $1/10 + 1/10 + o(1) < 1/3$.

Running time of the algorithm:

We assume an adjacency matrix representation of the input graph G, and analyze the running time of the removal and inclusion phases. We observe that for each vertex of G, its degree can be determined in time $O(n)$ by scanning the corresponding row/column in the matrix. Hence, we can create a vector of all vertex degrees. Finding the minimum degree can then be done in time $O(n)$. When removing the vertex of minimum degree, updating the degrees of all other vertices takes time $O(n)$ (subtracting one from each neighbor of the removed vertex, and marking the removed vertex). Since less than n vertices are removed, the whole removal phase takes time $O(n^2)$. Regarding the inclusion phase, it is clear that the check in step 4.1

takes $O(n)$ time, and we perform it $O(n)$ times. Thus, the whole inclusion phase takes time $O(n^2)$. Overall, the running time of the algorithm is $O(n^2)$, which is linear in the size of the adjacency matrix of G.
This completes the proof of Theorem 2.4. ■

Corollary 2.11. *For a sufficiently large constant $c > 0$, there exists a randomized polynomial-time algorithm that finds, with probability at least $2/3$, the hidden biclique in a graph G drawn from $G(n/2, n/2, \frac{1}{2}, k/2, k/2)$ for $k = c\sqrt{n}$, where the probability is taken over the choice of the input graph G and the randomness of the algorithm.*

Proof: Follows from Theorem 2.1 and Theorem 2.4. ■

Remark: An algorithm for finding a hidden biclique of size $k \geq c\sqrt{n}$ in a random bipartite graph, that has the same performance as in Corollary 2.11, was presented by Ames and Vavasis [AV11].

3 Planting Two Cliques in a Random Graph

In this section we study a variation of the Hidden Clique problem, in which two cliques are planted in a random graph, instead of just one. Formally, the input graph $G = (V, E)$ is constructed as follows. First, a random graph G' is drawn from $G(n, \frac{1}{2})$. Then, two disjoint sets of vertices $K_1, K_2 \subseteq V$ of the same size $k = k(n) \gg 2\log_2 n$ are chosen independently at random, and each of them is made into a clique. We denote by $G(n, \frac{1}{2}, k+k)$ the distribution of G. The goal is to find at least one of the two hidden cliques in polynomial time (with high probability over the choice of G).

We start by pointing out that there exists a polynomial-time algorithm, which is based on spectral graph partitioning, that finds, with high probability, both hidden cliques in a graph G drawn from $G(n, \frac{1}{2}, k+k)$ for all $k \geq \Omega(\sqrt{n})$ [McS01], but no such algorithm is known for any $k = o(\sqrt{n})$. Moreover, the algorithm presented in [McS01]can find any number $t \geq 1$ of hidden cliques, each of varying size, as long as all of them are of size $\geq \Omega(\sqrt{n})$.

We present a simple polynomial-time reduction from the standard planted clique problem to the version with two planted cliques, showing that the latter problem is at least as hard as the former.

Proposition 3.1. *For all $k = k(n) \gg 2\log_2 n$, if there exists a randomized polynomial-time algorithm that finds, with high probability, at least one hidden clique in a graph drawn from*

$G(n, \frac{1}{2}, k+k)$, then there exists an algorithm that solves the Hidden Clique problem with the same $k = k(n)$.

Proof: As there exist algorithms that solve the Hidden Clique problem for all $k \geq \Omega(\sqrt{n})$, it remains to prove the proposition for all $k = o(\sqrt{n})$. Let A be a randomized polynomial-time algorithm that finds, with high probability, at least one hidden clique in a graph G drawn from $G(n, \frac{1}{2}, k+k)$ (we assume that if A fails to find a hidden clique, then it returns an empty set). We present an algorithm that solves the Hidden Clique problem for the same k, which we call algorithm FindHC.

Algorithm 4 FINDHC (H, k)

Input: A graph $G = (V, E)$ drawn from $G(n, \frac{1}{2}, k)$, with $2\log_2 n \ll k = k(n) = o(\sqrt{n})$.
Output: The hidden clique in G (w.h.p.).

1. Choose a random set of vertices $T \subseteq V$ of size k and make it into a clique, denoting the resulting graph by H.
2. Execute $Q_1, Q_2 \leftarrow A(H)$, and let $Q = (Q_1 \cup Q_2) \backslash T$.
3. If $(Q \neq \emptyset)$, **return** Q.
4. **return** "FAIL".

Analysis:

Denote by S the hidden clique in G. Clearly, if the algorithm succeeds to choose a set T (in step 1) such that $T \cap S = \emptyset$, then the graph H is distributed according to $G(n, \frac{1}{2}, k+k)$. The probability to fail in choosing such a T is, by a union bound:

$$\Pr[T \cap S \neq \emptyset] \leq \sum_{i=0}^{k-1} \frac{k}{n-i} \leq \frac{k^2}{n-k} = o(1).$$

We now compute the probability that algorithm A will return the true hidden clique S when executed on the graph H (assuming that $T \cap S = \emptyset$). We look upon the generation of the graph H as the following process: First, a random graph H' is drawn from $G(n, \frac{1}{2})$. Then, two disjoint sets of vertices $K_1, K_2 \subseteq V(H')$ of the same size k are chosen independently at random, and each of them is made into a clique. We observe that for a given graph H as above, with two planted cliques K_1, K_2, the probability that K_1 is the true planted clique S is exactly $1/2$ of the probability for H to be drawn (by symmetry). As algorithm A succeeds in recovering at least one of the hidden cliques with probability $1 - o(1)$, it will return the true hidden clique with probability at least $1/2 - o(1)$, over the choice of G and its randomness.

Clearly, the running time of the algorithm is polynomial in n. ■

Remark: We note that if a constant number $r > 2$ cliques are planted, then the same reduction works (with straightforward modifications), and our algorithm will succeed to recover the hidden clique with probability at least $1/r - o(1)$.

4 The Hidden Clique Problem in a Communication Complexity Setting

In this section we analyze the communication complexity of the Hidden Clique problem in two models, which are defined in Section 1.1.3. The motivation for studying such protocols is to examine the following question: what is the minimum amount of information Bob needs about the underlying random graph of G, or about G itself but with a random permutation of the vertices, so that he could recover the hidden clique in G in polynomial time? In addition, we would like to compare between the two models, by analyzing and comparing the communication complexity of protocols in each of them. This may suggest which input for Alice contains more helpful information for recovering the hidden clique. Moreover, we would like to compare different protocols in the same model.

4.1 A Protocol Based on Sums of Degrees

In this section we present a protocol in the first model, that is based on sums of degrees and has a polylogarithmic communication complexity. We start by constructing a protocol that has communication complexity $O(k \cdot \text{polylog}(n))$ (where k is the size of the hidden clique), and then refine it to have a polylogarithmic communication complexity. The first protocol is defined as follows.

Protocol FindClique1

Alice's Input: A graph $G' = (V, E')$ drawn from $G(n, \frac{1}{2})$; $k = k(n) \gg 2\log_2 n$.
Bob's Input: A graph $G = (V, E)$, constructed by choosing, uniformly at random, a set K of $k = k(n) \gg 2\log_2 n$ vertices in G', and making them into a clique; $k = k(n) \gg 2\log_2 n$.
Shared/public randomness: A random string r of polynomial (in n) size.

1. Alice initializes an empty string T.

2. Then, Alice repeats the following $m = 10k \log_2 n$ times (independently):

   2.1. She samples a set $S \subseteq V$ by including each $v \in V$ independently with probability $1/k$.

   2.2. She then computes the sum of degrees of the vertices of S in the graph G', and concatenates it to T.

3. Alice sends the string T to Bob.

4. As Alice and Bob have a shared randomness, Bob knows exactly which sets of vertices Alice had sampled (and their order). For each such set $S \subseteq V$, Bob does the following:

   4.1. He computes the sum of degrees of the vertices of S in the graph G.

   4.2. Then, he compares it to the corresponding value in T. If the two values are equal, he marks all the vertices of S as not belonging to the hidden clique.

5. Bob counts the number of vertices that were not marked. If their number is exactly k, he **returns** them; Otherwise, he **returns** "FAIL".

Theorem 4.1. *With high probability, the output of Bob in the protocol FindClique1 will be the vertices of the hidden clique in the graph G, and the protocol uses $O(k \log^2 n)$ bits of communication.*

We start by proving the following lemmas.

Lemma 4.2. *Let $v \in V$ be some fixed vertex. Then, the probability that v was not sampled (by Alice) into S in all iterations is $\leq n^{-10}$.*

Proof: Let $v \in V$ be some fixed vertex. In each iteration, we have that

$$\Pr[v \text{ was not sampled to } S] = 1 - 1/k.$$

Thus, due to independence between the iterations, we have that

$$\Pr[v \text{ was not sampled in all iterations}] = (1 - 1/k)^{10k \log_2 n} \leq e^{-10 \log_2 n} \leq n^{-10}. \blacksquare$$

Lemma 4.3. *Let $v \in V$ be some fixed vertex, and assume that $v \notin K$. Then, the probability that v was sampled (by Alice) only into sets S, where some vertex from K was also sampled is $\leq n^{-\Omega(k)}$.*

Proof: Let $v \in V$ be some fixed vertex, and assume that $v \notin K$. In each iteration, as S includes each vertex independently, we have that

$$\Pr[S \subseteq V \backslash K \mid v \in S \wedge v \notin K] = \Pr[S \subseteq V \backslash K] = (1 - 1/k)^k \geq 1/4. \tag{2}$$

Thus, due to independence between the iterations, we have that the probability that $S \nsubseteq V \backslash K$ in all iterations is at most

$$(1 - 1/4)^{10k \log_2 n} \leq e^{-k \log_2 n} \leq n^{-k}. \tag{3}$$

■

We can now prove Theorem 4.1.

Proof: By Lemma 4.2 and Lemma 4.3, we get that the probability that some fixed vertex $v \in V$ (assuming that $v \notin K$) was sampled into a set S, to which no vertex from K was sampled is at least $1 - 2n^{-10}$. Thus, applying a union bound over all the vertices in V, we get that with high probability, every $v \in V \backslash K$ will be a sampled into a set S such that the sums of degrees of its vertices in G' and in G are the same (as only the degrees of the vertices in K change). Therefore, Bob will eventually mark all the vertices which are not in K, and thus the vertices of the hidden clique will be returned. Clearly, Bob runs in polynomial time. The communication complexity is the length of the string T, which is $O(k \log^2 n)$ bits. ■

Remark: We note that protocol FindClique1 can be extended to a simultaneous protocol (with the same communication complexity), where the referee does not have an access to the inputs of the parties but does have an access to the shared randomness, as follows. Alice will send to the referee the same message she sends to Bob in the original protocol. And, Bob will send to the referee a message consisting of the values he computed in step 4.1 (for every set S). Clearly, the referee has now all the required information to perform the computation that Bob does in steps 4.2 and 5. As the message that Bob sends to the referee is of size $O(k \log^2 n)$, we have that the communication complexity is $O(k \log^2 n)$.

We now refine protocol FindClique1, in order to reduce the communication complexity, as follows.

Protocol FindClique2

Alice's Input: A graph $G' = (V, E')$ drawn from $G(n, \frac{1}{2})$; $k = k(n) \gg 2\log_2 n$.
Bob's Input: A graph $G = (V, E)$, constructed by choosing, uniformly at random, a set K of $k = k(n) \gg 2\log_2 n$ vertices in G', and making them into a clique; $k = k(n) \gg 2\log_2 n$.
Shared/public randomness: A random string r of polynomial (in n) size.

1. Alice samples a set $A \subseteq V$ by including each $v \in V$ independently with probability $\frac{10\log_2 n}{k}$.

2. Set $k' = 10\log_2 n$ and $m' = 10k'\log_2 n = 100\log_2^2 n$.

3. Execute protocol FindClique1 on the induced subgraphs $G'[A]$ (Alice) and $G[A]$ (Bob), using the values k', m' instead of k, m (both parties know the same set A as they have a shared randomness).

4. If the execution failed, **return** "FAIL"; Otherwise, denote by Q the set of vertices that Bob got from the execution of protocol FindClique1.

5. Bob computes the set of all common neighbors of the vertices in Q (in the graph G), denoted by T, and **returns** $Q \cup T$.

Theorem 4.4. *With high probability, the output of Bob in the protocol FindClique2 will be the vertices of the hidden clique in the graph G, and the protocol uses $O(\log^3 n)$ bits of communication.*

We start by proving the following lemma.

Lemma 4.5. *Let G* be a graph drawn from $G(n, \frac{1}{2}, k)$ *for $k = k(n) \gg 2\log_2 n$, let K be the set of vertices of the hidden clique in G, and let C be a random subset of K of size $3\log_2 n$. Then, with high probability, the common neighbors of C are exactly the remaining vertices of K.*

Proof: We observe that over the choice of $G \leftarrow G(n, \frac{1}{2}, k)$, a random subset C of the hidden clique is basically a random subset of $V(G)$. Let C be a random subset of the vertices of G of size $3\log_2 n$ and let $v \in V \backslash C$. The probability that v is a common neighbor of all the vertices in C in the underlying random graph of G is $2^{-3\log_2 n} = n^{-3}$. Taking a union bound over all the vertices, we get that the probability that all the vertices in C have a common neighbor (in the underlying random graph) is $o(1)$. Therefore, with high probability, all common neighbors of a random subset of size $3\log_2 n$ of the hidden clique are exactly the rest of the vertices of the hidden clique. ■

We can now prove Theorem 4.4.

Proof: We observe that the induced subgraph $G'[A]$ in step 3 is a random graph on roughly $n' = \frac{10n\log_2 n}{k}$ vertices (by Chernoff-Hoeffding bounds). We further observe that the induced subgraph $G[A]$ in step 3 is the random graph $G'[A]$, with a (randomly) planted clique of size roughly $k' = 10\log_2 n$ (by Chernoff-Hoeffding bounds). Moreover, clearly the planted clique in $G[A]$ is a random subset of K (of size roughly $k' = 10\log_2 n$). Due to the fact that the size of the planted clique K' in $G[A]$ is $\hat{k} = ck' = \Theta(k')$ and not exactly k' (for some constant $c > 0$), we need to show that protocol FindClique1 will still recover the hidden clique, with high probability, when executed on $G'[A], G[A]$ (with k', m'). It is easy to see that the proof of Lemma 4.2 is still valid, as it has nothing to do with the size of the planted clique. As for Lemma 4.3, we change equation (2), as follows:

$$\Pr[S \subseteq V(G[A])\backslash K' \mid v \in S \wedge v \notin K'] = \Pr[S \subseteq V(G[A])\backslash K'] = (1-1/k')^{\hat{k}} = (1-1/k')^{ck'} \geq (1/4)^c.$$

This results in the following changes in equation (3):

$$(1-1/4^c)^{10k'\log_2 n} \leq e^{-(10/4^c)k'\log_2 n} \leq n^{-\Omega(k')} = n^{-\Omega(\hat{k})} = n^{\Omega(\log n)}.$$

Thus, this lemma also holds, and so does Theorem 4.1. Therefore, the execution of protocol FindClique1 will still recover the hidden clique (in $G[A]$), with high probability. In addition, by Lemma 4.5, Bob succeeds to recover the whole hidden clique in step 5, with high probability. Clearly, Bob runs in polynomial time. The communication complexity is $O(k'\log^2 n) = O(\log^3 n)$ bits (due to the execution of protocol FindClique1). ■

4.2 A Protocol Based on ℓ_0-samplers

In this section we present a protocol in the first model, that is based on ℓ_0-samplers and has a polylogarithmic communication complexity.

We start by defining ℓ_0-sampling, as follows. The *support* of a vector $\mathbf{x} \in \mathbb{R}^n$, $\mathrm{supp}(\mathbf{x})$, is defined by $\mathrm{supp}(\mathbf{x}) \stackrel{\text{def}}{=} \{i \in [n] \mid x_i \neq 0\}$. Consider an input vector $\mathbf{x} \in \mathbb{R}^n$, given in a streaming fashion, where at each step an update (increment or decrement) to coordinate x_i is given. The goal is to sample a coordinate from $\mathrm{supp}(\mathbf{x})$, uniformly at random. We refer to such an algorithm as an ℓ_0-sampler.

Theorem 4.6. (ℓ_0-sampler [[CF14], Corollary 1]). *There is an ℓ_0-sampler that uses $O(\log^4 n)$ bits, succeeds with probability at least $1 - n^{-c}$, and, conditioned on a successful recovery,*

outputs a coordinate i with non-zero x_i with probability $\frac{1}{|\mathrm{supp}(\mathbf{x})|} \pm n^{-c}$ for arbitrarily large constant c.

We note that the construction of an ℓ_0-sampler that is presented in [CF14] is a linear sketch (i.e., for two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ we have that $sk(\mathbf{x}+\mathbf{y}) = sk(\mathbf{x}) + sk(\mathbf{y})$, using the same random coins).

We now describe the protocol.

Protocol FindClique3

Alice's Input: A graph $G' = (V, E')$ drawn from $G(n, \frac{1}{2})$ (we assume that $V = [n]$).
Bob's Input: A graph $G = (V, E)$, constructed by choosing, uniformly at random, a set K of $k = k(n) \gg \log_2^2 n$ vertices in G', and making them into a clique.
Shared/public randomness: A random string r of polynomial (in n) size.

1. Alice initializes $m = 10 \log_2 n$ independent copies of an ℓ_0-sampler.
2. Let $\mathbf{x} \in \mathbb{R}^n$ be the degree vector of the graph G'. Alice feeds each of the (independent) ℓ_0-samplers with the vector $\mathbf{x}$ as incremental input.
3. Alice sends to Bob the "memory state" of all the ℓ_0-samplers.
4. Let $\mathbf{y} \in \mathbb{R}^n$ be the degree vector of the graph G. Bobs feeds each of the ℓ_0-samplers he received with the vector $-\mathbf{y}$ as incremental input.
5. Bob samples a coordinate from each of the ℓ_0-samplers, and gets a set of vertices, denoted by Q.
6. If all the $m = 10 \log_2 n$ vertices in Q are distinct, then Bob computes the set of all common neighbors of the vertices in Q, denoted by T, and **returns** $Q \cup T$; Otherwise, he **returns** "FAIL".

Theorem 4.7. *With high probability, the output of Bob in the protocol FindClique3 will be the vertices of the hidden clique in the graph G, and the protocol uses $O(\log^5 n)$ bits of communication.*

Proof: We observe that the support of the vector $\mathbf{x} - \mathbf{y}$ consists exactly of the vertices of the hidden clique (with high probability over the choice of G), as they are the only ones whose degree is not the same in the graphs G' and G. We also observe that Bob can

continue feeding the ℓ_0-samplers in step 4, as they are linear sketches. Due to the fact that $k \gg \log_2^2 n$, the probability that two of the $m = O(\log n)$ ℓ_0-samplers will return the same vertex is $o(1)$, by the Birthday Paradox (as all the ℓ_0-samplers are mutually independent, and output a uniformly random vertex from the hidden clique (by Theorem 4.6)). In addition, the probability that a given ℓ_0-sampler fails is $\leq 1/n$ (by Theorem 4.6), and thus, by a union bound, the probability that at least one of them fails is $o(1)$. Since Q is a random subset of K (except for the events just mentioned), by Lemma 4.5, Bob succeeds to recover the whole hidden in step 6, with high probability. Therefore, the total probability of failure is $o(1)$. Clearly, Bob runs in polynomial time. The communication complexity is basically the storage required for the ℓ_0-samplers, which is $m \cdot O(\log^4 n) = O(\log^5 n)$ bits (by Theorem 4.6). ■

4.3 A Protocol Based on the Degree Sequences of Neighbors

In this section we present a protocol in the second model, that is based on the degree sequences of neighbors of vertices, and has communication complexity $O(\log^2 n)$ bits. The protocol is defined as follows.

Protocol FindClique4

Alice's Input: A graph $G = (V, E)$ drawn from $G(n, \frac{1}{2}, k)$, with $2\log_2 n \ll k = k(n) = o(\sqrt{n})$.
Bob's Input: A graph $H = (V, E')$, obtained from G by randomly permuting the labels of the vertices.
Shared/public randomness: A random string r of polynomial (in n) size.

1. Alice finds the vertices of the hidden clique in G, denoted by K, by exhaustive search.
2. Alice picks, uniformly at random, a hash function $h : [2^{n \log_2 n}] \to [n^3]$ from a *universal* hash functions family $\mathcal{H}$.
3. Alice picks a random set S of $m = 10 \log_2 n$ vertices from K.
4. For each vertex $v \in S$, Alice performs the following steps:
   4.1. She initializes (with the value 0) an array of size n, denoted by D_v.
   4.2. She then computes the degrees of all the neighbors of v, sorts them in ascending order, and fills in D_v with the values of the degrees (from smallest to largest).

5. Alice computes the hash values $\{h_v = h(D_v)\}_{v \in S}$ (note that the length of each D_v is $n \log_2 n$ bits), and sends them to Bob.

6. For each vertex $u \in V$, Bob performs the following steps:

   6.1. He initializes (with the value 0) an array of size n, denoted by D'_u.

   6.2. He then computes the degrees of all the neighbors of u, sorts them in ascending order, and fills in D'_u with the values of the degrees (from smallest to largest).

7. Bob computes the hash values $\{h'_u = h(D'_u)\}_{u \in V}$ (for the same h, using shared randomness).

8. If all the hash values sent by Alice are distinct and appear exactly once among those that Bob computed, then Bob denotes by Q the set of vertices that correspond to those hash values, computes the set of all their common neighbors, denoted by T, and **returns** $Q \cup T$; Otherwise, he **returns** "FAIL".

Theorem 4.8. *With high probability, the output of Bob in the protocol FindClique4 will be the vertices of the hidden clique in the graph H, and the protocol uses $O(\log^2 n)$ bits of communication.*

We start by proving the following lemmas.

Lemma 4.9. *Let $c > 0$ be some constant, and let $X \sim \mathrm{Bin}(cn, \frac{1}{2})$. Then, $\Pr[X = k] \leq O(\frac{1}{\sqrt{n}})$ for all k.*

Proof: Let k be some value. We have that $\Pr[X = k] = \binom{cn}{k} 2^{-cn} \leq \binom{cn}{cn/2} 2^{-cn} \underset{\text{Stirling}}{\leq} O(\frac{2^{cn}}{\sqrt{n}}) \cdot 2^{-cn} = O(\frac{1}{\sqrt{n}})$. ■

Lemma 4.10. *Let $c > 0$ be some constant, let $X \sim \mathrm{Bin}(cn, \frac{1}{2})$, and Z be some discrete random variable, independent of X. Then, $\Pr[X = Z + k] \leq O(\frac{1}{\sqrt{n}})$ for all k.*

Proof: By the law of total probability,

$$\begin{aligned}
\Pr[X = Z + k] &= \sum_z \Pr[Z = z] \Pr[X = Z + k \mid Z = z] \\
&\underset{\text{indep.}}{=} \sum_z \Pr[Z = z] \Pr[X = z + k] \\
&\underset{\text{Lemma 4.9}}{\leq} O(\frac{1}{\sqrt{n}}) \sum_z \Pr[Z = z] \\
&= O(\frac{1}{\sqrt{n}}).
\end{aligned}$$

■

Lemma 4.11. *Let $G = (V, E)$ be a graph drawn from $G(n, \frac{1}{2}, k)$ for $2\log_2 n \ll k = k(n) = o(\sqrt{n})$. For every $v \in V$, denote by D_v the list of degrees of its neighbors, sorted in ascending order (as in the protocol). Then, the probability that there exist $u \neq v \in V$ such that $D_u = D_v$ is $o(1)$.*

Proof: Let $u, v \in V$ be some fixed vertices, and let K denote the set of vertices of the planted clique in G. We start by exposing the edges inside K, the neighbors of u and the neighbors of v. By a standard combination of Chernoff-Hoeffding bounds and union bound, we have that with high probability, for all $a \neq b \in V$, the number of neighbors of a which are not neighbors of b is in the range $[n/5, n/3]$. Let $u_1, \ldots, u_r$ be the neighbors of u which are not neighbors of v, and let $v_1, \ldots, v_r$ be the neighbors of v which are not neighbors of u (we can condition on their amount being the same, as otherwise clearly $D_u \neq D_v$). Let $\overline{Q} = K \cup \{u_i, v_i\}_{i\in[r]} \cup \{u, v\}$ and $Q = V \backslash \overline{Q}$. For all $i \in [r]$, we can express the degree of u_i (and v_i) in G as its degree into Q plus its degree into $\overline{Q}$. Let the random variable U_i denote the degree of u_i into Q, and let the random variable $\overline{U_i}$ denote its degree into $\overline{Q}$. Similarly, let the random variable V_i denote the degree of v_i into Q, and let the random variable $\overline{V_i}$ denote its degree into $\overline{Q}$. We denote the degree of vertex u_i by $U_i^* = U_i + \overline{U_i}$, and the degree of vertex v_i by $V_i^* = V_i + \overline{V_i}$. We have that for all $i \neq j \in [r]$, U_i and $\overline{U_j}$ are independent, and same goes for V_i and $\overline{V_j}$. This is because all the vertices of the hidden clique are in $\overline{Q}$ (and so are u_i, u_j), and hence the edges to Q are "still" random (i.e., occur independently with probability $\frac{1}{2}$). As $r \leq n/3$, we have that $|\overline{Q}| \leq o(\sqrt{n}) + 2n/3 + 2 < 5n/6$, and thus $|Q| > n/6$. Therefore, $U_i, V_i \sim \text{Bin}(cn, \frac{1}{2})$ for some constant $c > 0$, and they are independent of each other (even after exposing some edges as described above). We want to estimate the probability that the multisets $\{U_i^*\}_i$ and $\{V_j^*\}_j$ are equal. We now expose only $U_1^*, ..., U_{r-5}^*$ and $V_1^*, \ldots, V_r^*$. Clearly, if $\{U_1^*, ..., U_{r-5}^*\} \nsubseteq \{V_1^*, \ldots, V_r^*\}$, then $\Pr[\{U_1^*, ..., U_r^*\} = \{V_1^*, \ldots, V_r^*\}] = 0$. However, if $\{U_1^*, ..., U_{r-5}^*\} \subseteq \{V_1^*, \ldots, V_r^*\}$, then there is still a chance that the multisets are equal. In this case, we basically have at most 5 values among the V^*'s, denoted by $v_1^*, \ldots, v_5^*$, that appear in $\{V_1^*, \ldots, V_r^*\} \backslash \{U_1^*, ..., U_{r-5}^*\}$, and so when we will expose $\{U_{r-4}^*, ..., U_r^*\}$, there is a chance that their values will be equal to the ones we need in order to have an equality. We point out that $v_1^*, \ldots, v_5^*$ are random variables, which are independent of $\{U_{r-4}, \ldots, U_r\}$. By the law of total probability, we have that:

$$\begin{aligned}
\Pr[\{U_i^*\}_i = \{V_j^*\}_j] &= \Pr[\{U_i^*\}_i = \{V_j^*\}_j \mid \{U_i^*\}_{i\in[r-5]} \nsubseteq \{V_j^*\}_{j\in[r]}]\Pr[\{U_i^*\}_{i\in[r-5]} \nsubseteq \{V_j^*\}_{j\in[r]}] \\
&\quad + \Pr[\{U_i^*\}_i = \{V_j^*\}_j \mid \{U_i^*\}_{i\in[r-5]} \subseteq \{V_j^*\}_{j\in[r]}]\Pr[\{U_i^*\}_{i\in[r-5]} \subseteq \{V_j^*\}_{j\in[r]}] \\
&\leq \Pr[\{U_i^*\}_i = \{V_j^*\}_j \mid \{U_i^*\}_{i\in[r-5]} \subseteq \{V_j^*\}_{j\in[r]}] \\
&= \Pr[\{U_{r-4}^*, ..., U_r^*\} = \{v_1^*, \ldots, v_5^*\} \mid v_1^*, \ldots, v_5^*].
\end{aligned}$$

Denote $Z = \{v_1^*, \ldots, v_5^*\} \cup \{\overline{U_{r-4}}, \ldots, \overline{U_r}\}$. We observe that conditioning on Z, the random variables $U_{r-4}, \ldots, U_r$ are still mutually independent. In addition, we have that for all $r-4 \leq i \leq r$ and for all $j \in [5]$,

$$\Pr[U_i^* = v_j^* \mid v_1^*, \ldots, v_5^*] = \Pr[U_i = v_j^* - \overline{U_i} \mid Z] \underset{\text{Lemma 4.10}}{\leq} O(\frac{1}{\sqrt{n}}). \tag{4}$$

Thus, continuing the calculation from above, we get that:

$$\begin{aligned}
\Pr[\{U_{r-4}^*, ..., U_r^*\} = \{v_1^*, \ldots, v_5^*\} \mid v_1^*, \ldots, v_5^*] &= \sum_{\sigma\in S_5} \Pr[\wedge_{i\in[5]}(U_{r-i+1} = v_{\sigma(i)}^* - \overline{U_{r-i+1}}) \mid Z] \\
&\underset{\text{indep.}}{=} \sum_{\sigma\in S_5}\prod_{i\in[5]} \Pr[U_{r-i+1} = v_{\sigma(i)}^* - \overline{U_{r-i+1}} \mid Z] \\
&\underset{(4)}{\leq} 5! \cdot O(n^{-2.5}) \\
&= O(n^{-2.5}).
\end{aligned}$$

Therefore, we conclude that $\Pr[D_u = D_v] \leq O(n^{-2.5})$. Finally, taking a union bound over all pairs of vertices $u, v \in V$, we get that $\Pr[\exists u \neq v.(D_u = D_v)] \leq O(\frac{1}{\sqrt{n}}) = o(1)$. ■

We can now prove Theorem 4.8.

Proof: We start by noting that initializing the arrays with zeros is plausible, as the probability that some vertex would have degree 0 is $o(1)$. Now, by Lemma 4.11, the arrays of all the vertices are distinct. In addition, as the hash function h is chosen from a universal hash functions family, we have that for all $D_u \neq D_v$, $\Pr[h(D_u) = h(D_v)] \leq 1/n^3$. Taking a union bound over all pairs of vertices $u, v \in V$, we get that $\Pr[\forall u, v.(h(D_u) = h(D_v))] \leq 1/n = o(1)$. Therefore, with high probability, in step 8 Bob will acquire the new labels of $m = 10\log_2 n$ random vertices from the hidden clique, and, by Lemma 4.5, he will succeed to recover from them the whole hidden clique (in step 8), with high probability. Therefore, the total probability of failure is $o(1)$. Clearly, Bob runs in polynomial time. The communication complexity is basically the $O(\log n)$ hash values that Alice sends, and thus is $O(\log^2 n)$ bits overall. ■

4.4 A Lower Bound on the Communication Complexity

In this section we show a lower bound on the communication complexity of the Hidden Clique problem, assuming that there is no algorithm that solves the Hidden Clique problem for $k = o(\sqrt{n})$. The lower bound holds for any protocol where Bob's only input is a graph G drawn from $G(n, \frac{1}{2}, k)$ for $k = k(n) \gg 2\log_2 n$, regardless of the input of Alice. Formally, we prove the following proposition.

Proposition 4.12. *Let $k = k(n) \gg 2\log_2 n$. Assume that no algorithm can solve the Hidden Clique problem for $k = k(n)$. Then, every protocol for the Hidden Clique problem where Bob's only input is a graph H drawn from $G(n, \frac{1}{2}, k)$, requires $\omega(\log n)$ bits of communication.*

Proof: Assume to the contrary that there is a protocol P for the Hidden Clique problem as stated above, that requires $O(\log n)$ bits of communication, which we denote by S. We define the following algorithm B' for finding, with high probability, the hidden clique in a graph G drawn from $G(n, \frac{1}{2}, k)$. On input $G \leftarrow G(n, \frac{1}{2}, k)$, B' will choose a random string r of polynomial (in n) size, and then will simulate the algorithm executed by Bob in the protocol P on every possible value of the string S (communicated by Alice in P), with G being Bob's input and r being the (shared) randomness of the protocol. It will return the vertices of the largest clique found by Bob in these simulations (we can verify/check efficiently whether a set of vertices is a clique). If no clique was found in all simulations, it will return "FAIL".

We now prove the correctness of algorithm B'. Due to the fact that algorithm B' executes Bob's algorithm (in P) on every possible value of S, one of them will be identical to the bits communicated by Alice in some execution of the protocol, and thus we will have a realization of Bob's computation in the protocol. Therefore, the success probability of B' in recovering the hidden clique is the same as of Bob in the protocol, which is $1 - o(1)$ (when the case that the planted clique will not be the maximum clique in G, which happens with probability $o(1)$, is included in Bob's failure probability). In addition, clearly B' runs in polynomial time, as by the assumption, we have that $|S| \leq c \log n$ for some constant $c > 0$, and thus the number of possible strings S is $O(n^c)$. Combining it with the fact that Bob's running time (in the protocol) is poly(n), we get that B' is indeed a polynomial-time algorithm. This is a contradiction to the assumption, and thus completes the proof. ■

Remark: If we make a stronger hardness assumption that solving the Hidden Clique problem requires $n^{\Omega(\log n)}$ time, then we derive, by the same proof, a stronger lower bound on the communication complexity, which is $\Omega(\log^2 n)$ bits of communication (as the number of

possible strings S of size $c\log^2 n$ is $n^{c\log n}$). The motivation for making such stronger hardness assumption is that the best algorithm known for the Hidden Clique problem with $k = o(\sqrt{n})$ runs in time $n^{O(\log n)}$ (see Section 1 of [FR10]).

5 Finding a Hidden Clique Using Statistical Queries

In this section we discuss algorithms for the Hidden Clique problem, which are allowed to perform statistical queries either on the underlying random graph (i.e., before the clique was planted) or on the input graph itself. More specifically, the algorithm can make a query on any subset of the vertices, to obtain some aggregate statistic of this subset. The goal is to find the hidden clique using few such queries and in polynomial time. We shall consider only statistics that are not known to be computed in polynomial-time (or even information-theoretically), as otherwise the algorithm could have computed the required information by itself in polynomial time (without any queries).

We study the following type of statistical queries. For an input graph $G = (V, E)$ drawn from $G(n, \frac{1}{2}, k)$ for $k = k(n) \gg 2\log_2 n$, the algorithm can query for the sum of degrees of the vertices in any subset $S \subseteq V$, in the underlying random graph G' (i.e., before the clique was planted). Observe that n queries suffice, as one can query all the vertex degrees in G', and then identify the vertices of the planted clique as the ones whose degree has increased.

We first show a lower bound on the query complexity, assuming that there is no algorithm that solves the Hidden Clique problem for $k = o(\sqrt{n})$, and then present an algorithm (with queries as described above) for finding the hidden clique.

Proposition 5.1. *Let $k = k(n) \gg 2\log_2 n$. Assume that no algorithm solves the Hidden Clique problem for $k = k(n)$. Then, every algorithm that can make queries as described above, and solves the Hidden Clique problem for all $k = k(n)$, has query complexity $\omega(\frac{\log n}{\log k})$.*

Proof: Let $k = k(n) \gg 2\log_2 n$. Assume to the contrary that there is an algorithm, denoted by B, with query access as above, that solves the Hidden Clique problem for $k = k(n)$, and has query complexity $t = O(\frac{\log n}{\log k})$. We define the following algorithm A, that solves the Hidden Clique problem for $k = k(n)$. On input $G \leftarrow G(n, \frac{1}{2}, k)$, algorithm A will simulate algorithm B on the input G, step by step. Once algorithm B has to make a query on some subset $S \subseteq V(G)$, it will instead compute the sum of degrees (in G, not in G') of the vertices in S, denoting it by x_S, and then iterate over all the values $x - j$ for $j \in \{0, 1, \ldots, k^2\}$, trying each of them as an answer to that query (note that this can be done in parallel). Each time it finishes an execution of algorithm B (on some series of values as the answers to the queries),

it will check whether the output of B is a k-clique. If in any of the iterations it obtains a k-clique, then it will return its vertices. Otherwise, it returns "FAIL".

We now prove the correctness of algorithm A. For a subset $S \subseteq V(G)$, denote by x_S the sum of degrees of its vertices in G. It holds that the sum of degrees of the vertices of S in the underlying random graph G' is in the range $\{x_S - k^2, \ldots, x_S\}$, as only the degrees of the k vertices of the hidden clique increase (and at most by k each). Thus, an answer to every query made by algorithm B is in that range. Therefore, as algorithm A tries every possible answer for every query, one of these trials will have the correct answers for all the queries, and so will thus realize algorithm B. Therefore, it will find the hidden clique in G, with high probability. Denoting the running time of algorithm B by $T(n) = \text{poly}(n)$, we get that the running time of algorithm A is $T(n) \cdot O((k^2)^t) = T(n) \cdot O(k^{O(t)}) = \text{poly}(n)$. This contradicts the assumption, and thus completes the proof. ■

Remark: We point out that the smaller k is, the better lower bound is achieved. In addition, if we make a stronger hardness assumption that solving the Hidden Clique problem requires $n^{\Omega(\log n)}$ time, then we derive, by the same proof, a stronger lower bound of query complexity $\Omega(\frac{\log^2 n}{\log k})$, simply because the running time of algorithm A is $\text{poly}(n) \cdot O(k^{O(t)})$. The motivation for making such stronger hardness assumption is that the best algorithm known for the Hidden Clique problem with $k = o(\sqrt{n})$ runs in time $n^{O(\log n)}$ (see Section 1 of [FR10]).

Theorem 5.2. *There exists an algorithm that makes $O(\log^2 n)$ queries as described above, and solves the Hidden Clique problem for all $k = k(n) \gg 2\log_2 n$.*

Proof: We start with the description of an algorithm, as follows.

Algorithm 5 FindCliqueQueries (G, k)

Input: A graph $G = (V, E)$ drawn from $G(n, \frac{1}{2}, k)$, with $k = k(n) \gg 2\log_2 n$.
Output: The hidden clique in G (w.h.p.).

1. Initialize $T = \emptyset$, $i = 0$, $G_0 = G$.
2. **while** $(|T| < 10\log_2 n)$ **do**
   - 2.1. Sample a set $S \subseteq V[G_i]$ by including each $v \in V[G_i]$ independently with probability $\frac{\log_2 n}{k}$.
   - 2.2. Compute the sum of degrees of the vertices of S in the graph G, and denote it by x_S.
   - 2.3. Make a query on S and get the result y_S.
   - 2.4. If $y_S = x_S$, **return** "FAIL";
   - 2.5. Else, repeat the following steps:
     - 2.5.1. If $|S| = 1$, then update $T = T \cup S$, $i = i+1$, $G_i = G_{i-1} \backslash S$, and go to step 2.1.
     - 2.5.2. Partition S into two sets of equal size S_1, S_2, at random.
     - 2.5.3. Compute the sum of degrees of the vertices of S_1 and S_2 in the graph G, and denote them by w_{S_1} and w_{S_2}, respectively.
     - 2.5.4. Make queries on S_1 and S_2, and get the results z_{S_1}, z_{S_2}, respectively.
     - 2.5.5. If ($w_{S_1} > z_{S_1}$ and $w_{S_2} > z_{S_2}$), then choose $j \in \{1, 2\}$ at random and set $S = S_j$; Otherwise, if $w_{S_1} > z_{S_2}$, then set $S = S_1$; Else, set $S = S_2$.
     - 2.5.6. Go to step 2.5.1.
3. Compute the set of all common neighbors of the vertices in T, and denote it by Q.
4. **return** $T \cup Q$.

Analysis:

We observe that the number of vertices in the graph G_i is $n - i$ (for all i), as in step 2.5.1 we remove a single vertex from the previous graph. Due to the fact that the number of iterations of the main loop is $O(\log n)$, we get that the number of vertices from the hidden clique in each G_i is at least $k - O(\log n) = (1 - o(1))k$ (as $k \gg 2\log_2 n$). Let X_i be a random variable that denotes the number of vertices from the hidden clique that were sampled in step 2.1 in iteration i. We have that

$$\Pr[X_i = 0] \leq \left(1 - \tfrac{\log_2 n}{k}\right)^{k - O(\log n)} \underset{k \gg 2\log_2 n}{=} \left(1 - \tfrac{\log_2 n}{k}\right)^{(1-o(1))k} \leq e^{-(1-o(1))\log_2 n} \leq n^{-1/2}.$$

Thus, by a union bound, the probability that at least in one iteration we will not sample a vertex from the hidden clique in step 2.1 is $o(1)$. Therefore, with high probability, we will not fail in step 2.4 in any of the iterations. Thus, in step 2.4 we have a set S which contains at least one vertex of the hidden clique, and therefore we will certainly find it at the end of the binary-search-type procedure in steps 2.5.1-2.5.6, using the answers of the queries. Therefore, at the end of the main loop, the set T will contain $10 \log_2 n$ vertices of the hidden clique. We claim that T is a random subset of the hidden clique (of said size). To see this, observe that every vertex of the hidden clique is equally likely to be sampled into the set S in step 2.1 (in every iteration). We then observe that every vertex from the hidden clique that was sampled to S is equally likely to "survive" to the next iteration of the inner loop in step 2.5 (due to steps 2.5.2 and 2.5.5). Therefore, we conclude that T is indeed a random subset of the hidden clique. Hence, by Lemma 4.5, in step 3 we will recover the whole hidden clique (with high probability).

Clearly, the running time of the algorithm is polynomial in n. As for the query complexity, we observe that in each iteration of the main loop, we perform one query in step 2.3, and $O(\log n)$ queries during steps 2.5.1-2.5.6, as we reduce the size of the set S by half each time. Thus, the total query complexity is $O(\log^2 n)$. ■

6 Finding Large Cliques in Semi-Random Graphs

We analyze semi-random graphs that are constructed as follows. First, a random graph G is drawn from $G(n, \frac{1}{2})$. Then, an adversary adds to it a certain amount of edges, resulting in a semi-random graph we denote by H, in such a way that $\omega(G) < \omega(H)$. That is, the adversary must enlarge the size of the maximum clique in G. We further assume that $\omega(H) \geq (2 + \varepsilon) \log_2 n$, for some fixed $\varepsilon > 0$ (i.e., the growth is at least by a multiplicative factor with respect to $\omega(G) \approx 2 \log_2 n$). Furthermore, we enforce the following additional limitation on the adversary: every vertex outside the largest clique in H must have a degree that is smaller than the degree of any vertex inside the largest clique in H, unless it was bigger initially (i.e., in G).

We first show an algorithm that finds the largest clique in a semi-random graph H (constructed as above) whenever $\omega(H) \geq c\sqrt{n \log n}$ (for a sufficiently large constant $c > 0$), and then we show that we cannot approximate the largest clique in H within a factor of $2 - \varepsilon$, for every fixed $\varepsilon > 0$, assuming that there is no algorithm that solves the Hidden Clique problem for $k = o(\sqrt{n})$.

Proposition 6.1. *Let H be a semi-random graph constructed as above. If $\omega(H) \geq c\sqrt{n \log n}$ for a sufficiently large constant $c > 0$, then the vertices of the largest clique in H would, almost surely, be the ones with the largest degrees in H, and so can be found in polynomial time.*[1]

Proof: Using a standard combination of Chernoff-Hoeffding bounds and union bound, we get that with high probability, the degrees of all the vertices in G are in the range $[n/2 - \sqrt{n \log n}, n/2 + \sqrt{n \log n}]$. Let us look at the largest clique in H, which we denote by C. By the assumption, we have that $|C| \geq c\sqrt{n \log n}$ (for a sufficiently large constant $c > 0$). As the induced subgraph $G[C]$ is a truly random graph on $|C|$ vertices, we have that with high probability, the degree of every vertex of $G[C]$ is in the range $[|C|/2 - \sqrt{|C| \log |C|}, |C|/2 + \sqrt{|C| \log |C|}]$. However, the degree of each vertex in the induced subgraph $H[C]$ is $|C|-1$, as it is a clique in H. Thus, with high probability, the edges added by the adversary have increased the degree of every vertex in C by at least $\delta|C|$, for some constant $\delta > 0$. Therefore, with high probability, the degree (in H) of every $v \in C$ is at least $n/2 - \sqrt{n \log n} + \delta c\sqrt{n \log n} > n/2 + \sqrt{n \log n}$ (for large enough c). By the additional limitation we have enforced on the adversary, the degree (in H) of every vertex $u \in V(H) \setminus C$ is smaller than the degree of every vertex in C, unless it was bigger initially (i.e., in G), which occurs with probability $o(1)$. Therefore, almost surely, the vertices of the largest clique in H would be the ones with the largest degrees in H, and so can be found in polynomial time. ■

Proposition 6.2. *If there exists a polynomial-time algorithm that approximates the largest clique in a semi-random graph H (constructed as above) within a factor of $2 - \varepsilon$, for some fixed $\varepsilon > 0$, then there exists an algorithm that solves the Hidden Clique problem for all $6 \log_2 n \ll k = k(n) = o(\sqrt{n})$.*

Proof: Let A be a polynomial-time algorithm that approximates the largest clique in a semi-random graph H within a factor of $2 - \varepsilon$, for some fixed $\varepsilon > 0$. We define the following algorithm B, that solves the Hidden Clique problem for $6 \log_2 n \ll k = k(n) = o(\sqrt{n})$. On input $G \leftarrow G(n, \frac{1}{2}, k)$, algorithm B will execute A on the graph G, and get an output Q. It will then compute the set of all common neighbors of the vertices in Q, denoted T, and return $Q \cup T$.

We now prove the correctness of algorithm B. We first observe that G is a legal input for algorithm A, as an adversary is allowed, in particular, to plant a k-clique in the random graph it gets, as long as $k \gg 2 \log_2 n$. Let K denote the vertices of the hidden clique in G, and let

[1] Similar observation was presented by Kučera [Kuč95] regarding the Hidden Clique problem.

D denote the set of vertices of the clique, returned by algorithm A when executed on G. By our assumption, $|D| \geq k/(2-\varepsilon) \gg 3\log_2 n$ (as $k \gg 6\log_2 n$), and thus, with high probability, D is contained in the hidden clique K. By standard combination of Chernoff-Hoeffding bounds and union bound, we have that with high probability, for every $v \in V(G)\backslash K$, its degree into K in G' (the underlying random graph of G) is at most $k/2 + O(\sqrt{k\log n})$. As $|D| \geq k/(2-\varepsilon)$, we have that all its common neighbors are the remaining vertices of the hidden clique (with high probability). Thus, algorithm B succeeds to recover the hidden clique, with high probability. Clearly, B runs in polynomial time. ■

Remark: As mentioned in Section 1, there are a few polynomial-time algorithms that approximate the largest clique in a random graph G drawn from $G(n, \frac{1}{2})$ within a factor of 2, but no polynomial-time algorithm is known to achieve an approximation ratio of $2-\varepsilon$, for some fixed $\varepsilon > 0$. However, it was not yet proven that such an algorithm does not exist (based on some hardness assumption). Proposition 6.2 proves this claim for semi-random graphs, which include the class of random graphs.

References

[AKS98] N. Alon, M. Krivelevich, and B. Sudakov. Finding a large hidden clique in a random graph. *Random Structures Algorithms*, 13(3-4):457–466, 1998.

[AS92] N. Alon and J. H. Spencer. *The probabilistic method*. John Wiley & Sons, Inc., New York, 1992.

[AV11] B. P. Ames and S. A. Vavasis. Nuclear norm minimization for the planted clique and biclique problems. *Mathematical programming*, 129(1):69–89, 2011.

[BHK+16] B. Barak, S. B. Hopkins, J. Kelner, P. Kothari, A. Moitra, and A. Potechin. A nearly tight sum-of-squares lower bound for the planted clique problem. In *Foundations of Computer Science (FOCS), 2016 IEEE 57th Annual Symposium on*, pages 428–437. IEEE, 2016.

[BS95] A. Blum and J. Spencer. Coloring random and semi-random k-colorable graphs. *Journal of Algorithms*, 19(2):204–234, 1995.

[CF14] G. Cormode and D. Firmani. A unifying framework for ℓ_0-sampling algorithms. *Distributed and Parallel Databases*, 32(3):315–335, 2014.

[DGP14] Y. Dekel, O. Gurel-Gurevich, and Y. Peres. Finding hidden cliques in linear time with high probability. *Combinatorics, Probability & Computing*, 23(1):29–49, 2014. doi:10.1017/S096354831300045X.

[DKST01] M. Dawande, P. Keskinocak, J. M. Swaminathan, and S. Tayur. On bipartite and multipartite clique problems. *Journal of Algorithms*, 41(2):388–403, 2001.

[DM15a] Y. Deshpande and A. Montanari. Finding hidden cliques of size $\sqrt{N/e}$ in nearly linear time. *Found. Comput. Math.*, 15(4):1069–1128, August 2015. doi:10.1007/s10208-014-9215-y.

[DM15b] Y. Deshpande and A. Montanari. Improved sum-of-squares lower bounds for hidden clique and hidden submatrix problems. In *Conference on Learning Theory*, pages 523–562, 2015.

[Fei04] U. Feige. Approximating maximum clique by removing subgraphs. *SIAM Journal on Discrete Mathematics*, 18(2):219–225, 2004.

[FK00] U. Feige and R. Krauthgamer. Finding and certifying a large hidden clique in a semirandom graph. *Random Structures Algorithms*, 16(2):195–208, 2000. doi:10.1002/(SICI)1098-2418(200003)16:2<195::AID-RSA5>3.0.CO;2-A.

[FK03] U. Feige and R. Krauthgamer. The probable value of the Lovász–Schrijver relaxations for maximum independent set. *SIAM Journal on Computing*, 32(2):345–370, 2003.

[FK04] U. Feige and S. Kogan. Hardness of approximation of the balanced complete bipartite subgraph problem. *Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot, Israel, Technical Report MCS04*, 2004.

[FR10] U. Feige and D. Ron. Finding hidden cliques in linear time. In *21st International Meeting on Probabilistic, Combinatorial, and Asymptotic Methods in the Analysis of Algorithms*, pages 189–204. DMTCS Proceedings, 2010. Available from: http://www.dmtcs.org/dmtcs-ojs/index.php/proceedings/article/view/dmAM0114.

[Hås96] J. Håstad. Clique is hard to approximate within $n^{1-\varepsilon}$. In *37th Annual Symposium on Foundations of Computer Science*, pages 627–636, 14–16 October 1996.

[HKP+16] S. B. Hopkins, P. Kothari, A. H. Potechin, P. Raghavendra, and T. Schramm. On the integrality gap of degree-4 sum of squares for planted clique. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 1079–1095. SIAM, 2016.

[Jer92] M. Jerrum. Large cliques elude the metropolis process. *Random Structures & Algorithms*, 3(4):347–359, 1992.

[Joh87] D. S. Johnson. The NP-completeness column: An ongoing guide. *Journal of algorithms*, 8(2):285–303, 1987.

[Kar72] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations (Proc. Sympos.)*, pages 85–103. Plenum Press, 1972.

[Kar76] R. M. Karp. The probabilistic analysis of some combinatorial search algorithms. In *Algorithms and complexity (Proc. Sympos.)*, pages 1–19. Academic Press, 1976.

[KS98] M. Krivelevich and B. Sudakov. Coloring random graphs. *Information Processing Letters*, 67(2):71–74, 1998.

[Kuč95] L. Kučera. Expected complexity of graph partitioning problems. *Discrete Appl. Math.*, 57(2-3):193–212, 1995.

[McS01] F. McSherry. Spectral partitioning of random graphs. In *Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on*, pages 529–537. IEEE, 2001.

[MPW15] R. Meka, A. Potechin, and A. Wigderson. Sum-of-squares lower bounds for planted clique. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 87–96. ACM, 2015.

[Zuc06] D. Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 681–690. ACM, 2006.