Reducing the Randomness Complexity of Property Testing, with an Emphasis on Testing Bipartiteness

Or Sheffet

Abstract

Property testers are algorithms whose goal is distinguishing between inputs that have a certain property and inputs which are far from all instances with this property. We show that for a wide variety of properties, there exists no deterministic tester that queries only a sublinear number of input entries. Therefore, most sublinear property testers must be probabilistic algorithms. Nevertheless, we aspire to reduce the randomness complexity of property testers *without* increasing their query complexity. We also introduce and motivate the Q-R complexity of a tester, which is the number of queries that a tester makes multiplied by its randomness complexity.

We reduce the randomness complexity of testers mostly by using randomness-efficient hitters and samplers, rather than the ones that use total independence. This alone achieves a great improvement in the Q-R complexity of many testers. In addition, we focus on the bipartiteness tester presented in [GGR98], and reduce its Q-R complexity. To that end, we not only use randomness-efficient hitters and samplers, but also modify the original tester, in more than one fashion.

We also present some general results regarding property testers. In particular, we present a general, non-explicit, scheme to reduce the randomness complexity of all property testers that share the same basic outline.

Contents

1 Introduction

This thesis focuses on reducing the randomness complexity of property testers; that is, reducing the number of random bits that testers require. We show that testing for a property using sublinear number of queries inherently requires randomness, so we cannot fully derandomize testers without allowing them to view almost the entire input. Still, we wish to use as few random bits as possible, for two reasons. The first is that reducing computational resources is, in general, a good paradigm. The second is due to the implementation of property testers under what we call the "one weak random source" assumption. We show that one is able to reduce the randomness complexity without increasing the query complexity of many property testers.

1.1 A Brief Introduction to Property Testing

Property Testing, a concept invented more than 10 years ago by Goldreich, Goldwasser and Ron [GGR98], is in short the notion of deciding whether a given input is a "Yes"-instance or "far" from any "Yes"-instances. More formally - given some domain D, let $\mathcal{F}$ be the set of functions $\{f : D \to \{0,1\}^*\}$. Let Π be some property of $\mathcal{F}$ (we may consider Π to be some subset of $\mathcal{F}$). We say some g is *ϵ-far* from having Π if for every $f \in \Pi$, the set $\{x \in D;\ f(x) \neq g(x)\}$ is of size at least $\epsilon|D|$. Then a *Π-tester* is a probabilistic oracle machine M that gets as an input a distance parameter $\epsilon > 0$ and has oracle access to f such that

- If $f \in \Pi$, then $Pr[M^f(\epsilon) = 1] \geq 2/3$.
- If f is ϵ-far from having Π, then $Pr[M^f(\epsilon) = 0] \geq 2/3$.

If for every f in Π it holds that $Pr[M^f(\epsilon) = 1] = 1$ then we say that the tester is 1-sided.[1]

Note that we are not guaranteed anything if f doesn't belong to Π yet f isn't ϵ-far from Π. Thus, in fact, we have a relaxation of the notion of "deciding" a property (in which we should distinguish between the case where $f \in \Pi$ and the case where $f \notin \Pi$).

We define the *query complexity* of the tester to be the number of queries that M makes, and the *randomness complexity* of the tester to be the number of random bits M requires. The *error probability* of the tester is said to be the probability that T accepts some f which is ϵ-far from Π, or, if T is 2-sided, the probability of T to reject some $f \in \Pi$. By definition, the error probability of T is $1/3$, but sometimes we may wish to increase or decrease it, in which case we will state clearly that we take a property testers with error probability δ (for some $\delta \neq 1/3$). We note that by using the usual amplification methods, one can reduce the error probability of the tester.[2]

[1] We may refer to a standard tester as *2-sided tester*, to differentiate it from a 1-sided tester.

[2] We refer to Comment 1.6, where we discuss the cost of the usual amplification methods.

An equivalent model is the one in which M is a probabilistic random access machine. Given an input, M queries several (random) entries of its input, according to the random string it was fed. Again, M should distinguish, according to the part of the input it read, whether its input have Π or whether it is ϵ-far from Π. In this model, the query complexity of M is the number of input entries that M reads.

As a convention, we assume $D = [N] = \{0,1\}^n$ for some integer n. We wish to find property testers with query complexity of $o(N)$, since we usually consider input instances to be very large. That is, we assume N be very big. So big in fact, that reading the entire input is an infeasible task. Thus, we use property testers to give us an approximated decision to whether our instance is a "Yes" instance or not. If the property tester queries $\Omega(N)$ places in the domain, we manage to save no time by using the property tester, and we might as well decide if the input is a "Yes" instance or not, by querying all N entries.

A special role in the field of property testing is reserved for graph property testers. Several "natural" graph properties have been studied extensively throughout the years, such as k-connectivity, k-colorability, minor-exclusion, subgraph-freeness, etc. Furthermore, deciding whether a given graph has a certain graph-property or not, is a common task in computer science. When dealing with graphs of huge size, a graph-property tester may be the only feasible way to give an educated guess whether they have some property or not.

We denote a graph as $G = (V, E)$ (with no multiple edges or loops), where its set of vertices is $V = V(G) = [N] = \{0,1\}^n$. We consider two major representations of G:[3]

- For *dense* graphs, let A be the adjacency matrix of G. That is A is a matrix of size $N \times N$ containing zeros and ones, where $A_{i,j} = 1$ if and only if $\langle v_i, v_j \rangle \in E(G)$. This is called the *matrix adjacency model*, where the function represent G is of size $\binom{N}{2} = O(N^2)$. A query of the tester is of the form of a pair of vertices $(v_i, v_j) \in V \times V$, and is answered by a single bit reply, the value of $A_{i,j}$.

- For *sparse* graphs, we assume we have some bound d on the max-degree of G. We represent G by N adjacency lists, each of length d. The jth element in the list of v_i is u if indeed u is the jth neighbor of v_i, or $\emptyset$ if v_i has less than j neighbors. This is called the *bounded degree model*, where the function representing G is of size dN, and since we assume $d = O(1)$, then it is of size $O(N)$. A query of the tester is of the form of a pair $(v, j) \in V \times [d]$ and is answered by a n-bit reply, which is, as mentioned, either some $u \in V$ or $\emptyset$.

It turns out that there exists a great difference between the two representations, and while several properties are efficiently testable in one model (i.e. with query complexity $poly(1/\epsilon)$), testing them in the other requires a great increase in the query complexity.

[3][PR02] has introduces a third model, using some bound on $|E(G)|$. We do not deal with this model in this thesis.

1.2 On the Importance of Randomness for Property Testers

One crucial observation is that property testing is a procedure that inherently requires randomness. For many natural properties no deterministic tester with query complexity of $o(N)$ exists. In fact, Canetti et al [CEG95] gave a lower bound on the randomness of samplers (objects which are a special case of property tester and will be explained later on). As we show in Section 3.1, a similar proof applies to a wide variety of graph property testers (in the matrix adjacency model) as well. Similarly, the proof's outline holds for other property testers, and not only for graph properties.

So one cannot get rid altogether of the need for randomness in property testers. Nevertheless, it does not mean that one can disregard the randomness complexity of a tester. Randomness should be looked at a valuable resource for computation, and as such it is a good costume to use as few random bits as possible. We believe that one should measure the efficiency of a tester not only by its query complexity, but by its randomness complexity, running time, space complexity, and any other resource that the tester might exploit. In this work, we focus on the query complexity and the randomness complexity.

As we show next, reducing the randomness complexity of property testers have even greater importance than just the good costume of saving computational resources. Another reasoning for reducing the randomness complexity of property testers, presented also in [**?**], involves the issue of using weak sources of randomness. We elaborate.

1.2.1 Weak Sources of Randomness

Let T be the TM that is a tester for some property Π, and denote its randomness complexity as r. The machine T is a probabilistic TM, and often it is said that T tosses coins, and operates according to the results of these coins.[4] However, what we actually mean is that T has a random tape with r totally random bits on it. This random tape comes from some *randomness source*, meaning, the bits on the random tape are the result of some random variable taking values uniformly in $R = \{0,1\}^r$.

The question of sources of randomness, and how common they in fact are, is a very complex and philosophical question, which is outside the scope of computer science. Nonetheless, most property testing algorithms assume that the tester has an access to a source of randomness, which, as said, takes values *uniformly* over some set R. In this work, we wish to base property testing on a weaker assumption.

We assume that sources of randomness are scarce, and that they are "impure", in some sense. We first introduce some definitions:

Definition 1.1. *Let X be a random variable taking values in $\{0,1\}^n$. The* min-entropy *of*

[4] In fact, even in this thesis we analyze the randomness complexity of several algorithms by describing the number of coins they toss.

X is the largest k so that for every $x \in \{0,1\}^n$, it holds that $Pr[X = x] \leq 2^{-k}$. We denote $H_\infty(X) = k$.

We assume that we only have access to one[5] "weak" source of randomness that takes values in $\{0,1\}^n$ for some integer n, but has only k min-entropy, for some $k < n$. We denote the *rate* of the source as $\frac{k}{n}$. We wish to extract from the weak source a random variable that is distributed uniformly, or almost uniformly, over $\{0,1\}^k$.

Definition 1.2. *Let X_1 and X_2 be two distributions over the same domain D. Then the* variation distance *or the* statistical distance *of X_1 and X_2 is denoted as $\|X_1 - X_2\|$ and is*

$$\|X_1 - X_2\| = \max_{T \subset D} |Pr_{X_1}[T] - Pr_{X_2}[T]| = \frac{1}{2} \sum_{x \in D} |Pr_{X_1}[x] - Pr_{X_2}[x]|$$

We use the notation U_a to denote the uniform distribution over the set $\{0,1\}^a$. Our goal is to turn X, our weak randomness source with k min-entropy, into a source whose distribution is very close to U_m, where m is as high as possible (obviously $m \leq k$). Unfortunately, this cannot be done without using a few additional truly uniform bits. The entire field of extractors deals with this task.

Definition 1.3. *A function $Ext : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is called a (k, ϵ)-*extractor *if for every random variable X over $\{0,1\}^n$ with $H_\infty(X) \geq k$, we have that $\|Ext(X, U_d) - U_m\| < \epsilon$.*

That is to say that an extractor takes such a "weak" random source, and uses addition d truly random bits, to produce an output which is ϵ-close to uniform over $\{0,1\}^m$.

In the field of extractors, a great deal of research was and is made. Let us state just few known results regarding the connection between d, n and k. [RTS97] have proven a lower bound on d, where $2^d = \Omega\left(\frac{1}{\epsilon^2}(n-k)\right)$. For ϵ taking constant values (for example, $\epsilon = \frac{1}{12}$), we consider this lower bound to be $2^d = \Omega(n-k)$. Both works of [LRVW03] and of [GUV06] give an explicit construction of an extractor, where, for $\epsilon = O(1)$, it holds that $d = O(\log n)$.

Two common cases are usually discussed regarding the connection between the values of n, m, and k, where ϵ is some small constant:

- $k = \Omega(n)$ and $m = \Omega(k)$. In this case $n = O(m)$. In such a case we say that X is a *constant rate* source. Out of the two common cases, constant rate sources seems to be the more natural and simple case. The best known construction of an extractor for constant rate sources is the latest construction of Zuckermann [Zuc06] that uses $d = (1 + o(1)) \log n = (1 + o(1)) \log m$ truly random bits. This matches the lower bound, which for constant rate sources is $d \geq \log(n-k) = (1 - o(1)) \log m$.

[5] We assume that such sources are relatively scarce, and therefore the tester have access to only one "weak" source, and not several.

- $k = n^{\Omega(1)}$ and $m = k^{\Omega(1)}$. In this case $n = m^{O(1)}$. Here the best known constructions of extractors belong to [LRVW03] and [GUV06], where $d = O(\log n) = c \cdot \log m$, for some constant $c > 1$. The lower bound in this case is of the form $d \geq \log(n - k) = \Omega(\log m)$.

1.2.2 A "Typical" Implementation of a Property Tester

So under the assumption that we only have access to one weak random source, can we still accept all "Yes" instances and reject all far from "Yes" instances? The answer to this question is: "yes, but with some cost..." We present here a method of using T and a weak random source in order accept all "Yes" instances and reject all far from "Yes" instances, with error probability of no more than 1/3. We call this method a "typical implementation of T under the one weak random source assumption", or just a "typical" implementation, in short. In the next section, we discuss the cost of such a "typical" implementation.

Fix some property tester T and some distance parameter ϵ for T. Denote the query complexity of T for this fixed distance parameter ϵ, by Q. Denote the randomness complexity of T for this parameter ϵ by r. Assume also that T has error probability of at most 1/12.[6] We are given a random variable X with min-entropy $k \geq r$, and a $(k, \frac{1}{12})$-extractor $Ext : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^r$, where $d = c \cdot \log r$ for some constant $c \geq 1$.

The tester T requires r truly random bits, but we only have an access to a "weak" random source. So we use an extractor Ext, to extract a random variable whose distribution is ϵ-close to the uniform U_r. Therefore, Ext requires $d = O(\log m) = c \cdot \log r$ truly random bits, in order to produce an output whose distribution is ϵ-close to the uniform distribution over $\{0,1\}^r$. Alas, the extractor requires d truly uniform random bits, and we have no feasible way to produce even a single random bit, let alone d bits.

The "typical" implementation of T, given some instance I begins by sampling X one time. Let x be the value sampled. Then, for every possible string $i \in \{0,1\}^d$, we compute $Ext(x, i)$, the output of the extractor on x and i, and we simulate T when fed $Ext(x, i)$ as a random string. We finally accept or reject I by a majority vote of the 2^d simulations we do.

Claim 1.4. *Let T be a property tester with error probability $1/12$, and randomness complexity r. Let X be a random variable over $\{0,1\}^n$ with k min-entropy. Let $Ext : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^r$, be a $(k, 1/12)$-extractor. The probability that the "typical" implementation of T errs, taken over $x \leftarrow X$, is at most $1/3$.*

Proof. Fix an input I for the tester. Let f be the function $f : \{0,1\}^r \to \{0,1\}$, where $f(z) = 1$ if and only if the tester T errs over the input I and the random input z. For example, assume I is a "Yes" instance. Then $f(z) = 1$ if the tester T, when I is its input, and when z is fed to T as its random input, rejects I. Otherwise, $f(z) = 0$.

[6] See Comment 1.6.

We know that $Pr_{z \leftarrow U_r}[f(z) = 1] < 1/12$, because the error probability of T is at most $1/12$. Since by definition of the extractor, $\|Ext(X, U_d) - U_r\| < 1/12$, then also $\|f(Ext(X, U_d)) - f(U_r)\| < 1/12$, and therefore $Pr_{z \leftarrow Ext(X, U_d)}[f(z) = 1] < 1/6$. Note that in order to produce a string z that is distributed according to $Ext(X, U_d)$, we can sample $x \leftarrow X$, and $i \leftarrow U_d$, then compute $z = Ext(x, i)$. We deduce that

$$Pr_{x \leftarrow X,\, i \leftarrow U_d}[f(Ext(x, i)) = 1] < 1/6 \qquad (*)$$

For every $x \in \{0,1\}^n$, denote by P_x the probability $P_x = Pr_{i \leftarrow U_d}[f(Ext(x, i)) = 1]$. Call a $x \in \{0,1\}^n$ *bad* if $P_x \geq 1/2$. We know, by $(*)$ that $E_{x \leftarrow X}[P_y] < 1/6$, and so, the Markov inequality gives that

$$Pr_{x \leftarrow X}\left[P_x > \frac{1}{2}\right] \leq \frac{1/6}{1/2} = \frac{1}{3}$$

All that is left to show, is that when x isn't bad, then the "typical" implementation does not err.

If x isn't bad, it means that $Pr_{i \leftarrow U_d}[f(Ext(x, i)) = 1] < 1/2$. So, for less than half of the values i takes, $f(Ext(x, i)) = 1$. Therefore, less than half of the simulations we perform cause T to err. So the majority of the simulations are correct for I, causing the "typical" implementation to accept I if indeed I is a "Yes" instance, and to reject I if I is ϵ-far from all "Yes" instances. $\square$

1.2.3 The Cost of a "Typical Implementation" and the Q-R Complexity

We have shown that we can still use property testers even under the assumption that we can only access one weak random source, due to what we call a "typical" implementation of testers. But the drawback of this "typical" implementation is that it forces us to perform 2^d simulations of T. Suppose T has query complexity Q. Since we invoke it 2^d times, we make $2^d \cdot Q$ queries to the input. Now, if $2^d \cdot Q = \Omega(N)$ then we are in trouble...

Recall that the extractor from Section 1.2.2 uses $d = c \cdot \log r$ truly random bits. Therefore $2^d \cdot Q = r^c \cdot Q$, which leads us to the following definition:

Definition 1.5. *Let $c \geq 1$ be a constant. Given a property tester T for some property Π, that has query complexity Q and randomness complexity r, we denote its* Q-R^c complexity *by $r^c \cdot Q$. For $c = 1$ we simply refer to Q-R^1 complexity as the* Q-R complexity *of T.*

As we showed, the Q-R^c complexity is the number of queries we perform during one "typical" implementation of T, given an extractor that requires $d = c \cdot \log r$ truly uniform bits. The Q-R complexity applies to the case where we have an extractor for a *constant rate* source, in which case $c = 1$. The analysis of testers in this work, refers solely to the Q-R complexity of testers, and not to their Q-R^c complexity.[7] That is to say that we assume that our weak random source has some *constant rate*, and we use an extractor that requires $\log n = (1 + o(1)) \log r$ truly random bits. Therefore, a "typical" implementation of T using one weak *constant rate* source, makes $O(r)$ simulations of T, thus makes $O(r \cdot Q)$ queries. Thus, the higher the randomness complexity of a tester is, the higher is Q-R complexity is.

For example, the bipartiteness tester for bounded degree graphs [GR98] has query complexity of $\tilde{O}(\sqrt{N})$, and randomness complexity of $\tilde{O}(\sqrt{N})$, and therefore, its Q-R complexity is $\tilde{O}(N)$. Thus, a "typical" implementation of this tester may query N input entries. We show in Section 3.2.1 how to reduce the randomness complexity of this tester, to achieve a Q-R complexity of $\tilde{O}(\sqrt{N})$.

Comment 1.6. Recall that the "typical" implementation uses a property tester with error probability of $\frac{1}{12}$. Assuming we only have a property tester T with error probability $\frac{1}{3}$, we have to amplify its error probability. Let Q and r be the query complexity and the randomness complexity of T, respectively. The standard way to reduce the error probability of T, is to perform K simulations of T (for some constant K), and take a majority vote between all these K simulations. These simulations are performed using K independent random strings, each of length r. Therefore, we use $K \cdot r$ random bits, and perform $K \cdot Q$ queries. Since $K = O(1)$, this does not change asymptotically the complexities of T. In Appendix A we discuss another possible way to reduce the error probability of T. By using samplers and hitters of the appropriate parameters, we can sample K random strings for T, instead of picking them independently, and then perform the K simulations. This technique still has query complexity $K \cdot Q$, but the randomness complexity remains r. Therefore, we recommend the use of samplers and hitters, whenever K is not a constant.

1.3 Related Work

The work of Goldreich and Sudan [?] and the work of Ben-Sasson et al [?] discuss randomness-efficient low-degree testing. The motivation in these works is to reduce the length of a PCP, and they focus on testers for low-degree polynomials. Given a function $f : \mathbb{F}^m \to \mathbb{F}$, a low-degree tester accepts f if f is a polynomial of degree at most d (for some fixed d), and rejects f if f is any function which is far from being a polynomial of degree at most d. Goldreich and Sudan [?] prove that there exists a (non-explicit) low-degree tester that makes $O(|\mathbb{F}|^m \cdot \log(|\mathbb{F}|))$ queries, and Ben-Sasson et al [?] give an explicit construction with slightly worse query complexity. Another work is of Shpilka and Wigderson [?], in which they construct a randomness-efficient homomorphism tester. Given two fixed groups, G and H, and a function $f : G \to H$, a homomorphism tester should accept f if f is a homomorphism

[7] In this work we reduce the randomness complexity of known testers, while maintaining their original query complexity. This framework guarantees that reducing the Q-R complexity of a tester also reduces its Q-R^c complexity.

of groups (i.e. if for every $x \in G$ and $y \in H$, it holds that $f(x \cdot y) = f(x) \cdot f(y)$), and reject f if f is far from being a homomorphism. The construction in [**?**] is of a tester that requires only $(1 + O(1)) \log(|G|)$ random bits.

1.4 Organization

We start with preliminary notions and definitions in the next section (Section 2), where we also define the samplers and hitters we use in this work. We refer the reader to Appendix A to a more elaborated discussion regarding sampler and hitters. In Section 3 we present several results for reducing the randomness complexity of graph property testers. We discuss testers for properties of graphs in the adjacency matrix model, and give a lower bound and a non-constructive upper bound on their randomness complexity. We also show how to reduce the randomness complexity of several testers for bounded degree graphs. We apply the same technique for all of the testers we present - using randomness efficient hitters and samplers, instead of the ones that use total independence. In Section 4 we focus on the Q-R complexity of a single tester, the tester for bipartiteness of graphs in the matrix adjacency model. We show how to decrease its Q-R complexity, using not only better samplers and hitters, but also by modifying the tester, in more than one manner. We present all the modifications we found, including those that give *worse* Q-R complexity than the original, since we believe that these modifications are applicable to other property testers, and give good guidelines for reducing the Q-R complexity of other testers as well. We end with conclusions and open problems, in Section 5.

2 Preliminaries

2.1 General Notations

For readability, we often assume various quantities are integers, when they are not necessarily so. It is not hard to see that this does not affect our analysis. For an integer n, let $[n]$ denote the set $\{1, 2, \dots, n\}$. All $\log x$ functions used in this work are $\log_2(x)$, unless mentioned otherwise. Given some set X and some subset $A \subset X$, we say that A is of *density* $\frac{|A|}{|X|}$, and denote $\rho(A) = \frac{|A|}{|X|}$. In order to avoid tedious notation, we denote $O(x \cdot poly(\log(x)))$ and $\Omega(x \cdot poly(\log(x)))$ by $\tilde{O}(x)$ and $\tilde{\Omega}(x)$ respectively.

A graph $G = (V, E)$ is always an undirected, simple graph, with no multiple edges or self-loops. We sometimes refer to V and E as $V(G)$ or $E(G)$. The size of G is the number of vertices in V, which is usually considered to be $N = 2^n$. We also identify the set V with the set $\{0, 1\}^n$, or alternatively, with the set $[N]$. An edge in G is denoted as an *unordered pair* $\langle u, v\rangle$. A query, that a tester for graphs represented by a adjacency matrix makes, is of the form of an *ordered* vertex-pair, (u, v).

Given a subset $X \subset V$ we denote by $G[X]$ the subgraph of G that is induced on X. That is $G[X] = (X, E')$ where $\langle u, v\rangle \in E'$ if and only if both u and v belong to X, and the edge $\langle u, v\rangle$ belongs to E.

For any $u \in V$ we denote u's neighbors by $\Gamma(u)$. If G is a graph where the neighbors of each vertex are labeled, then we denote the ith neighbor of u by $\Gamma_i(u)$. For a subset $U \subset V$ we denote by $\Gamma(U)$ the set of neighbors that are adjacent to at least one vertex from U.

A (N, D, λ)-expander is a graph G over N vertices, each of degree D, where the second largest eigenvalue of the adjacency matrix of G is λ.

2.2 Samplers and Hitters

We present here a very succinct description of the notion of samplers and hitters, and the ones that we use throughout this work. We refer the reader to Appendix A, where we elaborate on the subject.

We begin with the formal definition of samplers and hitters. Fix n and set $N = 2^n$. Let $f : [N] \to [0, 1]$ be any function. Denote $E[f] = avg(f) = \frac{1}{N}\sum_{x\in[N]} f(x)$.

Definition 2.1. *A (ϵ, δ)-sampler S with query complexity Q and randomness complexity r, is a probabilistic oracle machine that for every n and every $f : [N] = \{0, 1\}^n \to [0, 1]$ tosses at most r coins and queries no more than Q of the entries of f, in order to output an estimation $Est_S(f)$ that satisfies*

$$Pr[|Est_S(f) - E[f]| > \epsilon] < \delta.$$

We call ϵ the *accuracy* of the sampler, and $1 - \delta$ the *confidence* of the sampler.

Definition 2.2. *For any $\epsilon, \delta > 0$ and $c \geq 1$, a c-*hitter *S for sets of density ϵ and confidence $1-\delta$, is a probabilistic oracle machine with the following property. For every set $T \subset N = \{0,1\}^n$ with density $\rho(T) > \epsilon$, the TM S uses at most r random bits, and queries at most Q points in N, which are denoted as $x_1, x_2, \ldots, x_Q$, that satisfy*

$$Pr\left[\,|\{x_1, x_2, \ldots, x_Q\} \cap T| < c\right] < \delta$$

In most of this work we use 1-hitters, which we simply refer to as hitters. It is obvious that a c-hitter is an extension of the standard notion of a hitter. The reason for introducing c-hitters, and not standard hitters, is discussed in Appendix A.

In this work we use four hitters and samplers. Whenever we refer to "a hitter/sampler which is linear/logarithmic in confidence", we mean one of the four presented below. We give their full description in Appendix A, where we also prove there their correctness. Here we just give a short description of their complexities. Fix any $n, \epsilon, \delta > 0$. Then we have:

1. A (ϵ, δ)-sampler with query complexity $O\left(\frac{1}{\epsilon^2\delta}\right)$ and randomness complexity n. We refer to this sampler simply as the sampler linear in confidence.

2. A c-hitter, for any $c = O(1)$, for sets of density ϵ and with confidence $1-\delta$, with query complexity $O\left(\frac{1}{\epsilon\delta}\right)$ and randomness complexity n. We refer to this c-hitter simply as the hitter linear in confidence.

3. A (ϵ, δ)-sampler with query complexity $O\left(\frac{1}{\epsilon^2}\log(1/\delta)\right)$ and randomness complexity $n + O(\log(1/\delta))$. We refer to this sampler simply as the sampler logarithmic in confidence.

4. A c-hitter, for any $c = O(1)$, for sets of density ϵ and with confidence $1-\delta$, with query complexity $O\left(\frac{1}{\epsilon}\log(1/\delta)\right)$ and randomness complexity $n + O(\log(1/\delta))$. We refer to this c-hitter simply as the hitter logarithmic in confidence.

3 Randomness-Efficient Graph Property Testers

We turn our attention to reducing the randomness complexity of graph property testers. As it was explained in the introduction, graphs can be represented in two different fashions, as an adjacency matrix or as adjacency lists. The first model is more suitable for dense graph and the second is usually used for graphs of bounded degree. Similarly, this section is divided into two parts, one dealing with each model.

For the adjacency matrix model, we discuss general methods for reducing the randomness complexity of any graph property testers. We first give a lower bound for the randomness complexity of property testers for some natural class of properties. The lower bound is roughly $\log(N^2/Q)$, where Q is the query complexity of the tester. In particular, the lower bound we prove, shows that for these properties, testers have no *deterministic* equivalent with query complexity $o(N^2)$. Therefore, unlike BPP, there is no hope for us to achieve a full derandomization of property testers, while keeping the query complexity of the testers sublinear. On the other hand, we show a non-constructive scheme to reduce the randomness complexity of property testers. We also present a non-constructive scheme that reduces the randomness complexity of all property testers with the same general outline, regardless of which property they check for.

For bounded degree graphs, we show how to reduce the randomness complexity of several specific property testers (such as bipartiteness, connectivety etc.). Note that reducing the randomness complexity of testers, automatically reduces their Q-R complexity. As we show, even the basic scheme of using better hitters and samplers than the ones that use total independence, has a dramatic effect for the Q-R complexity of the testers. This is best shown by the bipartiteness tester for bounded degree graphs, in Section 3.2.1. Its original version [GR98] has Q-R complexity of $O(N)$, which we were able to reduce to roughly $O(\sqrt{N})$.

3.1 Reducing the Randomness Complexity of Property Testers in the Adjacency Matrix Model

This part deals with some general results regarding property testing of graphs in the adjacency matrix model. We show a lower bound on their randomness complexity, and an almost matching (non-explicit) upper bound.

3.1.1 A Lower Bound on the Randomness Complexity of Graph Property Testers

As usual, before moving on to the claims of this section, we start with some notation.

Fix $N > 0$. Let S_N be the group of permutations over N elements. Let $G = (V, E)$ be a graph over N vertices. We denote the isomorphic copy of G under ϕ as $\phi(G)$, which is defined formally as the graph $\phi(G) = (V, E')$, where $\langle u, v\rangle \in E$ if and only if $\langle \phi(u), \phi(v)\rangle \in E'$.

Note that a graph property Π of N-vertex graphs is not merely a set of graphs, but rather must be closed under isomorphism. Formally, for any $G \in \Pi$, it holds that any graph isomorphic to G must also belong to Π. That is to say that if $G \in \Pi$ then for any $\phi \in S_N$, then the graph $\phi(G)$ also belongs to Π.

As it will be shown, the fact that Π is closed under isomorphism is important for the proof of the lower bound.

Theorem 3.1. *Fix $N > 6$, $\epsilon > 0$, and let Π be a graph property for graphs over N vertices. Assume that the complete graph over N vertices is 3ϵ-far from Π. Then any non-adaptive property tester for Π with distance parameter ϵ and query complexity Q, must have randomness complexity of at least $2n - \lg Q - O(1)$.*

Before going into the proof we wish to note that for many graph properties, the complete graph over N vertices, denoted K_N, is indeed far from having these properties. Let us name just a few of such graph properties: k-colorability, having max-cut (or max-clique) of size at most ρN^2 for some $\rho < 1$, minor-exclusion for some minor H, and subgraph-freeness. An analogous case is when we have a graph property where the empty graph is far from it. For such a graph property the theorem also holds, with basically the same proof.

Proof. The proof basically follows the outline of the proof in [CEG95], regarding the lower bound on the randomness complexity of samplers. Let T be a property tester for Π with distance parameter ϵ. Denote its randomness complexity by r. Let R be the set of all possible random strings of T. For every $\omega \in R$, let $D_\omega \subset V \times V$ be the set of vertex-pairs that T queries when fed ω as a random string.[8] Note that the random string suffices for us to determine which vertex-pair T queries, since T is non-adaptive. We know that T makes at most Q queries for every random string it is fed, so we have that for every $\omega \in R$ it holds that $|D_\omega| \leq 2Q$. Let $D = \bigcup_{\omega \in R} D_\omega$, be the set of possible queries that T might make. Clearly $|D| \leq 2|R| \cdot Q$.

We wish to show that $|D| \geq \frac{1}{2}N^2$. That would mean that $|R| \cdot Q \geq \frac{1}{4}N^2$ and would give us the required.

The way for us to show this, is to assume that the opposite hold, i.e. $|D| < \frac{1}{2}N^2$, and then to show that T can be cheated. Meaning, we show that exists two graphs, one is in Π and the other is ϵ-far from Π, which T acts the same way for either one of them. So if T accepts the first with probability at least 2/3, then it also accepts the other (with the same probability).

Proposition 3.2. *If $|D| < \frac{1}{2}N^2$ then there exists some G and H, such that $G \in \Pi$ and H is ϵ-far from Π, and for every $(u, v) \in D$, the edge $\langle u, v\rangle \in E(H)$ if and only if the edge $\langle u, v\rangle \in E(G)$.*

[8] Since an edge $\langle u, v\rangle \in E(G)$ is a non-ordered pair, then if T queries some vertex pair (u, v), then we add both the vertex-pair (u, v) and the vertex-pair (v, u) to D_ω.

Assume that Proposition 3.2 holds. Suppose that indeed $|D| < \frac{1}{2}N^2$, and let G and H be as in the claim. Note that when T gets H as an input, for every random string it is fed, we have that T views the *exact same* edges, in the *exact same* locations, as if its input is G. That is because any query that T makes is some vertex-pair $(u, v) \in D$, and so T gets the same reply from both G and H, which have representing matrices that are identical over the vertex-pairs in D. So when T gets H as its input, and is fed any random string in R, then T acts *identically* as though its input is G. Therefore, for every $\omega \in R$, when the random string of T is ω, then we have that T accepts G if and only if it accepts H. That means that $Pr[T^H = 1] = Pr[T^G = 1]$.

This gives an immediate contradiction to the fact T is a tester for Π. On the one hand we have that T accepts G, which is a graph in Π, with probability at least 2/3. On the other hand we have that T rejects H, which is ϵ-far from Π, with probability at least 2/3, which means that T accepts H with probability at most 1/3. Contradiction follows, and our lower bound is proven. $\square$

Proof of Proposition 3.2. Let G be a graph over N vertices, which is in Π, and that among all graphs that are in Π, it holds that G has the maximal number of edges. That is to say that exists no other graph $G' \in \Pi$ so that $|E(G')| > |E(G)|$. Let $I(G)$ denote the number of entries in the representing matrix of G that are 1. The maximality of G gives that for every other $G' \in \Pi$ we have that $I(G) \geq I(G')$.

Let $Z(G)$ be the set of vertex-pairs $(u, v) \in V \times V$ so that $\langle u, v \rangle \notin E(G)$. That means that $Z(G)$ is the set of all vertex-pairs $(v_i, v_j) \in V \times V$ where $v_i \neq v_j$ and the (i, j)-entry in the representing matrix of G is 0. We know that the complete graph over N vertices is 3ϵ-far from having Π, and that the complete graph has a representation matrix which has 1 in every entry (except for entries on the main diagonal). Therefore exist at least $3\epsilon N^2$ entries in the representing matrix of G (which are not on the main diagonal) that are 0. This means that $|Z(G)| \geq 3\epsilon N^2$.

Assume for a moment that $|Z(G) \setminus D| > \epsilon N^2$. By the definition of D, our assumption is that at least ϵN^2 of vertex-pairs in $Z(G)$ are never queried by T. Then let H be the graph where for every vertex-pair in $|Z(G) \setminus D|$ we add an edge to G connecting that pair of vertices. Clearly, $I(H)$, which is the number of 1-entries in the representing matrix of H, is $I(H) = I(G) + |Z(G) \setminus D| > I(G) + \epsilon N^2$, and therefore H is ϵ-far from G. By the maximality of G it also means that for any $G' \in \Pi$ it holds that $I(G') \leq I(G) < I(H) - \epsilon N^2$. This means that H is ϵ-far from any G' in Π, and so H is ϵ-far from Π. We deduce that if $|Z(G) \setminus D| > \epsilon N^2$ holds, then G and H are two graphs for which the Proposition 3.2 holds.

So far we have assumed that $|Z(G) \setminus D| > \epsilon N^2$. We next show that if this does not hold, then some isomorphic copy of G, denoted as $\phi(G)$, satisfies this condition (i.e. $|Z(\phi(G)) \setminus D| > \epsilon N^2$). Since Π is closed under isomorphism, we have that the graph $\phi(G)$ is in Π. Since every isomorphic copy of G has the same number of edges as G, then it also holds that for every $G' \in \Pi$, we have that $|E(\phi(G))| = |E(G)| \geq |E(G')|$. Therefore, we can construct H as before, only with respect to $\phi(G)$ instead of G, and the claim still holds.

We show that there exists some $\phi \in S_N$ so that $|Z(\phi(G)) \setminus D| > \epsilon N^2$, using the probabilistic method. For any pair $(u, v) \in Z(G)$ denote by $X_{u,v}$ the random variable indicating whether for a randomly and uniformly selected $\phi \in S_N$, the pair $(\phi(u), \phi(v))$ belongs to D or not. We have that

$$E[X_{u,v}] = Pr[X_{u,v} = 1] \leq \sum_{(w_1,w_2)\in D} Pr[(\phi(u) = w_1) \wedge (\phi(v) = w_2)] = \frac{|D|}{N(N-1)} \leq \frac{1}{2} + \frac{1}{N} < \frac{2}{3}$$

Let $X = \sum_{(u,v)\in Z(G)} X_{u,v}$. By linearity of expectation, $E[X] \leq \frac{2}{3}|Z(G)|$. This means that exists some $\phi \in S_N$ so that the number elements of $Z(\phi(G))$ that fall in D is less than $\frac{2}{3}|Z(G)|$, thus, for that ϕ, more than $\frac{1}{3}|Z(G)| \geq \epsilon N^2$ of the vertex-pairs of $Z(\phi(G))$ fall outsize D. We are done. □

We wish to give a few remarks about the lower bound proven in the last theorem.

1. The first remark is to note how essential randomness is for graph property testing. A property tester is assumed to have query complexity which is $o(N^2)$, for otherwise, we consider it to be inefficient. The previous claim give rigorous proof to the fact that no deterministic graph property tester (for non-trivial properties) has query complexity much smaller than N. More formally, if $Q = o(N^2)$ then $r \geq 2n - \log Q > 0$, and therefore the tester must be a probabilistic algorithm. Also note that if $Q = N^{o(1)}$ then $r \geq (2 - o(1)) \cdot n$.

2. The second remark is regarding the connection between r and $\log Q$. There is an obvious way for us to reduce the number of coin tosses that a tester uses by a single bit. Assume that T is one-sided. If T is a tester with query complexity Q and randomness complexity of r, we can construct a tester T' that has randomness complexity $r - 1$. Given a random input of length $r - 1$ for T', it will simulate T assuming the last bit is 0, and also simulate T assuming the last bit is 1. We reject if either one of the simulations rejects, and accept if both simulations accept. Note, however, that the query complexity of T' is twice the query complexity of T. This tradeoff between r and Q shows exactly why we believe the lower bound of $r \geq n - \log Q$ demonstrates the correct correlation between r, n and Q.

3. Another remark is that the lower bound given shows no relation between the randomness complexity of a property tester to the distance parameter ϵ. We admit it to be a drawback, and furthermore, we believe that by using more sophisticated probabilistic techniques, it is possible to show a similar lower bound to the one shown in [CEG95]

for samplers. That is to say, that we believe one can prove, by a more complicated proof than the last proof, that $2^r \cdot Q \geq (1 - O(\epsilon)) \cdot N^2$. However, we believe that this also does not reflect the true connection between r and ϵ, and in fact, the actual lower bound on the randomness complexity of a graph property tester should be something in the line of $r \geq 2n - \log Q + \log(1/\epsilon) - O(1)$.

4. The last remarks is that our hypothesis that the tester is non-adaptive is indeed a drawback. Despite our efforts, we found no way to extend the proof for adaptive testers as well.

3.1.2 An Upper Bound on the Randomness Complexity of Graph Property Testers

We now give the (non-explicit) upper bound. This upper bound holds only for specific testers, known as canonical graph tester. However, Goldreich and Trevisan [GT01] have proven that if Π is a testable graph property by a tester with query complexity Q, then it is testable by a canonical property tester that has query complexity Q^2.

Recall that for any $X \subset V$, we denote $G[X]$ the subgraph of G over the vertices of X. We now define formally a canonical tester.

Definition 3.3. *Let Π be a graph property for graphs over N vertices. T is said to be a* canonical *tester for Π if it gets as an input G, a graph over N vertices, and some distance parameter ϵ, and then, for some integer $K = K_\Pi(N, \epsilon)$, it operates in the following manner:*

1. *Randomly and uniformly selects a subset $X \subset V$ of size K.*

2. *Queries all edges in the subgraph $G[X]$.*

3. *Accepts or rejects G solely based on the adjacency matrix of $G[X]$, disregarding the labeling of the vertices in X.*

Note that a canonical property tester only tosses coins to choose X, and after choosing X it no longer requires any randomness, so by knowing which graph over K vertices it viewed, we can *determine* whether it will accept or reject G. The query complexity of T is $\binom{K}{2}$, and its randomness complexity is $K \log N$. Recall that the error probability of the tester is said to be the probability that T accepts some G which is ϵ-far from Π, or, if T is 2-sided, the probability that T rejects some $G \in \Pi$. We assume that a canonical tester has error probability of at most $1/3$.

Lemma 3.4. *Let T be a canonical tester for some graph property Π. Fix any N and $\epsilon > 0$, and denote K as the number of vertices T picks, randomly and uniformly. For every $0 < \delta < 1/6$, there exists T', a tester for Π, that has the same query complexity as T, randomness* *complexity of* $2n + 2\log(1/\delta) + O(1)$*, and error probability of at most* $1/3 + \delta$*. Furthermore, if* T *is one-sided then so is* T'*.*

Proof. The proof uses the probabilistic method. We use the Chernoff inequality to show that replacing all possible coin tosses of T by a randomly selected set of size $O((N/\delta)^2)$, is likely to give the desired T'.

Denote by $\Omega = \binom{V}{K}$ the set of all possible subsets that the tester selects, or the set of all possible coin-tosses of the tester. Denote $\mathcal{G}$ as the set of all possible graphs over N vertices, which either have Π or ϵ-far from Π. Since T is canonical, then for every $X \in \Omega$ and every $G \in \mathcal{G}$, we can determine what T outputs just by knowing $G[X]$. We denote by $T(G[X])$ the output of T when it views the subgraph $G[X]$. Denote by $\chi(G)$ the indicating function of Π, meaning for every $G \in \Pi$ we have that $\chi(G) = 1$, and for every G which is ϵ-far from Π we have that $\chi(G) = 0$.

We pick randomly a set $R \subset \Omega$, of size $O((N/\delta)^2)$, by repeating $O((N/\delta)^2)$ times the process of selecting uniformly, *independently* and randomly an element from Ω. Denote $r = |R|$, and let $X_1, X_2, \ldots, X_r$ be the sets picked, each X_i is a set of K vertices.

Call G, a graph over N vertices, *R-bad* if for more than $1/3 + \delta$ of the X_is, we have that $T(G[X_i]) \neq \chi(G)$. Let us show that there exists a set R_0 such that no G is R_0-bad, and once we have R_0, we will use it to construct the desired T'.

Fix any $G \in \mathcal{G}$. For every i, let Y_i be the random variable indicating whether $T(G[X_i]) \neq \chi(G)$ or not. Since we know that for every i it holds that $E[Y_i] = Pr_{U_\Omega}[T(G[X_i]) \neq \chi(G)] \leq 1/3$, and since we pick each X_i independently, we apply the Chernoff inequality and bound the probability that G is R-bad by:

$$Pr_{R \in_R \Omega^r}\left[G \text{ is } R\text{-bad}\right] = Pr\left[\frac{1}{r}\sum_i Y_i > \frac{1}{3} + \delta\right] \leq Pr\left[\left|\frac{1}{r}\sum_i Y_i - E[Y_i]\right| > \delta\right] \leq 2e^{-\frac{1}{2}\delta^2 r}$$

Thus, for a random R, the probability that exists a R-bad graph $G \in \mathcal{G}$, is bounded from above, using the union bound, by $2^{\binom{N}{2}} \cdot 2e^{-\frac{\delta^2}{2}r}$. By setting $r = \frac{2}{\delta^2}N^2$ we have that this probability is strictly less than 1. We deduce that there exists some subset $R \subset \Omega$ of size $O((N/\delta)^2)$ such that for every G, it holds that $Pr_{U_R}[T(G[X]) \neq \chi(G)] \leq 1/3 + \delta$. Fix R_0 to be this subset.

We now define T', which depends on the fixed set R_0. Formally, T' acts as follows:

1. Selects a set $X \in R_0$ randomly and uniformly.
2. Queries $G[X]$.
3. Replies as T would reply when viewing the graph $G[X]$.

Then T' is the required tester: its query complexity is just like the query complexity of T, its randomness complexity is $\log(|R_0|) = 2n + 2\log(1/\delta) + O(1)$, and its error probability is $Pr_{U_{R_0}}[T(G[X]) \neq \chi(G)] \leq 1/3 + \delta$. $\square$

Note that we can set the value of δ to be some constant strictly smaller than 1/6. This yields a tester T' which has randomness complexity of $2n + O(1)$, and error probability bounded by $1/3 + \delta < 1/2$. (For example, if we take $\delta = 1/9$, then we have randomness complexity of $2n + 8$ and error probability of at most 4/9.)

We can view Lemma 3.4 as a statement of the following type: For every graph property Π, and every canonical tester T for Π, exists some set R such that some condition holds. Indeed, Lemma 3.4 fits the framework of this statement. First, we are given T, and then, we find the set R_0, to construct a tester T', whose randomness complexity is smaller than the randomness complexity of T. In the proof of Lemma 3.4, the choice of the set R_0 was based on T. It may very well be the case, that for one tester we pick a certain set R_0, and for another tester we pick a different set R_0.

We wish to prove something stronger. We wish to fix *in advance* one set U that will enable us to reduce the randomness complexity of every property tester T. That is to say that first we pick U, and then we are given a property tester T for some graph property Π, and we want to use U in order to construct some tester T', similarly to the construction shown in the proof of Lemma 3.4. That is, we wish to prove a statement of the type: There exists a set U, such that for *every* graph property Π and *every* canonical tester T for Π, some condition (the same condition as before) holds. As we show, this statement can be proven.

We begin with definitions. Fix N, K. Denote by $\mathcal{G}_N$ the set of all graphs over N vertices. For any $\epsilon > 0$, and any Π, a property of graphs in $\mathcal{G}_N$, we say Π is *K-testable* for ϵ by T, if T is a canonical tester for Π and T randomly and uniformly selects a subset of K vertices in V and queries the subgraph over these K vertices. As before, denote $\Omega = \binom{V}{K}$, the set of all possible subsets of vertices of size K.

Definition 3.5. *Let $\delta > 0$. We call U, a subset of Ω, a (K,δ)-*universal *set if for any Π, a property for graphs in $\mathcal{G}_N$, any $\epsilon > 0$ and any canonical tester T such that Π is K-testable for ϵ by T, then there exists a tester T' that does the following:*

1. *Selects uniformly a set $X \in U$.*

2. *Queries $G[X]$.*

3. *Replies as T replies when viewing $(G[X])$.*

and has error probability at most $1/3 + \delta$.

A universal set U means the following. We have a fixed U. We then take *any* property Π and *any* canonical tester T such that Π is K-testable (for some $\epsilon > 0$) by T. We now restrict

T to pick a subset of K vertices, uniformly from U, and *not from all* Ω. Then, the definition of U guarantees us that the error probability of this restricted version of T does not exceed $1/3 + \delta$.

For instance, for any K, the set $\Omega = \binom{V}{K}$, containing all possible sets of K vertices, is $(K, 0)$-universal, by definition. Note that Ω is of size approximately N^K. We show, that a significantly smaller (K, δ)-universal set exists.

Theorem 3.6. *Fix N, K, $0 < \delta < 1/6$. Then, exists a (K, δ)-universal set U of size $O\left(2^{K^2} N^2/\delta^2\right)$.*

At first glance, it seems that Theorem 3.6 has to be false. The number of all possible graph properties can be huge. Even if we restrict ourselves to graph properties that are testable, then it still can be very high. It seems highly unlikely that a set of size roughly $N^2 \ll N^K$ can be universal, since a universal set means the same set is good for all graph properties. However, we exploit the fact that there are at most $2^{\binom{K}{2}}$ possible canonical testers that query a subgraph over K vertices.[9] Once we focus on the fact that U has to be good for all property testers, then Theorem 3.6 seems much more reasonable.

Recall that we denote $N = 2^n$. Theorem 3.6 means, that for every property Π which for some ϵ is K-testable, there exists a tester T' with randomness complexity of $2n + K^2 + 2\log(1/\delta) + O(1)$, while the straightforward canonical tester for Π has randomness complexity of $K \cdot n$. When K does not depend on n, or when K is relatively small, this gives a great decrease in randomness.

Proof. Again, we use the probabilistic method to show that a set of certain properties exists. Given that set, we show that its properties yield the required universal set. Denote by $\mathcal{G}_K$ the set of all graphs over K vertices.

Proposition 3.7. *There exists a set $U \subset \Omega$ of size $O\left(2^{K^2} N^2/\delta^2\right)$, with the following property: For every $G \in \mathcal{G}_N$ and every $H \in \mathcal{G}_K$ we have that*

$$|Pr_{X \in_R \Omega}[G[X] = H] - Pr_{X \in_R U}[G[X] = H]| < \delta/|\mathcal{G}_K|$$

That is to say, that for every U, any graph G over N vertices, and any graph H over K vertices, we look at two probabilities: The first, p_1, is the probability that when selecting a set from all possible sets of K vertices, the subgraph of G over the set picked is H. The second, p_2, is that probability that when selecting a set of K vertices uniformly from U, the subgraph of G over the set picked is H. The proposition claims that exists a set U that for any such G

[9] We explain the difference between the high number of graph properties to the relatively small number of property testers, by the fact that the same tester may be good for several graph properties, and also by the fact that many properties cannot be tested by a tester with query complexity $\binom{K}{2}$.

and any such H, we have that p_1 and p_2 are extremely close: $|p_1 - p_2| < \frac{\delta}{|\mathcal{G}_K|} = \delta / \exp\binom{K}{2}$. This property of U, suffices for us to show that U is (K, δ)-universal.

Proof (of Proposition 3.7). Similar to the proof of Lemma 3.4. Fix $u = |U| = O\left(2^{K^2} N^2/\delta^2\right)$ and select U by randomly, *independently* and uniformly selecting u elements from Ω. Denote those elements as $X_1, X_2, \ldots, X_u$.

Fix any G, H. Let Y_i be the random variable indicating whether $G[X_i] = H$. Denote $\rho = E[Y_i] = Pr_{X \in_R \Omega}[G[X] = H]$, and also denote $Y = \frac{1}{u}\sum_{i=1}^{u} Y_i$. Let us bound the probability that $|\rho - Y| > \delta/|\mathcal{G}_K|$, using the independence of the X_is and applying the Chernoff inequality:

$$Pr\left[|Y - \rho| > \delta/|\mathcal{G}_K|\right] = Pr\left[\left|\sum_i Y_i - u\rho\right| > u \cdot \delta/|\mathcal{G}_K|\right] \leq 2\exp\left(-\frac{\delta^2}{2|\mathcal{G}_K|^2} u\right)$$

Since we set $u = 4 \cdot 2^{K^2} N^2/\delta^2$, and since $|\mathcal{G}_K| \leq 2^{\frac{1}{2}(K^2-K)}$ we get that this probability is upper bounded by $2\exp(-N^2 2^K)$.

Applying the union bound, and the fact that $|\mathcal{G}_N| \leq 2^{\frac{1}{2}N^2}, |\mathcal{G}_K| \leq 2^{\frac{1}{2}K^2}$ we have that for any $N, K \geq 4$, it holds that

$$Pr\left[\exists G, H \text{ s.t. } |\rho - Y| > \delta/|\mathcal{G}_K|\,\right] \leq |\mathcal{G}_N||\mathcal{G}_K| \exp(-N^2 2^K) \leq 2^{\frac{1}{2}(N^2+K^2)} \cdot 2e^{-N^2 2^K} < 1$$

We deduce that $\exists U \subset \Omega$ of size $4 \cdot 2^{K^2} N^2/\delta^2$, with the required property. $\square$

Continuing with the proof of Theorem 3.6, we show that the set U from Proposition 3.7 is the alleged (K, δ)-universal set. Let Π ,$\epsilon > 0$ and T be such that Π is K-testable for ϵ by T. As before, let $\chi_\Pi : \mathcal{G}_n \to \{0, 1\}$ be the indicating function of Π. Meaning, for every $G \in \Pi$ we have that $\chi_\Pi(G) = 1$, and for every G which is ϵ-far from Π we have that $\chi_\Pi(G) = 0$.[10] Let $G \in \mathcal{G}_N$ be any graph which either has Π or it is ϵ-far from every graph that has Π.

Denote by M_T the set $\{H \in \mathcal{G}_k;\ T(H) \neq \chi_\Pi(G)\}$. I.e. M_T is the set of all graphs $H \in \mathcal{G}_K$ that if T finds that $G[X] = H$, it outputs the wrong answer (rejects G if $G \in \Pi$, or accepts G if G is ϵ-far from Π). Thus, $Pr[T \text{ errs}] = Pr[T \text{ views a graph from } M_T]$. Since T is the canonical tester that picks a set of K vertices uniformly from Ω, then we have:

$$Pr[T \text{ errs}] = Pr_{X \in_R \Omega}[G[X] \in M_T]$$

Define T' as the tester from the definition of (K, δ)-universal S. Since T' replies as T replies, we have that $Pr[T' \text{ errs}] = Pr[T' \text{ views a graph from } M_T]$. Since T' picks a set

[10]For any $G \in \mathcal{G}_N$ that is neither in Π nor is ϵ-far from Π, we care not what value χ_Π gives G.

uniformly from U, then we have:

$$Pr[T' \text{ errs}] = Pr_{X \in_R U}[G[X] \in M_T]$$

Given the property of U, we deduce that

$$\left|Pr[T' \text{ errs}] - Pr[T \text{ errs}]\right| = |Pr_{X \in_R U}[G[X] \in M_T] - Pr_{X \in_R \Omega}[G[X] \in M_T]| \leq$$
$$\sum_{H \in M_T} |Pr_{X \in_R U}[G[X] = H] - Pr_{X \in_R \Omega}[G[X] = H]| \leq |M_T| \frac{\delta}{|\mathcal{G}_K|} \leq \delta$$

This completes the proof of Theorem 3.6 $\square$

Note, as before, that we can fix δ to be some small constant, and obtain a universal set of size $O(2^{K^2}N^2)$. This means that we have a tester with query complexity of $\binom{K}{2}$, randomness complexity of $2n + K^2 + O(1)$, and error probability bounded from 1/2.

We conjecture that one can find an explicit way to construct a universal set U of size $N^c \exp(O(K^2))$ for some constant c (preferably 2). We consider it as a very interesting open problem.

3.2 Reducing the Randomness Complexity of Property Testers for Bounded Degree Graphs

In the following section, we discuss several property testers for graphs in the bounded-degree model, and show how to decrease their randomness complexity. We assume they all get as input some graph G, over N vertices, with degree bound d. For each testable property, we state the current known tester, how to reduce its randomness complexity, and how it affects its query, randomness and Q-R complexity. Note that unlike our the bipartiteness tester for the matrix adjacency model, we do not modify the testers discussed here. We do nothing more than simply "plug in" our hitters and samplers, instead of using total independence. This simple technique produces a significant decrease in the randomness complexity of these testers, yet keeps their original query complexity. As a result, we get a significant decrease in the Q-R complexity of these testers. We refer to Section 3.2.6 for a quantitative summary of the results. We note that in the majority of the cases here, we use a hitter or a sampler linear in confidence.

3.2.1 Testing Bipartiteness

We begin with stating the original algorithm which was proved in [GR98] to be a one-sided tester for bipartiteness.

Algorithm 1 Bipartiteness Tester

1: Pick $v_1, v_2, \ldots, v_s$ vertices, randomly, uniformly and independently, where $s = \Theta(1/\epsilon)$.
2: For every v_i, perform $K = O\left(\frac{\log^{1/2}(N/\epsilon)}{\epsilon^3}\right)\sqrt{N}$ independent random walks, each consists of $L = O\left(\frac{\log^6(N/\epsilon)}{\epsilon^8}\right)$ independent steps. If those random walks found cycle of odd length, reject G.
3: If no odd-cycle was found for any v_i, accept G.

We indicate that the analysis of [GR98] refers to random walks that aren't standard random walks. They discuss random walks in which each step is taken in the following manner: for every v in the walk and every $u \in \Gamma(v)$, with probability $(\frac{1}{2d})$ we go from v to u, and with probability $1 - \frac{|\Gamma(v)|}{2d}$ we remain in v.

Algorithm 1 has:

Query Complexity: $s(1 + K \cdot L) = \tilde{O}(\epsilon^{-12} \cdot \sqrt{N})$.

Randomness Complexity: $s \cdot \lg N + K \cdot L \cdot \lg(2d) = \tilde{O}(\epsilon^{-11} \cdot \sqrt{N})$.

Q-R Complexity: $\tilde{O}(\epsilon^{-23} \cdot N)$.

Note that the Q-R complexity is greater than N, and so it is crucial that we reduce the randomness complexity of this tester. We would like to add that Goldreich and Ron [GR02] gave a lower bound of $\Omega(\sqrt{N})$ on the query complexity of any one-sided bipartiteness tester for bounded degree graphs. Therefore, it is not possible to reduce (significantly) the query complexity of the tester. Thus, in order to reduce the Q-R complexity of the tester, we reduce its randomness complexity.

Due to the fact that the analysis of this tester is quite complicated, we modify this algorithm in two steps. The first step gives a one-sided testing algorithm whose error probability is slightly smaller than 1, yet by a noticeable gap. The second step is to amplify the success probability of the first tester, in order to achieve a tester with error probability at most 1/3. We begin with the weaker version of the tester.

Algorithm 2 Modified Bipratiteness Tester, Weak Version

1: Pick a vertex u randomly and uniformly from V.
2: Let $\mathcal{L}$ be the set of all possible random walks of length $L = O\left(\frac{\log^6(N/\epsilon)}{\epsilon^8}\right)$. Pick $K = O\left(\frac{\log^{1/2}(N/\epsilon)}{\epsilon^3}\right)\sqrt{N}$ walks out of $\mathcal{L}$ in a 4-wise independent manner.
3: Perform the K random walks leaving v. If these walks close some odd-cycle, reject G. Otherwise, accept G.

Lemma 3.8. *Algorithm 2 accepts every G which is bipartite, and rejects any G which is ϵ-far from bipartite, with probability at least $\frac{\epsilon}{160}$.*

Lemma 3.8 is proven at a later stage. We first consider the complexity of Algorithm 2.

The set $\mathcal{L}$ is of size $(2d)^L$, since in every step we just need to pick one neighbor of the current vertex. Therefore, we can pick one element in $\mathcal{L}$ using $L\cdot\log(2d) = O\left(\frac{\log^6(N/\epsilon)}{\epsilon^8}\right)$ random bits, and we can pick K elements in a 4-wise independent manner using $4\cdot L\cdot\log(2d) = O\left(\frac{\log^6(N/\epsilon)}{\epsilon^8}\right)$ random bits. Therefore, the randomness complexity of Algorithm 2 is $\log N + 4\cdot L\cdot\log(2d) = O\left(\frac{\log^6(N/\epsilon)}{\epsilon^8}\right)$. Therefore, we have that Algorithm 2 is of:

Query Complexity: $K\cdot L = \tilde{O}(\epsilon^{-11}\cdot\sqrt{N})$.

Randomness Complexity: $O\left(\frac{\log^6(N/\epsilon)}{\epsilon^8}\right)$.

Q-R complexity: $\tilde{O}(\epsilon^{-19}\cdot\sqrt{N})$.

Assuming Lemma 3.8 holds, we can amplify Algorithm 2, using a hitter linear in confidence.

Algorithm 3 Modified Bipariteness Tester, Strong Version

1: Let R be the set of all possible coin tosses for Algorithm 2.
2: Pick a set $S\subset R$, using a hitter linear in confidence, for sets of density at least $\frac{\epsilon}{160}$, and confidence $2/3$.
3: For every $\omega\in S$, simulate Algorithm 2 with ω as its random string.
4: If in one of the simulation G was rejected, reject G. Otherwise, accept G.

Theorem 3.9. *Algorithm 3 is a one-sided tester for bipartiteness.*

Proof of Theorem 3.9. Let G be a bounded degree graph over N vertices. If G is bipartite, then we know Algorithm 2 always accepts G, and therefore, no matter what S Algorithm 3 picks, in all simulations G is accepted, and so Algorithm 3 also accepts G.

If G is ϵ-far from bipartiteness, then we know that there exists some subset $T\subset R$, of density at least $\frac{\epsilon}{160}$, such that for every $\omega\in T$, Algorithm 2 rejects T when given ω as a random input. By the definition of the hitter of Step 2, we will hit a string from T with probability at least $2/3$. Therefore, with probability at least $2/3$ in at least one of the simulations G is rejected, causing Algorithm 3 to reject G. $\square$

We wish to note that Algorithm 3, which simulates Algorithm 2 many times in order to reduce the error probability of Algorithm 2, is a specific case of a general error amplification scheme, as shown in Appendix A. Using this scheme, we can perform many simulations of Algorithm 2, using the same number of random bits as Algorithm 2 uses.

In Step 2 of Algorithm 3, we use a hitter linear in confidence. This means that the number of samplers it requires is $O(\frac{1}{\epsilon})$, and it uses $\log(|R|)$ random bits. Denote the randomness complexity of Algorithm 2 by r, and we have that $\log(|R|) = r = O\left(\frac{\log^6(N/\epsilon)}{\epsilon^8}\right)$. Note that for every sample in S we simulate Algorithm 2, which means that the total query complexity is $\tilde{O}(\frac{1}{\epsilon}\cdot\frac{1}{\epsilon^{11}}\cdot\sqrt{N}) = \tilde{O}(\epsilon^{-12}\cdot\sqrt{N})$. Therefore, Algorithm 3 is of:

Query Complexity: $\tilde{O}(\epsilon^{-12}\cdot\sqrt{N})$.

Randomness Complexity: $O\left(\frac{\log^6(N/\epsilon)}{\epsilon^8}\right)$.

Q-R Complexity: $\tilde{O}(\epsilon^{-20} \cdot \sqrt{N})$.

Note that we have not changed the query complexity of the tester, and it is the same as in the original tester. The only difference is in the dramatic decrease in the randomness complexity of the tester, which leads to a decrease in the Q-R complexity.

All is left is for us to prove that indeed Algorithm 2 has a non-negligible probability of success.

Proof of Lemma 3.8. If G is bipartite, no odd-length cycles exist in G, so the modified tester finds no odd-length cycles, and must accept G.

So from now on, assume G is ϵ-far from being bipartite. We show that the proof of [GR98] still holds, despite the modifications made. The proof of [GR98] is fairly complicated. They define, for every G, every $H \subset V$, and any two intergers l_1, l_2, a Markov chain $M_{l_1}^{l_2}(H)$ that simulates a random walk of length $l_1 \cdot l_2$ over $G[H]$, that whenever it leaves H, it takes at least l_2 steps before returning to H.

For any single random walk, each step is independent of the rest of the steps, so all lemmas in the proof of [GR98] regarding the properties of such a walk hold.

Only a single lemma (Lemma 4.5 in [GR98]) refers to the behavior of a set of walks rather than to an individual walk. Specifically, this lemma lower bounds the probability that K random walks succeed in finding an odd-length cycle (given some properties of $M_{l_1}^{l_2}(H)$). For any two walks, i, j, denote by $\eta_{i,j}$ the random variable indicating whether some vertex $v \in H$ was met both in the ith walk and in the jth walk, while the lengths of the paths to v in both walks are of different parity (i.e. one walk corresponds to an even-length path, and the other walk corresponds to an odd-length path). In Lemma 4.5 of [GR98], the Chebyshev's inequality is used to upper bound the probability $Pr\left[\sum_{i<j} n_{i,j} = 0\right]$, meaning the probability that in none of the walks, no such v was found.

Note that not all pairs of $\eta_{i,j}, \eta_{k,l}$ are independent. For example $\eta_{i,j}$ and $\eta_{i,k}$ are very much dependent (the same walk affects them both). Nevertheless, one can bound the variance $var\left(\sum_{i<j} \eta_{i,j}\right)$. In fact, most of the proof of Lemma 4.5 of [GR98] deals with bounding this variance, in order to apply Chebyshev's inequality.

We note that the original argument holds also if the walks are 4-wise independent, rather than taken totally independent.

The original argument is based on the following two observations:

- For every i, j, l, k the random variables $\eta_{i,j}$ and $\eta_{l,k}$ are identically distributed.
- For every 4 distinct i, j, k, l, we have that $\eta_{i,j}$ and $\eta_{l,k}$ are independent.

Using these observations, they give bounds for the first and second moments of $\sum_{i<j} \eta_{i,j}$, based on bounds for the first and second moments of $\eta_{1,2} \cdot \eta_{1,2}$ and $\eta_{1,2} \cdot \eta_{1,3}$.

Therefore, it is easy to see that by taking the walks to be 4-wise independent rather than totally independent, both observations still hold. Furthermore, the analysis of the first and second moments for $\eta_{1,2} \cdot \eta_{1,2}$ and $\eta_{1,2} \cdot \eta_{1,3}$ still holds, since the 4-wise independence of the walks means also that every 2 or 3 walks are independent of one another. We conclude that Lemma 4.5 of [GR98] holds, for the modified tester.

The rest of the analysis of [GR98] applies here. Let us elaborate. The final step in the proof of [GR98] introduces the definition of a *bad* vertices - vertices for which such random walks will find an odd-length cycle with probability at least 0.1. It is proven in [GR98] that there must be at least $\frac{\epsilon}{16}N$ bad vertices in G, for otherwise, one can find a partition of G with less than ϵdN violating edges. This proof is based on Lemma 4.5, and the analysis of the success probability of the K random-walks to find an odd-length cycle. As we have shown just now, performing these K random walks in a 4-wise independent manner, gives the same success probability on finding an odd-length cycle. Therefore, there must be also at least $\frac{\epsilon}{16}N$ vertices, which are bad with respect to the random walks of Algorithm 2. Meaning, that by taking the K random walks in a 4-wise independent manner, we have that they find an odd-length cycle with probability at least 0.1.

Therefore, the probability that Algorithm 2 picks a bad vertex as its starting point is at least $\frac{\epsilon}{16}$. Given that the starting vertex is bad, we know that Algorithm 2 has probability of at least 0.1 to find an odd-length cycle. Therefore, we deduce that with probability at least $\frac{\epsilon}{16} \cdot \frac{1}{10} = \frac{\epsilon}{160}$, Algorithm 2 finds an odd-length cycle, and rejects G. □

3.2.2 Testing Connectivity

We begin with stating the original algorithm, which was proven in [GR02] to be a one-sided tester for connectivity.

Algorithm 4 Connectivity Tester

for $i = 1, 2, ..., l = \log(8/\epsilon d)$ **do**
  Select $m_i = \frac{32\log(8/\epsilon d)}{2^i \epsilon d}$ vertices randomly, uniformly and independently.
  For each vertex v picked, do a BFS starting from v until 2^i vertices are found. If a component of size at most 2^i is found then reject G.
end for
If for all i no component of size at most 2^i was found, then accept G.

Algorithm 4 has:
Query complexity: $\sum_{i=1}^{l} \frac{32\log(8/\epsilon d)}{2^i \epsilon d} 2^i = O(l \frac{\log(1/\epsilon d)}{\epsilon d}) = O(\frac{\log^2(1/\epsilon d)}{\epsilon d})$
Randomness complexity: $\sum_{i=1}^{l} \log N \frac{32\log(8/\epsilon d)}{2^i \epsilon d} = O(\frac{\log(1/\epsilon d)}{\epsilon d} \log N)$
Q-R complexity: $O(\frac{\log^3(1/\epsilon d)}{(\epsilon d)^2} \log N)$.
We now introduce a modified version of Algorithm 4, with better randomness complexity.

Algorithm 5 Modified Connectivity Tester

For $i = 1, 2, \ldots, l = \log(8/\epsilon d)$, let S_i be a hitter linear in confidence, for sets of density at least $\frac{2^i \epsilon d}{16l}$, and confidence at least $2/3$.
Let R_i be the set of all possible random inputs for S_i. Since all S_i are linear in confidence, we may use $R_1 = R_2 = \ldots = R_l = R$.
Pick randomly and uniformly a string $\omega \in R$.
for $i = 1, 2, \ldots, l$ **do**
  Use S_i and ω to produce a set of m_i vertices.
  For each vertex v in this set, do a BFS starting from v until 2^i vertices were found. If a component of size at most 2^i is found then reject G.
end for
If for all i no component of size at most 2^i was found, then accept G.

We wish to note the use of several hitters, which all use the same random string. As we show, the analysis is based on the success probability of some single iteration (i.e. for a certain i). We do not know a-priori for which of the l values we wish for this hitter to succeed. But that does not change the fact that we only need it to succeed in one iteration. The fact that we could use the same coins for all hitters is due to the fact that we can construct hitters and samplers with the same randomness complexity, regardless to the value of ϵ.

Claim 3.10. *Algorithm 5 is a one-sided tester for connectivity.*

Proof. If G is connected then a single connected component of size N exists, and so the tester accepts G. If G is ϵ-far from being connected, then, as it was shown in [GR02], the number of connected components of size at most $\frac{8}{\epsilon d}$ in G is at least $\frac{\epsilon}{8} dN$. Denote by B_i the number of connected components which are of size t for some t that belongs to the interval $[2^{i-1}, 2^i)$. Clearly, $\sum_{i=1}^{l} |B_i| \geq \frac{\epsilon}{8} dN$. Therefore, there exists i_0 such that $|B_{i_0}| \geq \frac{\epsilon}{8l} dN$. For every i, denote by T_i the set of vertices belonging to the connected components that are in B_i. Thus, $|T_{i_0}| \geq 2^{i_0 - 1} \cdot |B_{i_0}| = (2^{i_0} \epsilon d N)/16l$.

The probability that for this i_0, the hitter S_{i_0} fails to hit a vertex from T_{i_0} is at most $1/3$. If a vertex that belongs to T_{i_0} is hit, then the BFS starting from vertex finds a connected component of size smaller than 2^{i_0}, and thus the tester rejects G. Thus, with probability at least $2/3$, the tester succeeds in finding a small connected component, and rejects G. ☐

This modification yields an algorithm of the following complexity:
Query complexity: $\sum_{i=1}^{l} m_i 2^i = O\left(\frac{\log^2(1/\epsilon d)}{\epsilon d}\right) = \tilde{O}\left(\frac{1}{\epsilon d}\right)$.
Randomness complexity: $\log N$.
Q-R complexity: $O\left(\frac{\log^2(1/\epsilon d)}{\epsilon d}\right) \cdot \log N = \tilde{O}\left(\frac{1}{\epsilon d}\right) \cdot \log N$.

Yet again we note that the query complexity of the original tester, and the query complexity of the modified tester, are identical. The change in the Q-R complexity is all due to the decrease in the randomness complexity.

3.2.3 Testing Cycle-Freeness

Before discussing the tester for cycle freeness, we introduce the following notation. A connected component C in G, is called *big* if its size is greater than $\frac{8}{\epsilon d}$, and *small* if its size is at most $\frac{8}{\epsilon d}$.

As always, we first state the original algorithm, which was proven in [GR02] to be a two-sided tester for cycle-freeness.

Algorithm 6 Cycle-Freeness Tester

1: Select randomly, uniformly and independently a sample S of $l = O(\frac{1}{\epsilon^2})$ vertices.
2: For each vertex selected, perform a BFS to check whether it belong to a small or big component. If any small connected component containing a cycle is found, reject G.
3: Denote by S_b the set of vertices in S that belong to big components, and denote $l_b = |S_b|$. Denote also by e_b half the sum of their degree. ($e_b = \frac{1}{2}\sum_{v \in S_b} deg(v)$). If $\frac{e_b - l_b}{l} \geq \frac{\epsilon d}{16}$ reject G, otherwise, accept G.

Algorithm 6 has:
Query complexity: $O(\frac{1}{\epsilon^2} \cdot \frac{1}{\epsilon d}) = O(\frac{1}{\epsilon^3})$.
Randomness complexity: $O(\frac{1}{\epsilon^2} \log N)$.
Q-R complexity: $O(\frac{1}{\epsilon^5 d} \log N)$.

We now introduce a modified version of Algorithm 6, with better randomness complexity.

Algorithm 7 Modified Cycle-Freeness Tester

1: Select S using a sampler logarithmic in confidence, with accuracy $\frac{\epsilon}{16d}$ and confidence at least $1 - \frac{1}{3(d+2)}$.
2: Perform Steps 2 and 3 as in the original Algorithm 6.

Claim 3.11. *Algorithm 7 is a 2-sided tester for cycle-freeness.*

Proof. A known fact is that a tree over t vertices contains $t-1$ edges. Therefore, if G is a cycle-free graph, and we denote its connected components as $C_1, C_2, \ldots, C_k$, then we have that the number of edges in G is $\sum_{i=1}^{k}(|C_i| - 1) = N - k$. However, if G is ϵ-far from cycle-freeness, then at least $\frac{1}{2}\epsilon dN$ edges, which we call *superfluous edges*, need to be removed from G in order to have a cycle-free graph. That means that the total number of edges in G is at least $N + \frac{1}{2}\epsilon dN - k$. Therefore, the basic notion of the tester is to estimate the number of edges in G. Recall that the number of edges in a graph is $|E| = \frac{1}{2}\sum_{v \in V} deg(v)$, so we can estimate that number of edges in G by estimating the average degree of the vertices in G.

However, the number of edges in both cases depends on k, the number of connected components in G, which can be as high as $O(N)$. Therefore, Goldreich and Ron [GR02] treat big and small components differently. They only estimate the number of vertices and edges that reside in big components. Small components are dealt by traversing all vertices in them, and seeking a cycle.

Suppose we only had to test for graphs which are either cycle-free or that they have many cycles in *small components*. That means, that a lot of cycles belong to components with no more than $\frac{8}{\epsilon d}$ vertices. This is easily solvable by repeating the following procedue: Pick a vertex at random and perform a BFS of no more than $\frac{8}{\epsilon d}$ vertices. If G is cycle free, this procedure never finds a cycle, and if G is a graph with many cycles, we are likely to find a cycle, if we repeat this procedure a sufficient number of times.

Suppose the opposite case, in which we have to test for graphs which are either cycle-free, or that have many cycles in *big components*. Note that we can upper bound the number of big connected components by $\frac{N}{8/\epsilon d} = \frac{1}{8}\epsilon dN$. This time, we estimate the number of edges in the big components, by estimating the average degree of vertices that belong to the big components of G. If we have that too many edges belong to big components, we can reject G, since that means that some cycle in these components exists.

Since we know that G has at least $\frac{1}{2}\epsilon dN$ superfluous edges, then one of two must hold: either G has at least $\frac{1}{4}\epsilon dN$ superfluous edges that belong to small connected components, or at least $\frac{1}{4}\epsilon dN$ superfluous edges belong to big connected components. We therefore test for both cases. If G has at least $\frac{1}{4}\epsilon dN$ edges that belong to small connected components, then we need a sample set S which is likely to hit a vertex in the set $C = \{v \in V;\ v$ belongs to a small component$\}$. If G has at least $\frac{1}{4}\epsilon dN$ edges belong to big connected components, then we need S to give a good estimation on the following $d+1$ sets:
$B_0 = \{v \in V\ ; v$ belongs to a big component$\}$,
$B_j = \{v \in V\ ; v$ belongs to a big component, and $deg(v) = j\}$ for $j = 1, 2, \ldots, d$.

The point is that we can use a *single* sampler that gives a good estimation for all $d+2$ sets. For any set $A \in \{C, B_0, B_1, B_2, \ldots, B_d\}$, a sampler of accuracy of $\epsilon/16d$ and confidence at least $1 - \frac{1}{3(d+2)}$ produces an estimation which is $\frac{\epsilon}{16d}$-far from the density of A, with probability at most $\frac{1}{3(d+2)}$. Using the union bound, the probability that *exists* some set $A \in \{C, B_0, B_1, \ldots, B_d\}$ for which the sampler gives an estimation which is $\frac{\epsilon}{16d}$-far from the density of A, is at most $(d+2)\frac{1}{3(d+2)} = \frac{1}{3}$. We have that with probability at least $2/3$, the sample S allows us to estimate the densities of all sets in $\{C, B_0, B_1, B_2, \ldots, B_d\}$, up to an accuracy factor of $\frac{\epsilon}{16d}$.

So now, let us prove the correctness of the tester. Denote by t_b the number of big components of G. Denote by N' the number of vertices that belong to big components of G and by M' the number of edges in these components. For any $i \in \{1, 2, \ldots, l\}$, denote by χ_i the indicator random variable which equals 1 if and only if the ith vertex of S is in a big component. For any $i \in \{1, 2, \ldots, l\}$ and any $j \in \{1, 2, \ldots, d\}$, denote by $\psi_{i,j}$ the indicator random variable which equals 1 if any only if the degree of the ith vertex is j, and the ith vertex belongs to a big component.

This notation means that for l_b, the number of a vertices in S that belong to big components, it holds that $l_b = \sum_{i=1}^{l} \chi_i$. Similarly, for $e_b = \frac{1}{2}\sum_{v\in S_b} deg(v)$, it holds that $e_b = \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{d} j\cdot\psi_{i,j}$. Also, $E[\frac{l_b}{l}] = E[\chi_i] = \frac{N'}{N}$, and $E[\frac{e_b}{l}] = \frac{M'}{N}$.

Now, if G is cycle free, Step 2 does not find a cycle in G. Let us assume that for every set in $\{C, B_0, B_1, \ldots, B_d\}$, the sampler gave an estimation which is $\frac{\epsilon}{16d}$-close to its density. Since $d \geq 2$ we have that $\frac{l_b}{l} = \frac{1}{l}\sum_{i=1}^{l}\chi_i$ is $\frac{\epsilon}{32}$-close to $E[\frac{l_b}{l}] = \frac{N'}{N}$. Also, for every j, we have that we have that $\frac{1}{l}\sum_i \psi_{i,j}$ is $\frac{\epsilon}{16d}$-close to $E[\frac{j\cdot|B_j|}{N}]$. That means that

$$\left|\frac{e_b}{l} - E\left[\frac{e_b}{l}\right]\right| = \left|\frac{1}{l}\cdot\frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{d} j\cdot\psi_{i,j} - \sum_{j=1}^{d} E\left[\frac{j\cdot\psi_{i,j}}{2l}\right]\right|$$
$$\leq \sum_{j=1}^{d}\frac{j}{2}\cdot\left|\frac{1}{l}\sum_i \psi_{i,j} - E\left[\psi_{i,j}\right]\right| \leq \frac{d^2}{2}\cdot\frac{\epsilon}{16d} \leq \frac{\epsilon d}{32}$$

and therefore
$$\left|\frac{e_b - l_b}{l} - \frac{M'-N'}{N}\right| \leq \left|\frac{e_b}{l} - \frac{M'}{N}\right| + \left|\frac{l_b}{l} - \frac{N'}{N}\right| \leq \frac{\epsilon d}{16}$$

Recall that G is cycle free, therefore, $M' < N'$ and so we have that $\frac{e_b - l_b}{l} < 0 + \frac{\epsilon d}{16} = \frac{\epsilon d}{16}$. Therefore, if S estimates all $d+2$ sets correctly, we have that the tester accepts G. Since with probability at least $2/3$ we have that all $d+2$ sets are correctly estimated, we deduce that Algorithm 7 accepts G with probability at least $2/3$.

If G is ϵ-far from cycle-free, we have shown that either G has at least $\frac{1}{4}\epsilon dN$ superfluous edges that belong to small connected components, or at least $\frac{1}{4}\epsilon dN$ superfluous edges belong to big connected components.

In the first case - since any component with e edges must contain at least $\frac{2e}{d}$ vertices, then the total number of vertices in small components with at least one superfluous edge is at least $\frac{1}{2}\epsilon N$. Thus, with probability greater than $2/3$ the set S hits a vertex from such a small component, and so Step 2 rejects G, in this case.

If at least $(\epsilon d/4)N$ superfluous edges belong to big connected components, we have that $t_b \leq \frac{N}{8/\epsilon d} = \frac{\epsilon d}{8}N$, and so

$$\frac{M'-N'}{N} = \frac{M' - (N' - t_b) - t_b}{N} \geq \frac{(\epsilon d/4)N}{N} - \frac{1}{8}\epsilon d \geq \frac{\epsilon d}{8}$$

As before, with probability at least $2/3$ we have that $\left|\frac{e_b - l_b}{l} - \frac{M'-N'}{N}\right| \leq \frac{\epsilon d}{16}$, meaning that $\frac{e_b - l_b}{l} \geq \frac{\epsilon d}{8}$. Therefore, with probability at least $2/3$, Step 3 rejects G.

We have that in each case we reject G with probability at least $2/3$, and therefore, we have a tester for cycle-freeness. $\square$

We wish to note that we want a sampler of accuracy of $\epsilon/(16d)$ and confidence at least

$1 - \frac{1}{3(d+2)}$. By using a sampler logarithmic in confidence, we have that the set S of samples that the sampler produces, is of size $O(d^2 \cdot \epsilon^{-2} \log(1/d)) = O(\epsilon^{-2})$, and that the sampler requires $\log N + O(\log(3(d+2)))$ random bits. That means that Algorithm 7 has:

Query complexity (which remains the same as in Algorithm 6): $O(\frac{1}{\epsilon^3})$.

Randomness complexity: $\log N + O(\log(d)) = \log N + O(1)$.

Q-R complexity: $O(\epsilon^{-3} \log N)$.

3.2.4 Testing Subgraph Freeness

Fix some graph H, and denote $v_h = V(H), d_h = diam(H)$. We say a graph G is H-free if it contains no subgraph isomorphic to H.

We state the original algorithm, which was proven in [GR02] to be a one-sided tester for subgraph-freeness.

Algorithm 8 H-freeness Tester

1: Select randomly, uniformly and independently $m = O(\frac{1}{\epsilon})$ vertices.
2: For every v selected, perform a BFS starting from s that ends after depth d_h.
3: If found a subgraph isomorphic to H, Reject G. Otherwise, accept G.

Algorithm 8 has:

Query complexity: $O(\frac{1}{\epsilon} d^{d_h})$.

Randomness complexity: $O(\frac{1}{\epsilon} \log N)$.

Q-R complexity: $O(\epsilon^{-2} d^{d_h} \log N)$.

We now introduce a modified version of Algorithm 8, with better randomness complexity.

Algorithm 9 Modified H-freeness Tester

1: Select S using a hitter linear in confidence for sets of density at least ϵ and confidence at least $\frac{2}{3}$.
2: Perform Steps 2 and 3 as in the original Algorithm 8.

Claim 3.12. *Algorithm 9 is a one-sided tester for H-freeness.*

Proof. Clearly if G is H-free then this algorithm accepts G, since no matter what vertices we pick, the BFS will never find a subgraph isomorphic to H.

If G is ϵ-far from H-freeness, then at least $\frac{1}{2}\epsilon dN$ edges reside in copies of H, since every vertex is of degree at most d then at least $\frac{1}{d} \cdot 2 \cdot \left(\frac{1}{2}\epsilon dN\right) = \epsilon N$ vertices reside in these copies of H. Thus, with probability at least 2/3 the hitter will hit one, and the tester will reject G. □

This modification yields a tester which has:

Query complexity (which remains the same as in Algorithm 8): $O(\frac{1}{\epsilon} d^{d_h})$.

Randomness complexity: $\log N$.

Q-R complexity: $O(\epsilon^{-1} d^{d_h} \log N)$.

3.2.5 Testing if a Graph is Eulerian

Recall that a graph G is Eulerian if exists a path that traverses every edge of G exactly once. A well known fact is that G is Eulerian if and only if it is connected, and either all of its vertices are of even degree, or exactly two of them are of odd degree and the rest are of even degree.

We begin by stating the original algorithm, which was proven in [GR02] to be a tester for Eulerian graphs.

Algorithm 10 Eulerian Testing Algorithm

1: Check whether G is $\epsilon/2$-far from being connected, using the previous connectivity tester. If it rejects - reject G.
2: Select randomly, uniformly and independently $m = O(\frac{1}{\epsilon})$ vertices. Query for every vertex its neighbors.
3: If more than two vertices out of the m vertices are found to have odd-degree - reject G.
4: Otherwise - accept G.

Algorithm 10 has:

Query Complexity: $O(\frac{\log^2(1/\epsilon d)}{\epsilon})$.

Randomness Complexity: $O\left(\frac{\log(1/\epsilon d)}{\epsilon} + \frac{1}{\epsilon}\log N\right) = O\left(\frac{\log(1/\epsilon d)}{\epsilon}\log N\right)$.

Q-R Complexity: $O(\frac{\log^3(1/\epsilon d)}{\epsilon^2}\log N) = \tilde{O}(\epsilon^{-2}\log N)$.

We now introduce a modified version of Algorithm 10, with better randomness complexity. Recall that there exist Eulerian graphs G with two of their vertices having odd degree. That is why Algorithm 10 rejects its input only if 3 vertices of odd degree are found. If it were the case that Algorithm 10 rejects the input after seeing solely one odd-degree vertex, then it would be possible for the tester to reject some Eulerian graph, and therefore it wouldn't be a one-sided tester. As in all modifications done thus far, the randomness-efficient version of the tester that we show here, uses a hitter. This time, we cannot use a hitter in order to find vertices of odd-degree, because this may cause the tester to reject Eulerian graphs. Therefore, we must hit 3 vertices of odd-degree in order to reject the input. This is why our modified version uses a *3-hitter*, and not merely a 1-hitter, as used in all previous modifications.

Algorithm 11 Modified Eulerian Testing Algorithm

1: Check whether G is $\epsilon/2$-far from being connected, using the modified tester for connectivity, Algorithm 5. If it rejects, then reject G.
2: Use a 3-hitter, linear in confidence, for sets of density at least $\frac{\epsilon d}{16}$ and confidence at least $2/3$, to pick a sample S of $m = O(\frac{1}{\epsilon d})$ vertices.
3: If found distinct $v_1, v_2, v_3 \in S$ with odd degree, then reject G. Otherwise, accept G.

Claim 3.13. *Algorithm 11 is a one-sided Eulerian graph tester.*

Proof. Clearly, if G is Eulerian then it is connected and no distinct 3 vertices of G are of odd degree, so the algorithm accepts G.

Suppose G is ϵ-far from being Eulerian. The following lemma was proven by Goldreich and Ron.

Lemma 3.14 ([GR02]). *If G is ϵ-far from being Eulerian, then either it has at least $\frac{\epsilon}{8}dN$ connected components, or it has at least $\frac{\epsilon}{16}dN$ vertices of odd degree.*

Suppose G has at least $\frac{\epsilon}{8}dN$ connected components. Following the same logic from the proof of the tester for connectivity, at least half of them contain at most $\frac{16}{\epsilon d}$ vertices. This means that the modified connectivity tester, Algorithm 5 rejects G with probability at least 2/3.

Suppose G has at least $\frac{\epsilon}{16}dN$ vertices of odd degree. The 3-hitter of Step 2 hits at least 3 of the vertices of odd-degree with probability at least 2/3, and it rejects G.

We have that in any case, the tester rejects G with probability at least 2/3. □

We wish to note that naively, we have randomness complexity of $2\log N$. The first $\log N$ bits are used for the hitter of Algorithm 5 we simulate, and the latter $\log N$ bits are used for the 3-hitter used in Step 2. However, we can only use $\log N$ random bits by simply using the same random string for both Step 1 and Step 2, from the same reasoning which we used in the randomness complexity analysis of Algorithm 5.

Query complexity (which remains the same as in Algorithm 10): $O(\frac{\log^2(1/\epsilon d)}{\epsilon})$.

Randomness complexity: $\log N$.

Q-R complexity: $O(\frac{\log^2(1/\epsilon d)}{\epsilon}) \cdot \log N$.

3.2.6 Summation

We end by a sum-up of the tester we presented in this section.

	Query Complexity	Original Randomness Complexity	Modified Randomness Complexity	Q-R Complexity
Bipartiteness	$\tilde{O}(\epsilon^{-12}\sqrt{N})$	$\tilde{O}(\epsilon^{-11}\sqrt{N})$	$\tilde{O}(\epsilon^{-8}\log N)$	$\tilde{O}(\epsilon^{-20}\sqrt{N})$
Connectivity	$\tilde{O}(\epsilon^{-1})$	$\tilde{O}(\epsilon^{-1}) \cdot \log N$	$\log N$	$\tilde{O}(\epsilon^{-1}) \cdot \log N$
Cycle-Freeness	$O(\epsilon^{-3})$	$O(\epsilon^{-2}) \cdot \log N$	$\log N + O(1)$	$O(\epsilon^{-3}) \cdot \log N$
Subgraph-Freeness	$O(\epsilon^{-1} \cdot d^{d_h})$	$O(\epsilon^{-1}\log N)$	$\log N$	$O(\epsilon^{-1}d^{d_h}) \cdot \log N$
Eulerian	$\tilde{O}(\epsilon^{-1})$	$\tilde{O}(\epsilon^{-1}) \cdot \log N$	$\log N$	$\tilde{O}(\epsilon^{-1}) \cdot \log N$

4 Reducing the Q-R Complexity of Testing Bipartiteness in the Adjacency Matrix Model

We use testing bipartiteness as a test-case. We wish to modify the original [GGR98] tester in order to reduce its Q-R complexity. We first take the original tester and implement it using good hitters, and already improve its randomness and Q-R complexity. Then we modify the tester, to achieve even better results, using techniques which we believe that are implementable for several other testers. We refer to section 4.8 for a quantitative summary of all the results. Note that in order to reduce the Q-R complexity of the tester, we do not necessarily focus on achieving the smallest possible randomness complexity. The emphasis in this section is on reducing the randomness complexity *without* increasing the query complexity. As an intermediate step, we present testers that have randomness complexity that is close to optimal, yet have worse query complexity than the original tester. We also present testers that have the same query complexity as the original tester, yet have randomness complexity far from optimal. Eventually, in Section 4.7, we present a tester whose Q-R complexity is close to optimal.

Recall that a graph is considered to be *bipartite*, or *2-colorable*, if there exists $S \subset V$, so that for every two *adjacent* vertices, $u, v \in V$, either $u \in S$ and $v \in \overline{S} = V \setminus S$, or $v \in S$ and $u \in \overline{S}$. We call $\langle S, \overline{S} \rangle$ a 2-coloring of G, and we identify $\langle S, \overline{S} \rangle$ with $\langle \overline{S}, S \rangle$. For every vertex $u \in S$, we say that S is *side* (or the *color*) of u. A well known fact is that any graph is bipartite if and only if it contains no odd-length cycles.

4.1 The Original Tester

Recall our basic settings. We are given a graph $G = (V, E)$ as input, where $|V| = N = 2^n$. The graph is represented as a matrix of size $N \times N$, for which we can check for every pair of vertices $(v_1, v_2) \in V \times V$ whether $\langle v_1, v_2 \rangle \in E$ or not.

We begin with a review of the known tester, due to Goldreich, Goldwasser and Ron [GGR98]. The most simple tester is to select $O(\frac{1}{\epsilon^2})$ vertices at random (uniformly and independently), and to query the subgraph over these vertices. If the subgraph queried is bipartite - accept G, and if it contains an odd-length cycle - reject G. We however, go over the more elaborate version of their tester, which has lower query complexity. In order to do this, we first define the notion of a violating edge for a 2-partition of a subgraph of G:

Definition 4.1. *Given some $U \subset V(G)$, and a 2-partition of U, denoted as $\langle U_1, U_2 \rangle$, we say an edge $\langle u, v \rangle \in E(G)$* violates *this 2-partition, if for some $i \in \{1, 2\}$, both its endpoints belong to $\Gamma(U_i)$.*

A violating edge for a 2-coloring $\langle U_1, U_2 \rangle$ of U, gives witness to the fact that V have no 2-coloring in which all vertices in U_1 are given one color, all vertices in U_2 are given the other color. Had such a 2-coloring of V existed, then it would have given both endpoints of the violating edge the same color, which would have resulted in an immediate contradiction. Let us now state the original tester and prove its correctness.

Algorithm 12 GGR Bipartiteness Tester

1: Pick a set $U \subset V$ by picking randomly, uniformly and independently $\tilde{O}(\frac{1}{\epsilon})$ vertices of V. Query all of $O(|U|^2) = \tilde{O}(\frac{1}{\epsilon^2})$ vertex-pairs in $G[U]$.
2: Pick a set $A \subset V \times V$ by picking randomly, uniformly and independently $\tilde{O}(1/\epsilon^2)$ vertex-pairs in G. For every pair $(v_1, v_2) \in A$ query it to see whether the edge $\langle v_1, v_2 \rangle$ belongs to $E(G)$, as well as all potential edges between v_1 and U, and between v_2 and U.
3: If for every 2-coloring of U a violating edge is found - output *"not bipartite"*, otherwise output *"bipartite"*.

Theorem 4.2 ([GGR98]). *Algorithm 12 is a 1-sided bipartiteness tester, with query complexity $\tilde{O}(\frac{1}{\epsilon^3})$.*

Proof. We say that the set U is *good* if it dominates all but at most a fraction of $\epsilon/3$ of the vertices with a degree of least $(\epsilon/3)N$.

Our goal is to show that we are likely to pick a good U, and that if indeed G is ϵ-far from bipartiteness, then by going over all $2^{|U|}$ possible 2-partitions of U, we will be able to deduce that no 2-coloring of U exists. The analysis is based on the following claim:

Claim 4.3. *1. With probability at least $5/6$, the set U is good.*

2. If G is ϵ-far from being bipartite, and U is good, then for any 2-partition of U there are at least $(\epsilon/3)N^2$ edges in the graph that violate that partition.

We first show that Claim 4.3 implies the validity of Theorem 4.2. Clearly, if G is bipartite, then any 2-coloring of V induces a 2-coloring of U with no violating edges. Hence, U has (at least) one 2-coloring for which no violating edge will be found. So the tester outputs *"bipartite"*.

Let G be a graph ϵ-far from being bipartite. If U is good, then for every possible 2-partition of U, the probability that a uniformly chosen vertex-pairs does not violate the partition is at most $(1 - \epsilon/3)$. Thus, by picking $\tilde{O}(1/\epsilon^2)$ vertex-pairs *independently*, we have that the probability that a violating edge for a fixed 2-partition of U won't be found is at most $(1 - \epsilon/3)^{\tilde{\Omega}(\frac{1}{\epsilon^2})} \leq e^{-\frac{\epsilon}{3} \cdot \tilde{\Omega}(1/\epsilon)} = e^{-\tilde{\Omega}(1/\epsilon)}$. By the union bound over all $2^{|U|}$ possible 2-partitions of U, the probability that for some partition of U no violating edges is found is upper bounded by

$$2^{|U|} \cdot e^{-\tilde{\Omega}(1/\epsilon)} = 2^{\tilde{O}(1/\epsilon)} \cdot e^{-\tilde{\Omega}(1/\epsilon)} \leq 1/6$$

Thus, the probability that the tester doesn't reject G is upper bounded by $1/6+1/6 = 1/3$, when one term comes from the probability that U isn't good, and the other comes from the probability the for some 2-partition of U, a violating edge will not be found. Since the query complexity of the tester is $\tilde{O}\left(\frac{1}{\epsilon^2} + \frac{1}{\epsilon^2 \cdot \epsilon}\right) = \tilde{O}\left(\frac{1}{\epsilon^3}\right)$, we conclude that if indeed the claim holds then we are done. $\square$

Proof of the Claim.

1. Let v be a vertex in G with degree at least $(\epsilon/3)N$. Thus, by uniformly selecting a vertex $u \in V(G)$, we have that $Pr[u \text{ dominates } v] \geq \epsilon/3$. Therefore, by picking $t = \tilde{O}(\epsilon^{-1})$ vertices *independently*, the probability that none of the vertices dominates v is at most $(1 - \epsilon/3)^t \leq \epsilon/18$.
   Therefore, the expected fraction of vertices in G with degree at least $(\epsilon/3)N$ that aren't dominated by U, is at most $\epsilon/18$. By Markov inequality, the probability that a fraction of $\epsilon/3$ of these vertices aren't dominated by U is bounded above by $\frac{\epsilon/18}{\epsilon/3} = 1/6$.

2. Assume U is good, and G is ϵ-far from being bipartite. Fix some 2-coloring $\langle U_1, U_2 \rangle$ of U. This 2-coloring of U induces the following 2-coloring of $V(G)$, denoted $\Pi = \langle\ \Gamma(U_1),\ (V(G) \setminus \Gamma(U_1))\ \rangle$.

   Since G is ϵ-far from being bipartite then for every 2-coloring of $V(G)$ there are at least ϵN^2 *bad* edges, which are edges Π gives both their endpoints the same color. We show that (1) any bad edge of Π with both endpoints adjacent to U is a $\langle U_1, U_2 \rangle$-violating edge, and (2) there are many bad edges with both endpoints adjacent to U. Both facts combined give that there are many $\langle U_1, U_2 \rangle$ violating edges.

   Showing (1) is no more than "playing" with the definitions. Let $\langle u, v \rangle$ be a bad edge with both endpoints adjacent to U. As a bad edge both its endpoints belong to some side of Π. If $u, v \in \Gamma(U_1)$, then by definition is violates $\langle U_1, U_2 \rangle$. Otherwise $u, v \in V(G) \setminus \Gamma(U_1)$, then both u and v are adjacent to vertices in $U \setminus U_1 = U_2$, and so $\langle u, v \rangle$ is a $\langle U_1, U_2 \rangle$ violating edge.

   To prove (2) we suppose by contradiction that at most $\frac{\epsilon}{3}N^2$ bad edges have that both their endpoints are adjacent to U. We now upper bound the number of edges with at least one vertex not adjacent to U. Recall that U is good. In fact we shall upper bound a larger set of edges - the set of edges with one endpoint that isn't a vertex dominated by U and has degree at least $(\epsilon/3)N$:

   - There are at most N vertices with degree smaller than $(\epsilon/3)N$, and they "give" at most $\frac{\epsilon}{3}N^2$ edges.
   - Since U is good, there are at most $(\epsilon/3)N$ vertices with degree at least $(\epsilon/3)N$ that aren't dominated by U, and these vertices "give" at most $\frac{\epsilon}{3}N^2$ edges.

   This means Π has less than ϵN^2 bad edges, and so G is ϵ-close to bipartiteness. We have a contradiction, and deduce that at least $\frac{\epsilon}{3}N^2$ violating edges for $\langle U_1, U_2 \rangle$ exist.

$\square$

Total Complexity of Algorithm 12:

Query complexity: $\tilde{O}(\epsilon^{-3})$.

Randomness complexity: $\tilde{O}(\epsilon^{-1}) \cdot n + \tilde{O}(\epsilon^{-2}) \cdot (2n) = \tilde{O}(\epsilon^{-2}n)$.

Q-R complexity: $\tilde{O}(\epsilon^{-5}n)$.

We look at this Q-R complexity, $\epsilon^{-5}n$, as our yardstick, which we wish to improve. (It may be worth noting that Alon et al [AK02] have found a bipartitness tester with Q-R complexity of $\tilde{\Theta}(\epsilon^{-4}n)$, and we also improve this Q-R complexity.)

4.2 Decreasing Randomness by Using Better Hitters

Our goal is to mimic the above tester, while using as few coin tosses as possible. A closer examination of the two stages of the tester shows us what exactly we need in order to achieve this goal:

1. A sample of vertices U, so that for each vertex $v \in V$ of degree at least $\frac{\epsilon}{3}N$, it holds that U hits one of v's neighbors with probability at least $1 - \frac{\epsilon}{18}$.
2. A sample of vertex-pairs A, so that for each 2-coloring of U, it holds that A hits an edge that violates the 2-coloring with probability at least $1 - \frac{1}{6} \cdot 2^{-|U|}$.

Item (1) gives us a set U that has "many" violating edges if G is ϵ-far from bipartiteness, and item (2) allows us to use the union bound and to rule out all $2^{|U|}$ possible 2-partitions of U. Below, we refer to $1/6$ and $2^{-|U|}/6$ as the error parameters for Steps 1 and 2 respectively.

4.2.1 Implementation with a Hitter Linear in Confidence

In order to save randomness, we use our hitter linear in confidence. The hitter makes $O(1/\epsilon\delta)$ queries, and uses only $n = \log N$ random bits.

The first stage needs a hitter for sets of density $\epsilon/3$, which has confidence at least $1 - \epsilon/18$. Thus, it takes $O(\frac{3}{\epsilon} \cdot \frac{18}{\epsilon}) = O(\epsilon^{-2})$ samples, meaning $|U| = O(\epsilon^{-2})$. The randomness complexity is simply $n = \log N$.

The second stage needs a hitter for a set of density $\epsilon/3$ with confidence $1 - 2^{-|U|}/6$. Thus it takes $O(\frac{3}{\epsilon} \cdot 6 \cdot 2^{|U|}) = 2^{O(\epsilon^{-2})}$ samples, meaning $|A| = 2^{O(\epsilon^{-2})}$. The randomness complexity is $\log(N^2) = 2n$.

Note that the total number of queries that the bipartiteness tester makes is $|U|^2 + |U| \cdot |A| = 2^{O(\epsilon^{-2})}$.

Total Complexity:

Query complexity: $2^{O(\epsilon^{-2})}$.

Randomness Complexity: $n + 2n = 3n$.

Q-R complexity: $2^{O(\epsilon^{-2})}n$.

This shows that just decreasing the randomness will not decrease the total Q-R complexity of the tester. However, should one only consider a way to minimize the randomness complexity of Algorithm 12, we believe that this is the best possible implementation, since this algorithm requires us to choose a subset $U \subset V$ and a subset $A \subset V \times V$, tasks which take (roughly) at least n and $2n$ random bits, respectively.

4.2.2 Implementation with a Hitter Logarithmic in Confidence

Another possible implementation of the original tester, is with the hitter logarithmic in confidence.

The first step needs a hitter for subsets of V of density $\epsilon/3$ and confidence $1 - \epsilon/18$. Thus, it takes $O(\frac{3}{\epsilon}\log(\frac{18}{\epsilon})) = \tilde{O}(\epsilon^{-1})$ samples, meaning that $|U| = \tilde{O}(\epsilon^{-1})$, and will use $n + O(\log(1/\epsilon))$ coin tosses. It then makes $O(|U|^2) = \tilde{O}(\epsilon^{-2})$ queries to the graph.

The second step needs a hitter for subsets of $V \times V$ of density $\epsilon/3$ and confidence $1 - \frac{1}{6} \cdot 2^{-|U|}$. Thus it takes $O(\frac{3}{\epsilon}\log(2^{|U|})) = \tilde{O}(\frac{1}{\epsilon^2})$ samples, meaning $|A| = \tilde{O}(\epsilon^{-2})$, and uses $2n + O(|U|) = 2n + \tilde{O}(\epsilon^{-1})$ coin tosses. It then makes $O(|U| \cdot |A|)$ queries to the graph, so this step takes $\tilde{O}(\epsilon^{-3})$ queries.

Total Complexity:

Query Complexity: $\tilde{O}(\epsilon^{-3})$.

Randomness Complexity: $n + O(\log(1/\epsilon)) + 2n + \tilde{O}(1/\epsilon) = 3n + \tilde{O}(1/\epsilon)$.

Q-R Complexity: $\tilde{O}(\epsilon^{-3})(3n + \tilde{O}(1/\epsilon)) = \tilde{O}(\epsilon^3) \cdot n + \tilde{O}(1/\epsilon^4)$.

Note the considerable improvement in the Q-R complexity, just by using the new hitter. However, we inspire to achieve a lower Q-R complexity.

We note that the best Q-R complexity one can expect (assuming one doesn't decrease the query complexity of the algorithm) is $\tilde{O}(\epsilon^{-3}n)$. The query complexity of Algorithm 12 is $\tilde{O}(\epsilon^{-3})$. The randomness complexity of any property tester must be $\Omega(n)$, as it is shown in Section 3.1. This results in a Q-R complexity which is roughly $\epsilon^{-3} \cdot n$, and we achieve this goal.

As a "warm-up", we starting by presenting three algorithms, in which we modify the original GGR tester (Algorithm 12), and use different approaches to the problem. These three testers (presented in Sections 4.4, 4.5 and 4.6) have Q-R complexity which is *worse* than the Q-R complexity of the implementation of Algorithm 12 with a hitter logarithmic in confidence. However, the ideas that they use eventually lead (see Section 4.7) to the tester with better Q-R complexity. Furthermore, we believe that they offer interesting approaches that might also be applicable to other graph-property testers.

4.3 The Requirements Graph

Before we describe the algorithms we have come up with, we begin with introducing an important notion. All algorithms we present in Sections 4.4 through 4.7 share, in a way, the same general approach for reducing the randomness complexity of Algorithm 12. It is fairly obvious that the bottleneck of the randomness in the original tester is Step 2. Since we use the union bound to find violations for all $2^{|U|}$ possible 2-colorings of U, we have to make the probability that for a single 2-coloring a violating edge isn't found be smaller than $2^{-|U|}$. This requires us to use a hitter with confidence at least $1 - 2^{-|U|}$, which in turn requires a randomness complexity at least $|U| > \epsilon^{-1}$. Roughly speaking, our goal, from now on, is to reduce the number of "bad events" in the union bound to something that isn't proportional to $\exp(|U|)$ but rather to $poly(|U|)$. Towards that end, we introduce the notion of requirements, and the notion of the requirements graph of a set $U \subset V$. Finding requirements will be the key ingredient in reducing the number of "bad events".

Definition 4.4. *Given any 2- partition of U, denoted $\langle U_1, U_2 \rangle$, we denote $u_1 \sim u_2$ if u_1 and u_2 belong to the* same *side of the 2-partition, and we denote $u_1 \nsim u_2$ if u_1 and u_2 belong to* different *sides. We call $(u \sim v)$ or $(u \nsim v)$ a* requirement*, and say that a 2-partition* satisfy the requirement $u_1 \sim u_2$ *(respectively $u_1 \nsim u_2$) if indeed u_1 and u_2 belong to the same side (respectively different sided) of the 2-partition.*

Edges and paths in G, connecting two vertices of U, automatically yield requirements that any 2-coloring of V must satisfy. We give here three archetypical examples, which will be repeatedly used in the analysis of the algorithms in the next sections. We denote by u_1 and u_2 two (different) vertices of U, and by v_1 and v_2 two (different) vertices in $V \setminus U$.

Example 4.5.

1. If we have an edge $\langle u_1, u_2 \rangle$ in $G[U]$, then we know that every 2-coloring of U must give u_1 and u_2 different colors. Therefore we have that requirement $u_1 \nsim u_2$ that every 2-coloring of U must satisfy.

2. If we have a path of length 2, of the form (u_1, v_1, u_2), then we know that any 2-coloring of V gives u_1 and v_1 different colors, and similarly, it gives u_2 and v_1 different colors. Since we only have two colors, we deduce that every 2-coloring of V must satisfy $u_1 \sim u_2$. Specifically, if G is bipartite, then any 2-coloring of V satisfies $u_1 \sim u_2$, and therefore, any 2-coloring of U that is induced by a 2-coloring of V, also must satisfy $u_1 \sim u_2$.

3. If we have a path of length 3, of the form (u_1, v_1, v_2, u_2), then by the same reasoning, any 2-coloring of V must satisfy $u_1 \nsim u_2$. And again, if G is bipartite, than any 2-coloring of U that is induced by a 2-coloring of V must satisfy $u_1 \nsim u_2$ as well.

We note that by the definition of a violating edge, it yields a requirement over two vertices in U. Suppose we fixed some 2-partition of U, and we found a violating edge for it, denoted as $\langle v_1, v_2\rangle$. That means that we now have two vertices u_1 and u_2 in U, where u_1 is adjacent to v_1, and u_2 is adjacent to v_2. Therefore, just like in Case 3 of Example 4.5, we have found a path (u_1, v_1, v_2, u_2), which yields the requirement $u_1 \nsim u_2$, which every 2-coloring of V must satisfy.

Definition 4.6. *Given a subset $U \subset V(G)$, and a set T of requirements that every 2-coloring of U must satisfy, the* requirements graph *of U and T, denoted $R(U,T)$, is a graph whose vertices are the vertices of U. An edge $\langle u, v\rangle$ belongs to $R(U,T)$ if and only if a requirement over u and v belongs to T, meaning if either $(u \nsim v) \in T$ or $(u \sim v) \in T$. We label every edge of $R(U,T)$ with $'\nsim'$ if the requirement $u \nsim v$ belongs to T, or with $'\sim'$, if the requirement $u \sim v$ belongs to T.*

Given an edge $\langle u, v\rangle$ in $R(U,T)$, we say that the requirement $u \sim v$ (respectively $u \nsim v$) is *associated* with the edge, if the edge $\langle u, v\rangle$ is labeled with $'\sim'$ (respectively with $'\nsim'$). Theoretically, it is possible that some edge is labeled both with $'\sim'$ and with $'\nsim'$.

We say a 2-coloring *satisfies an edge $\langle u, v\rangle$* in $R(U,T)$ if it satisfies the requirement over u and v associated with that edge.[11] Let C be a connected component of $R(U,T)$. We say a 2-coloring *satisfies the component C* if it satisfies all edges in C. Recall that we identify any 2-coloring $\langle U_1, U_2\rangle$ with $\langle U_2, U_1\rangle$. The following claim, as easy as it is to prove, is the basis for all of the algorithms presented in the next sections.

Claim 4.7. *Fix U and T. For every connected component C in $R(U,T)$, there exists at most one 2-coloring of the vertices of C that satisfy C.*

Proof. Clearly, if even one edge in C is labeled both with $'\sim'$ and with $'\nsim'$, then no 2-coloring of C satisfies this edge and we are done, so we assume all edges have a single label. Since all vertices in C are connected, then we have some spanning tree S. Fix an arbitrary vertex $u \in C$ and a side for u. Let v be a vertex adjacent to u is S. Since the side of u is fixed, then there is only one side we can give v in order to satisfy the edge $\langle u, v\rangle$. So fixing a side for one vertex, fixes the side of all of its neighbors. We can now go over the neighbors of v is S, and fix their side, and then go over their neighbors, and so on. Eventually, by traversing all edges in S, we fix a side for each vertex in C.

Therefore, we have found a *single* 2-partition of C, which is the only 2-partition that satisfies the edges in S. Since the edges in S are a subset of the edges in the connected component, then this is the only possible 2-partition of C that can satisfy C.

Now, look at any edge in C, that does not belong to S. If the 2-partition we fixed doesn't satisfy this edge, then it is obvious that no 2-partition can satisfy both this edge *and* all the edges in S, and therefore, no 2-partition of C satisfies C. Otherwise, all edges in C are satisfied by the 2-partition we fixed, and by definition this 2-partition is a 2-coloring that satisfy C. □

[11] If the edge is labeled with both $'\sim'$ and with $'\nsim'$, then no 2-coloring of U satisfies it.

How does Claim 4.7 help us? Assume that $R(U,T)$ has k connected components. Pick from each component C_i some arbitrary vertex u_i. The proof of Claim 4.7 shows that fixing the color of each u_i fixes the color of all the vertices in C_i. Therefore, we only need to consider all $\frac{1}{2} \cdot 2^k$ possible 2-colorings of $\{u_1, u_2, \ldots, u_k\}$. So the less components $R(U,T)$ has, the less 2-partitions of U we need to consider. Furthermore, note that for every new requirement we find, we add another edge to $R(U,T)$. That means that we either unite two connected components C_1 and C_2 into a single connected component, or we add an edge inside some connected component C. If we find an edge that unites two connected components, then we reduce that number of possible 2-partitions of U that we need to consider in half. If we find an edge that connects two vertices that belong to the same connected component, then it is possible that this edge causes no 2-partition of C to satisfy C. Therefore, all algorithms in the next sections will try to find more requirements and have a requirements graph $R(U,T)$ with as many edges as they can find.

4.4 Sequential Forcing of Partitions

We introduce a new algorithm, whose Q-R complexity isn't better than the one presented in Section 4.2.2, but it demonstrates the use of iterations in testing bipartiteness. Its analysis also demonstrates the usefulness of the requirements graph. The basic idea of this algorithm is as in the original Algorithm 12: We pick a set $U \subset V$, and we now wish to rule-out all $2^{|U|}$ possible 2-partitions it may have. Note that even though there are $2^{|U|}$ different 2-partitions for U, we can rule out these 2-partitions not all together, but rather one by one. More importantly, it suffices to consider only $|U|$ of these 2-partitions, which can be determined "on the fly", in iterations. The general outline of the algorithm is as follows.

First, we select a set $U \subset V$, and query all edges in $G[U]$. As shown in Example 4.5, every edge $\langle u, v\rangle$ in $G[U]$ yields an edge $\langle u, v\rangle$ labeled $' \nsim '$ in the requirements graph of U. So the algorithm begins with the set of requirements that the edges in $G[U]$ yield. Then the algorithm begins the iterations part.

At the beginning of each iteration, we have a set of requirements which we assume some 2-partition of U satisfies. We fix a 2-partition $\langle U_1, U_2\rangle$ of U, that satisfies all requirement found thus far. We then seek a violating edge for it. Once a violating edge, denoted $\langle v_1, v_2\rangle$, is found, it means that there exists a path of the form (u_j, v_1, v_2, u_k) for some two vertices u_j and u_k, both belong to either U_1 or both belong to U_2. We thus deduce that if G is bipartite, then the induced 2-coloring over U must satisfy the requirement $u_j \nsim u_k$. So we add the requirement $u_j \nsim u_k$ to the list of all requirements found thus far. We then repeat this iteration, with respect to a new 2-partition of U that satisfies all requirements found (which includes the newly found requirement $u_j \nsim u_k$).

Now, look at the requirements graph of U with respect to the requirements found prior to the iteration. The new requirement $u_j \nsim u_k$ adds a new edge to this requirements graph.

Either both endpoints of this edge belong to two different components, or we have that both endpoints of this edge belong to the same component. As we will show, after at most $|U|$ iterations, it is likely that we add an edge $\langle u_j, u_k \rangle$ where both its endpoints belong to the same connected component. We also show that once we have that u_j and u_k belong to the same component C in $R(U, T)$, then we have that no 2-partition of C satisfies both the newly add edge $\langle u_j, u_k \rangle$ *and* C. Therefore G isn't 2-colorable, for otherwise the 2-coloring of V would induce a 2-partition of U that satisfies all requirements found.

We wish to note that this algorithm uses a requirements graph with only $'\nsim'$-labeled edges. That is because we only look at requirements that result from edges in $G[U]$, as in Case 1 of Example 4.5, and at requirements that result for violating edges, as in Case 3 of Example 4.5.

We also note that in each iteration we use independent coin toss, in order to avoid any dependencies between the samplers used to rule out the 2-partition of one iteration, with the samples used in some other iteration. Since we have $|U|$ iterations, it means that the randomness complexity of this algorithm is still proportional to $|U|$.

Formally, our algorithm is as follows:

Algorithm 13 Testing Bipartiteness by Sequential Forcing of Partitions

Pick a set $U \subset V$, using a hitter for sets of density at least $\epsilon/3$ and confidence at least $1 - \epsilon/18$. Query $G[U]$, and let T be the set of requirements that the edges in $G[U]$ yield (all labeled $'\nsim'$).
for $i = 1, 2, \ldots, |U|$ **do**
 If there exists no 2-coloring of U that satisfies all requirements in T, output *"not bipartite"* and halt.
 Choose an arbitrary 2-partition of U that satisfy all requirements in T, denoted $\langle U_i, \overline{U_i} \rangle$.
 Pick a set $A \subset V \times V$ using a hitter for sets of density at least $\epsilon/3$ and confidence at least $1 - \frac{1}{6|U|}$. For every vertex-pair $(v_1, v_2) \in A$, query all potential edges between v_1, v_2 and U, to see whether the edge $\langle v_1, v_2 \rangle$ is a violating edge for $\langle U_i, \overline{U_i} \rangle$.
 For every $(v_1, v_2) \in A$ such that $\langle v_1, v_2 \rangle$ is found to be a violating edges for $\langle U_i, \overline{U_i} \rangle$, let u_1 and u_2 be two vertices in U that belong to the same side, where u_1 is adjacent to v_1 and u_2 is adjacent to v_2. Add the requirement $u_1 \nsim u_2$ to T.
end for
If there exists a 2-coloring of U that satisfies all requirements in T, output *"bipartite"*. Otherwise, output *"not bipartite"*.

Theorem 4.8. *Algorithm 13 is a 1-sided tester for bipartiteness.*

Proof. If G is bipartite, any 2-coloring of V satisfies T, the set of all requirements found by Algorithm 13. That is because the requirements in T are $'\nsim'$ requirements arising from the connectivity in G. Therefore, any 2-coloring of V induces some 2-coloring of U that satisfies T. This means that no matter which requirements the tester has found - there exists a 2-

coloring of U that satisfies all requirements found, and so the tester completes all iterations, and eventually outputs *"bipartite"*.

Fix a graph G that is ϵ-far from bipartiteness. As in Claim 4.3 and the discussion in Section 4.2, the probability that U isn't good is at most $1/6$. If U is good, again by Claim 4.3, any 2-partition of U has at least $(\epsilon/3)N^2$ violating edges. Specifically, for each iteration i, each $\langle U_i, \overline{U_i}\rangle$ has at least $(\epsilon/3)N^2$ violating edges. By the definition of a hitter, the set A fails to hit a violating edge for $\langle U_i, \overline{U_i}\rangle$ with probability at most $\frac{1}{6|U|}$. Therefore, given that U is good, the probability that for one of the $|U|$ iterations of the algorithm the set A fails to hit a violating edge, is upper bounded by $\sum_{i=1}^{|U|} \frac{1}{6|U|} = 1/6$. Thus, with probability at least $2/3$, at every iteration, the set A did manage to hit a violating edge, which means that a new requirement was found, as shown in Case 3 of Example 4.5.

Assume that at each iteration, we find a new violating edge, and add one more edge to the requirements graph of U. Suppose this new edge has its endpoints in two different connected components. Then after adding this new edge, those two components were united into a single component, and so the number of components in the requirements graph decreases by 1. Since we repeat $|U|$ iterations in which we add an edge to a graph over $|U|$ vertices, then at some iteration i, we add a new edge with both its endpoints in the same connected component.

We use the following notation: At the beginning of the ith iteration, we had a set of requirements, denoted T. At the end of the ith iteration, we add an edge $\langle u_1, u_2\rangle$ to the requirements graph $R(U, T)$, where both its vertices belong to the same connected component C. By the definition of the tester, the edge $\langle u_1, u_2\rangle$ was added to $R(U, T)$ because we have found a violating edge $\langle v_1, v_2\rangle \in E(G)$ for the 2-partition of U that we fixed, that is to $\langle U_i, \overline{U_i}\rangle$.

Recall, we fixed $\langle U_i, \overline{U_i}\rangle$ because it was a 2-partition that satisfies all requirements in T, and specifically, it satisfied C. By Claim 4.7 we have that at most one 2-partition of C can satisfy C. Therefore, $\langle (U_i \cap C), (\overline{U_i} \cap C)\rangle$ is the only 2-partition of C that satisfies C. However, we add the edge $\langle u_1, u_2\rangle$ to $R(U, T)$, which we know this 2-partition does not satisfy, because it results from the violating edge $\langle v_1, v_2\rangle$. Thus, there exists no 2-partition that satisfies both C and the requirement $u_1 \nsim u_2$, and so, in the next iteration, the tester outputs *"not bipartite"*.

We deduce that if G is ϵ-far from bipartiteness, then with probability at least $2/3$ the algorithm finds a new requirement in each iteration, and eventually outputs *"not bipartite"*.

□

4.4.1 Implementation with a Hitter Linear in Confidence

As before, U is picked by using a hitter for sets of density $\epsilon/3$ with probability at least $1-\epsilon/18$ which requires $O(\epsilon^{-2})$ samples and n random bits, and thus Step 1 take $O(\epsilon^{-4})$ queries.

As noted before, each iteration uses new coins, independent of the coins used in other iterations. Otherwise, it is possible that the random bits used to find a violating edge for the 2-coloring of iteration i may fail to find a violating edge in some iteration j, where $j > i$. One extreme example to that is where all iterations use the same random bits to produce A, and so A is the exact same set in all iterations. In such a case iterations 2 through $|U|$ will fail to introduce new requirements for a 2-coloring of U, even through that G might be ϵ-far from bipartite, and U is good.

We repeat independently $|U|$ times the procedure of hitting a set of edges with density at least $\epsilon/3$ with confidence at least $1 - \frac{1}{6|U|}$. This requires $2n$ random bits and $s = O(|U|\epsilon^{-1})$ samples per iteration, where for each sample we query $O(|U|)$ edges in the graph. Thus, the entire process takes $|U| \cdot 2n$ random bits, and $|U| \cdot s \cdot O(|U|) = O(\epsilon^{-7})$ queries.

Total Complexity:

Query Complexity: $O(1/\epsilon^7)$.

Randomness Complexity: $n + O(|U|) \cdot 2n = O(\epsilon^{-2}n)$.

Q-R Complexity: $O(\epsilon^{-9}n)$.

4.4.2 Implementation with a Hitter Logarithmic in Confidence

Note that the query and randomness complexity of the tester is polynomially dependent on the size of U, so reducing the size of U will reduce drastically the number of samples and random bits the tester needs. It was already shown that if we use a hitter logarithmic in confidence, then U is of size $\tilde{O}(\epsilon^{-1})$. This way U is produced using $n + O(\log(1/\epsilon))$ random bits, and the number of queries made in Step 1 is $O(|U|^2) = \tilde{O}(\epsilon^{-2})$.

As before, we need to repeat independently $|U|$ times the procedure of hitting a set of edges with density at least $\epsilon/3$ with confidence at least $1-\frac{1}{6|U|}$. This requires $2n+O(\log(|U|))$ random bits and $s' = O(\epsilon^{-1}\log(|U|))$ samples per iteration, where for each sample we query $O(|U|)$ edges in the graph. Thus, the entire process takes $|U| \cdot s' \cdot O(|U|) = \tilde{O}(\epsilon^{-3})$ queries and $\tilde{O}(|U| \cdot n)$ random bits.

Total Complexity:

Query Complexity: $\tilde{O}(1/\epsilon^3)$.

Randomness Complexity: $n + O(\log(1/\epsilon)) + 2n \cdot |U| = \tilde{O}(\epsilon^{-1}) \cdot n$.

Q-R Complexity: $\tilde{O}(\epsilon^{-4}) \cdot n$.

Again, we note that even though this algorithm gives worse Q-R complexity than the previous algorithm, we described it here because the notion of the requirements graph will play a crucial role in the future design of a tester that has smaller Q-R complexity than what we have thus far. Furthermore, the idea of iterations will be applied there.

4.5 Concurrent Forcing of a Partition

This time we try another approach. Instead of choosing sequentially several 2-colorings of U, we choose in advance just *one* 2-coloring and seek violating edges only for this 2-coloring. Our goal is to choose wisely a "useful" 2-coloring of U, "useful" in the sense that if G is bipartite then it has "few" violating edges, and if G is ϵ-far from being bipartite then it has "many" violating edges. This will be achieved by finding requirements and adding them to the requirements graph of U, and focusing only on a single 2-coloring that satisfy all components in the requirements graph. Details follow.

Suppose that we've already picked a set U. Denote by l the number of connected component in $G[U]$ (naturally $l \leq |U|$), and denote these components by $C_1, C_2, \ldots, C_l$. Note, that as it was already shown, if two vertices $u, v \in U$, are adjacent, then the edge $\langle u, v \rangle \in E(G)$ yields the requirement $u \nsim v$, which every 2-coloring of U must satisfy. Suppose we are lucky, and $G[U]$ is connected. Then the requirements graph of U consists of a single connected component, and by Claim 4.7 only a single 2-partition of U can satisfy the requirements that the edges in $G[U]$ yield. We thus fix this 2-partition, and seek violations only for it. If G is bipartite, then this 2-partition has no violations, and if G is ϵ-far from bipartite (and we have a good U), then by Claim 4.3 it has at least $(\epsilon/3)N^2$ violating edges.

Now assume that another extreme case holds, and that $G[U]$ is consisted of isolated vertices. Assume also that for any two vertices, u and v in U, there exists no path, even in G, that connects u and v. In this case, if G is bipartite, any 2-partition we fix for U will have no violating edges, because any 2-coloring of U is induced by some 2-coloring of V. If G is ϵ-far from bipartiteness (and we have a good U), then any 2-partition of U still has at least $(\epsilon/3)N^2$ violating edges, by Claim 4.3. If this extreme case holds, then the endpoints of each violating edge are both adjacent to the same vertex $u \in U$.

Naturally, we cannot assume that $G[U]$ is connected, nor can we assume its connected components are not connected even in G. However, we aspire to find a requirements graph of U in which every two different components are, in a way, very close to being disconnected in G. We therefore seeks vertices and edges in $V \setminus U$ that connect two components in $G[U]$ by paths of length 2 or 3. As shown in Example 4.5, such paths yield requirements for U. Let us formally define what we wish to find.

Definition 4.9. *We say that a pair of vertices* (v_1, v_2) forms a bypass *between* C_i *and* C_j *(where* $i \neq j$*), if* C_i *and* C_j *are connected in* $G[C_i \cup C_j \cup \{v_1, v_2\}]$.

This means that we have some vertex $u_i \in C_i$ and some vertex $v_j \in C_j$, such that (at least) one of the following cases must hold.

- For some $v \in \{v_1, v_2\}$ the path (u_i, v, u_j) exists in $G[C_i \cup C_j \cup \{v\}]$, and we have found the requirement $u_i \sim u_j$.

- The path (u_i, v_1, v_2, u_j) exists in $G[C_i \cup C_j \cup \{v_1, v_2\}]$, and we have found the requirement $u_i \nsim u_j$.

Therefore, if the vertex-pair (v_1, v_2) forms a bypass between C_i and C_j, then we have found a new requirements, that unites the components C_i and C_j in the requirements graph of U into a single connected component.

Conceptually, the algorithm goes over all $\binom{l}{2}$ pairs of connected components C_i and C_j, and tries to find vertex-pairs that form a bypass between C_i and C_j. This raises the question of when can we expect the tester to find a vertex-pair that forms a bypass between C_i and C_j. To that end, we introduce the next definition.

Definition 4.10. *A pair of connected component (C_i, C_j) is said to be* heavily connected *if at least $\frac{\epsilon}{5|U|^2}N^2$ vertex-pairs in $V \times V$ form a bypass between C_i and C_j.*

Using a good hitter for pairs of vertices, we are likely to hit a vertex-pair that forms a bypass between C_i and C_j, if they are a pair of heavily connected components. But what about pairs of components which aren't heavily connected? This also helps us, since, as we show, a pair of connected components that aren't heavily connected has relatively few edges interfering with any 2-partition that satisfy both components. So, in a way, this resembles the case in which all components of $G[U]$ are disconnected even in G.

And so, this is what our algorithm does. It goes over all pairs of components of $G[U]$ and tries to hit a vertex-pair that forms a bypass for each pair. Each vertex-pair that forms a bypass between two connected components, yields some requirement that connects these two components in the requirements graph of U. After going over all $\binom{l}{2}$ pairs of components, assume we have a requirements graph of U with k components. The algorithm fixes one of the 2^{k-1} possible 2-partitions of U that satisfy all requirements found. Now, if G is bipartite, we show that any violating edge for this 2-partition is an edge whose endpoints form a bypass between two connected components of $G[U]$. We also show that if for any pair of components in $G[U]$ that are heavily connected, the algorithm hits a vertex-pair that forms a bypass for this pair, then only relatively few vertex-pairs interfere with the fixed 2-partition. So, on the one hand we have that if G is bipartite, we are likely to fix a 2-partition of U that has relatively few violating edges. On the other hand, if G is ϵ-far from bipartiteness (and U is good), then Claim 4.3 assures us that this 2-partition has relatively many violating edges. So the final step of the algorithm is to estimate the number of violating edge for the 2-partition it fixed, and to accept or reject G based on finding a few or many violating edges, respectively.

Before introducing the algorithm, we wish to note, that the fact we go over all $\binom{l}{2} = O(|U|^2)$ pairs of connected components, does not mean that we need to use a hitter $O(|U|^2)$ times, each time using an independent coin toss. We use the same sample set for *all* $\binom{l}{2}$ pairs, but we make the failure probability of the hitter be $O\left(l^{-2}\right)$. Using the union bound, we can upper bound the probability that there exists some pair of connected components for which the hitter fails, by $\frac{1}{2}l^2 \cdot O\left(l^{-2}\right) = 1/6$.

Formally, the algorithm is:

Algorithm 14 Testing Bipratitness by Forcing a Partition (2-Sided Version)

1: Pick a set $U \subset V$, using a hitter for sets of density at least $\epsilon/3$ and confidence at least $1-\epsilon/18$. Query $G[U]$, and let T be the set of requirements that the edges in $G[U]$ yield.

2: Pick a set $A \subset V \times V$ using a hitter for sets of density at least $\frac{\epsilon}{5|U|^2}$ and confidence at least $1 - \frac{1}{6} \cdot \frac{1}{\frac{1}{2}|U|^2}$. For every vertex-pair $(u, v) \in A$, query all potential edges connecting u, v and U, to see whether the vertex-pair (u, v) forms a bypass between some pair of connect components of $G[U]$.

3: For every vertex-pair (v_1, v_2) that forms a bypass between some two connected component, C_i and C_j, let $u_i \in C_i$ and $u_j \in C_j$ be the two vertices that (v_1, v_2) yields a requirement for. Add to T the requirement found (either $u_i \sim u_j$ or $u_i \nsim u_j$, according to Example 4.5).

4: Choose an arbitrary 2-partition $\langle U_1, U_2\rangle$ of U that satisfies all requirements in T (if such exists, otherwise declare *"not bipartite"* and halt). Pick a set $B \subset V \times V$ using a sampler with accuracy $\epsilon/10$ and confidence $5/6$. For every vertex-pair $(u, v) \in B$ query all potential edges between u, v and U, to see whether $\langle u, v\rangle$ is a violating edge for $\langle U_1, U_2\rangle$.

5: If the fraction of violating edges for $\langle U_1, U_2\rangle$ is found to be at most $\epsilon/5$ then output *"bipartite"*, and if it is found to be greater than $\epsilon/5$ output *"not bipartite"*.

Theorem 4.11. *Algorithm 14 is a 2-sided error bipartiteness tester.*

The reason that this tester has 2-sided error is due to the fact that we allow the case where G is bipartite and still we fix a 2-partition of U that has violations. The analysis of previous algorithms was based showing that if G is bipartite then some 2-partition of U has no violations, and if G is ϵ-far from bipartiteness then all 2-partitions of U have many violations. Therefore, once for any 2-partition of U we found even one violation, we had a proof that G isn't bipartite. The analysis of Algorithm 14 is based on the fact that we fix an *arbitrary* 2-partition of U for which the following holds: If G is bipartite, it has few violating edges, and if G is ϵ-far from bipartiteness it has many violating edges. Therefore, we need to estimate the number of violating edge for the 2-partition we fix. It might hold that G is bipartite, yet we fix a 2-partition that has violating edges, and obtaine a wrong estimation as to the number of violating edges. Therefore, it is possible we reject a graph which is bipartite.

Proof. By Claim 4.3 and the analysis in Section 4.2, Step 1 will fail to produce a set U that dominates all but a fraction of $\epsilon/3$ of the vertices with high degree ($deg(v) \geq (\epsilon/3)N$) with probability at most $1/6$. Thus, we will assume that we have a good U.

The probability that Step 2 fails is the probability that for some pair (C_i, C_j), even though more than $\frac{\epsilon}{5|U|^2}N^2$ of the vertex-pairs of the graph do form a bypass between C_i and C_j, we failed to hit such a vertex-pair. For a certain pair (C_i, C_j) this happens with probability at most $\frac{1}{6} \cdot \frac{1}{\frac{1}{2}|U|^2}$ and since that there are at most $\frac{1}{2}|U|^2$ pairs, by the union bound we have that Step 2 fails with probability at most $1/6$.

By definition, the sampler of Step 4 produces an estimation that is $\epsilon/10$ far from the real fraction of violating edges in the graph with probability at most $1/6$.

Suppose G is bipartite, and so, no matter which $U \subset V$ we pick, $G[U]$ is bipartite. Assume Step 2 did not fail; that is, for all pairs of connected components, which are heavily connected, we found a vertex-pair that forms a bypass between them. Under that assumption, let us look at Step 4, and prove that the 2-partition we pick, $\langle U_1, U_2 \rangle$, doesn't have many violating edges.

If we picked $\langle U_1, U_2 \rangle$ which is a 2-coloring of U induced from a 2-coloring of V, then it has no violating edges, and we are done. So assume we picked some 2-partition of U that does have violating edges. Recall that we picked $\langle U_1, U_2 \rangle$ that satisfies all requirements in T. In particular, $\langle U_1, U_2 \rangle$ satisfies all connected components in the requirements graph $R(U, T)$. We claim that if G is bipartite, and $\langle v_1, v_2 \rangle$ is a violating edge for $\langle U_1, U_2 \rangle$, then the vertex-pair (v_1, v_2) forms a bypass between two connected components of $G[U]$, that are *not heavily connected.*

We know that each violating edge found, $\langle v_1, v_2 \rangle$, yields a requirement for some u_i and u_j in U, and therefore, adds an edge $\langle u_i, u_j \rangle$ to the requirements graph $R(U, T)$. Suppose u_i and u_j belong to the same component. We know that $\langle U_1, U_2 \rangle$ satisfies this component, and still, a violating edge for the 2-coloring of this component was found. By Claim 4.7 we deduce that this component has no 2-coloring. But G is bipartite, and so the 2-coloring of V induces some 2-coloring of this connected component, and we have a contradiction. Therefore, we deduce that u_i and u_j must belong to two different components in $R(U, T)$. Specifically, it means that $u_i \in C_i$ and $u_j \in C_j$ for some two different components, C_i and C_j, of the subgraph $G[U]$. If C_i and C_j were heavily connected, then, assuming the hitter of Step 2 did not fail, we would hit a vertex-pair that forms a bypass between C_i and C_j, and they both would be contained in the same component in $R(U, T)$. That means that C_i and C_j are not heavily connected.

We deduce that for all violating edges of $\langle U_1, U_2 \rangle$, their endpoints must form a bypass between pairs of connected components of U that are *not* heavily connected. Since at most $\frac{1}{2}|U|^2$ pairs are not heavily connected, we deduce that the total number of such vertex-pairs is at most $\frac{1}{2}|U|^2 \cdot \frac{\epsilon}{5|U|^2}N^2 = \frac{\epsilon}{10}N^2$. If Step 4 doesn't fail then the sampler's estimation *must* be at most $\frac{\epsilon}{10} + \frac{\epsilon}{10} = \frac{\epsilon}{5}$. As shown before, the probability that both step 2 and step 3 do not fail is bounded by $1 - (1/6 + 1/6) = 2/3$, so with probability at least $2/3$ the tester outputs *"bipartite"*.

Suppose G is ϵ-far from being bipartite. By Claim 4.3 and the analysis in Section 4.2, we have that Step 1 picks a good U with probability at least 5/6. By Claim 4.3, for a good U and for every 2-partition of U, there are at least $(\epsilon/3)N^2$ violating edges. Note that no matter what 2-partition we fix in Step 4, it will have at least $(\epsilon/3)N^2$ violating edges, and if sampler's estimation is $\epsilon/10$ close to the actual fraction of violating edges in the graph, then its estimation is at least $\epsilon/3 - \epsilon/10 > \epsilon/5$. The probability of failure is thus upper bounded by the probability that either Step 1 or Step 3 fail, which is upper bounded by $1/6 + 1/6 = 1/3$. □

4.5.1 Implementation with a Hitter/Sampler Linear in Confidence

By the same analysis from before, U will be of size $O(\epsilon^{-2})$ which means that Step 1 takes $O(\epsilon^{-4})$ queries and n random bits. Step 2 will require $s = \frac{5|U|^2}{\epsilon} \cdot \frac{1}{2}|U|^2 = O(\epsilon^{-9})$ samples (since we use the same samples for all pairs of connected components), where for each vertex-pair we make $(2|U|+1) = O(|U|)$ queries. So Step 2 takes a total of $s \cdot O(|U|) = O(\epsilon^{-11})$ queries, and $2n$ random bits. Step 4 requires $O(\epsilon^{-2})$ samples, where for each we make $O(|U|)$ queries, meaning in Step 4 we make $O(\epsilon^{-2} \cdot |U|) = O(\epsilon^{-4})$ queries, and use additional $2n$ random bit. All in all, we make $O(\epsilon^{-11})$ queries, and use $5n$ random bits.

Total Complexity:

Query Complexity: $O(\epsilon^{-11})$.

Randomness Complexity: $n + 2n + 2n = 5n$.

Q-R Complexity: $O(\epsilon^{-11} \cdot n)$.

4.5.2 Implementation with a Hitter/Sampler Logarithmic in Confidence

U is of size $\tilde{O}(\epsilon^{-1})$ and so the number of queries made in Step 1 is $\tilde{O}(\epsilon^{-2})$, where we use $n + O(\log(1/\epsilon))$ random bits to produce U. The number of samples needed in Step 2 is $s = O(\frac{5|U|^2}{\epsilon}\log(|U|^2)) = \tilde{O}(\epsilon^{-3})$, where for each edge we make $O(|U|) = \tilde{O}(\epsilon^{-1})$ queries, meaning we do $s \cdot O(|U|) = \tilde{O}(\epsilon^{-4})$ queries, while using $2n + O(\log(1/\epsilon))$ random bits. In Step 4 we use a sample of $O(1/\epsilon^2 \log(1/6)) = O(\epsilon^{-2})$ vertex-pairs, and for each vertex-pair we make $O(|U|)$ queries, thus Step 4 takes $\tilde{O}(1/\epsilon^3)$ queries, and additional $2n + O(1)$ random bits. All in all, our query complexity is $\tilde{O}(\epsilon^{-4})$, and the number of random bits is $5n + O(\log(1/\epsilon))$.

Total Complexity:

Query Complexity: $\tilde{O}(\epsilon^{-4})$.

Randomness Complexity: $5n + O(\log(1/\epsilon))$.

Q-R Complexity: $\tilde{O}(\epsilon^{-4}) \cdot n$.

4.6 Changing the 2-Sided Tester into a 1-Sided Tester

Clearly, the fact that we have a 2-sided tester and not a 1-sided tester is a drawback, which, as we show here, can be removed without asymptotically increasing the query/randomness/Q-R complexity of the tester. In order to turn Algorithm 14 into a 1-sided tester, let us look carefully at its analysis.

Suppose we picked a set U and found a set T of requirements. What we showed in the analysis of Algorithm 14 is that with respect to $R(U, T)$ there exist two type of violating edges. The first type are violating edges that yield a requirement over two vertices that belong to two different components of $R(U, T)$. The second type are violating edge that yield a requirement over two vertices that belong to the same component in $R(U, T)$. If G is bipartite, then any 2-coloring of G satisfies T, and more importantly any 2-partition of U that satisfies T has only violating edges of the first type. If G is ϵ-far from bipartite, then any 2-partition of U, is likely to have at least $(\epsilon/3)N^2$ violating edges of both types.

In the analysis of Algorithm 14, we have shown how to come up with a set of requirements T, such that the number of violating edges of the first type is (likely to be) at most $(\epsilon/10)N^2$. This means that by finding such a set T, and fixing a 2-partition of U that satisfies all requirements in T, the following holds: If G is bipartite, then no violating edge of the second type exists. If G is ϵ-far from bipartite, then at least $\left(\frac{\epsilon}{3} - \frac{\epsilon}{10}\right) N^2 \geq \frac{\epsilon}{6} N^2$ violating edges of the second type exist. Now, using a hitter for sets of density $\frac{\epsilon}{6}$ we are likely to hit a violating edge of the second type, if G is ϵ-far from bipartiteness.

All is left for us is to define formally what is a violating edge of the first type, and what is a violating edge of the second type.

Definition 4.12. *Given $U \subset V(G)$ and a set T of requirements for U, a partition $\Pi = (U_1, U_2, \ldots, U_t)$ of U is said to be T-*legal *if every U_i consists of a single connected component in $R(U, T)$.*

That means, that when we have U and a set of requirements T, then, by Claim 4.7, for every set U_i in the T-legal partition of U, there exists at most one 2-partition of U_i that can satisfy all requirements in T. As we have already shown in the end of Section 4.3, if there exists $U_i \in \Pi$ that has no 2-partition that satisfies T, then U has no 2-partition that satisfies T. Otherwise, for every U_i there exists a 2-partition that does satisfy T, and U has 2^{t-1} possible 2-partitions that satisfy T.

The following is the formal definition for a violating edge of the second type mentioned above.

Definition 4.13. *Let U be a subset of V and let T be a set of requirements. Let Π be a T-legal partition of U. Let $\langle S, \overline{S} \rangle$ be any 2-partition of U that satisfies every requirement in T. Suppose that $\langle v_1, v_2 \rangle$ is a violating edge for $\langle S, \overline{S} \rangle$, which yields a requirement over some*

u_j and u_k in U. Then we say that the vertex-pair (v_1, v_2) is Π-violating *if for some i we have that both u_j and u_k belong to U_i.*

This means that once we have Π, we have that for each violating edge, either its endpoints are Π-violating, or its endpoints are such that for u_j and u_k from Definition 4.13, there exist two different sets, U_j and U_k, such that $u_j \in U_j$ and $u_k \in U_k$.

We wish to show that if G is bipartite, and we have a set of requirements T that every 2-coloring of U must satisfy, and Π is a T-legal partition, then no Π-violating edges exist.

Claim 4.14. *Let G be a bipartite graph. Let U be any subset of V. Let T be a set of requirements that every 2-coloring of $V(G)$ must satisfy. Let Π be the T-legal partition of U, and let $\langle S, \overline{S} \rangle$ be a 2-partition of U that satisfies T. Then $\langle S, \overline{S} \rangle$ has no Π-violating vertex-pairs.*

Proof. Since $\langle S, \overline{S} \rangle$ satisfies T, then it satisfies every connected component C in $R(U, T)$. By Claim 4.7, we have that for every C, the 2-partition $\langle (S \cap C), (\overline{S} \cap C) \rangle$ is the only 2-partition of C that can satisfy T. Assume that there exists an edge $\langle v_1, v_2 \rangle$ which is a $\langle S, \overline{S} \rangle$-violating edge, so that (v_1, v_2) is a Π-violating vertex-pair. Then by definition, the pair (v_1, v_2) yields a requirement over some u_j and u_k, which belong to the same U_i. By the definition of a T-legal partition, we have that U_i consists of the vertices of some connected component C_i in $R(U, T)$, so u_j and u_k both belong to C_i. We have that the edge $\langle v_1, v_2 \rangle$ violates the 2-partition $\langle (S \cap C_i), (\overline{S} \cap C_i) \rangle$. We deduce that there exists no 2-partition of C_i that satisfies both T and the edge $\langle u_j, u_k \rangle$. This gives an immediate contradiction to the fact that G is bipartite. ☐

The tester we introduce begins, as usual, by picking a subset $U \subset V$. Then it goes over all pair of connected components in $G[U]$, and finds a requirement for all pairs of components that are heavily connected. It then fixes a partition Π that is legal, with respect to all requirements found, and seeks a Π-violating vertex-pair. As Claim 4.14 proves, if G is bipartite then no Π-violating vertex-pairs exist. If G is ϵ-far from bipartiteness and U is good, then by Claim 4.3 we have that every 2-partition of U has $(\epsilon/3)N^2$ violating edges. We show that if we found a requirement for every pair of connected component that is heavily connected, then the number of the non Π-violating vertex-pairs is upper bounded by $(\epsilon/10)N^2$. Therefore, at least $(\epsilon/6)N^2$ vertex-pairs in $V \times V$ must be Π-violating. We note that, just as we did in Algorithm 14, when we go over all pairs of connected components of $G[U]$, we use the same sample set for all of them.

Let us formally introduce our next tester.

Algorithm 15 Testing Bipratitness by Forcing a Partition (1-Sided Version)

1: Pick a set $U \subset V$, using a hitter for sets of density at least $\epsilon/3$ and confidence at least $1-\epsilon/27$. Query $G[U]$, and let T be the set of requirements that the edges in $G[U]$ yield.
2: Select randomly a set $A \subset V \times V$ using a hitter for sets of density at least $\frac{\epsilon}{5|U|^2}$ and confidence at least $1 - \frac{1}{9} \cdot \frac{1}{\frac{1}{2}|U|^2}$. For every pair $(u, v) \in A$, query all potential edge connecting u,v and U, to see whether (u, v) forms a bypass between some pair of connected components in $G[U]$.
3: For every $(v_1, v_2) \in A$ that forms a bypass between some C_i and C_j, let $u_i \in C_i$ and $u_j \in C_j$ be the two vertices in U that (v_1, v_2) yield a requirement for. Add the requirement found to T.
4: Find the T-legal partition Π of U, by uniting components of $G[U]$ that were found to be connected in $R(U,T)$. If U has no 2-partition that satisfies T, output *"not bipartite"* and halt.
5: Fix an arbitrary 2-partition $\langle S, \overline{S} \rangle$ of U that satisfies T. Select randomly a set $B \subset V \times V$ using a hitter for sets of density at least $\epsilon/6$ and confidence at least $8/9$. For every vertex-pair $(u, v) \in B$ query all potential edges connecting u, v and U, to see whether (u, v) is a Π-violating pair.
6: If a Π-violating vertex-pair is found, then output *"not bipartite"*. Else output *"bipartite"*.

Theorem 4.15. *Algorithm 15 is a 1-sided bipartiteness tester.*

Proof. If G is bipartite, so is $G[U]$. Since all requirements in T were yield from edges or paths in G, as shown in Example 4.5, then every 2-coloring of $V(G)$ must satisfy T. By Claim 4.14 we have that the 2-partition $\langle S, \overline{S} \rangle$ that we fixed in Step 5, has no Π-violating edges, and so the tester outputs *"bipartite"*.

If G is ϵ-far from being bipartite, then due to Claim 4.3 and the discussion in Section 4.2, we have that the set U picked at Step 1 is good with probability at least 8/9. Due to Claim 4.3 this means that for any 2-partition of U there exist at least $(\epsilon/3)N^2$ violating edges in G. In particular, for S found in Step 5, the 2-partition$\langle S, \overline{S} \rangle$ has at least $(\epsilon/3)N^2$ violating edges.

Suppose the hitter of Step 2 did not fail. This means that we found a vertex-pair that forms a bypass for every pair of connected components of $G[U]$ that is heavily connected. Let us now look at any two sets in Π, which we denote as U_i and U_j. Denote U_i as a union of k_i connected components of $G[U]$, so we have $U_i = \bigcup_{l=1}^{k_i} C_{i_l}$ and denote U_j as a union of k_j *other* connected components, $U_j = \bigcup_{l'=1}^{k_j} C_{j_{l'}}$. Clearly, for every i_l and $j_{l'}$, the components C_{i_l} and $C_{j_{l'}}$ can *not* be heavily connected. Because we assume that the hitter of Step 2 did not fail, if C_{i_l} and $C_{j_{l'}}$ were heavily connected, then the tester would find a vertex-pair forming a bypass between C_{i_l} and $C_{j_{l'}}$, and they would have been united into a single connected component in $R(U,T)$. This would contradict the fact that Π is a T-legal partition.

Recall that a pair of connected components, C_i and C_j is heavily connected if the number of vertex-pairs that form a bypass between C_i and C_j is at least $\frac{\epsilon}{5|U|^2}N^2$. Look at all the violating edges of $\langle S, \overline{S} \rangle$, whose endpoint yield a requirement between a vertex from U_i and a vertex of U_j. We call these edges (i,j)-*interfering*. The (i,j)-interfering edges yield a requirement between some C_{i_l} and some $C_{j_{l'}}$, that are not heavily connected. Therefore, the number of the (i,j)-interfering edges is at most

$$\sum_{l=1}^{k_i}\sum_{l'=1}^{k_j}\frac{\epsilon}{5|U|^2}N^2 = k_i\cdot k_j\cdot\frac{\epsilon}{5|U|^2}N^2.$$

It follows (see details next) that the number of Π-violating pairs is at most $\frac{\epsilon}{10}\cdot N^2$. We assume Π has t sets, and recall that we denote by l the number of connected component in $G[U]$. We also denoted each $U_i \in \Pi$ to be the union of k_i connected components of $G[U]$. By going over all pair of sets in Π, we have that the number of violating edges for $\langle S,\overline{S}\rangle$ that are (i,j)-interfering for some i and some j is at most

$$\sum_{1\le i<j\le t} k_i\cdot k_j\left(\frac{\epsilon}{5|U|^2}N^2\right) \qquad \text{where } k_1+k_2+\ldots+k_t = l$$

Due to the concavity of the function $\sum_{1\le i<j\le t} k_i\cdot k_j$, we have that the maximum value is reached when $k_1 = k_2 = \ldots = k_t = \frac{l}{t}$. That means that the number of vertex-pairs that form a bypass between any pair of U_i, U_j is upper bounded by

$$\frac{\epsilon}{5|U|^2}N^2\cdot\binom{t}{2}\cdot\left(\frac{l}{t}\right)^2 \le N^2\cdot\frac{\epsilon\cdot t^2l^2}{5\cdot 2\cdot t^2|U|^2}\le\frac{\epsilon}{10}N^2$$

where the last inequality is due to the fact that $l \le |U|$.

Therefore, out of all violating edges for $\langle S,\overline{S}\rangle$, the number of edges whose endpoints are Π-violating is at least $\frac{\epsilon}{3}N^2 - \frac{\epsilon}{10}N^2 > \frac{\epsilon}{6}N^2$. Therefore, we have that if the hitter of Step 5 does not fail, then the tester hits a Π-violating vertex-pair, and so it outputs *"not bipartite"*. So, with probability at least $\left(1-\frac{1}{9}-\frac{1}{9}-\frac{1}{9}\right) = 2/3$ we have that the hitters of Steps 1,2 and 5 do not fail, the tester finds a Π-violating edge for Π, and output *"not bipartite"*. □

4.6.1 Implementation with a Hitter Linear in Confidence

As before, U will be of size $O(\epsilon^{-2})$ which means that Step 1 takes $O(\epsilon^{-4})$ queries, and n random bits. In Step 2 we take $s = \frac{5|U|^2}{\epsilon}(1/2)|U|^2 = O(\epsilon^{-9})$ samples, where for each sample we make $(2|U|+1) = O(|U|)$ queries to the edges of the graph, so in Step 2 we make $s\cdot O(|U|) = O(\epsilon^{-11})$ queries, and use $2n$ random bits. In Step 5 we take $O(\epsilon^{-1})$ samples, where for each we make $O(|U|)$ queries, meaning that in Step 5 we make $O(|U|/\epsilon) = O(\epsilon^{-3})$ queries, and use additional $2n$ random bits. In total we have query complexity of $O(\epsilon^{-11})$, and randomness complexity of $5n$.

Total Complexity:

Query Complexity: $O(\epsilon^{-11})$.
Randomness Complexity: $5n$.
Q-R Complexity: $O(\epsilon^{-11} \cdot n)$.

4.6.2 Implementation with a Hitter Logarithmic in Confidence

The size of U we pick is $\tilde{O}(\epsilon^{-1})$, thus the number of queries made in Step 1 is $\tilde{O}(\epsilon^{-2})$, while using $n + O(\log(1/\epsilon))$ random bits. The number of samples needed in Step 2 is $s = O(\frac{6|U|^2}{\epsilon}\log(|U|^2)) = \tilde{O}(\epsilon^{-3})$, where for each sample we make $O(|U|) = \tilde{O}(\epsilon^{-1})$ queries. So in Step 2 we query $s \cdot O(|U|) = \tilde{O}(\epsilon^{-4})$ edges in the graph, using $2n + O(\log(1/\epsilon))$ random bits. The number of samples in Step 5 is $O(\frac{1}{\epsilon} \cdot \log(1/6)) = O(\epsilon^{-1})$ and for each we do $O(|U|)$ queries, thus Step 5 takes $\tilde{O}(1/\epsilon^2)$ queries, and additional $2n + O(1)$ random bits. In total the number of queries is $\tilde{O}(\epsilon^{-4})$, and the randomness complexity is $5n + O(\log(1/\epsilon))$.

Total Complexity:
Query Complexity: $\tilde{O}(\epsilon^{-4})$.
Randomness Complexity: $5n + O(\log(1/\epsilon))$.
Q-R Complexity: $\tilde{O}(\epsilon^{-4}) \cdot n$.

4.7 Combining the Iterative Approach and the Concurrent Approach

We now introduce our last tester, which will achieve Q-R complexity of $\tilde{O}(\epsilon^{-3}) \cdot n$. This tester combines the previous iterative approach with the concurrent approach. The tester performs iterations, in which it seeks requirements that every 2-coloring of U must satisfy.

At every iteration, the tester looks at the requirements graph it currently has. Each component in the requirement graph is marked either "interesting" or "not-interesting", where initially, all component are interesting. It goes only over the *interesting* components in the requirements graph. For every interesting component C, it seeks a vertex-pair in $V \times V$ that yields a requirement between C and *any other* interesting component C', as shown in Example 4.5. If the tester finds a requirement that connects C with some C', then in the requirements graph of the next iteration, we have a single component $C \cup C'$ instead of the two separated C and C', and the tester marks this component as "interesting". If the tester finds no requirement that connects C with any other C', then C is marked as "not-interesting", and it will be disregarded in all next iterations. We will show that the tester either unites many components in the requirements graph, or it marks many components as "not-interesting". In any case, the number of interesting components decreases by a factor of $1/2$ from one iteration to the next. So after at most $\log(|U|)$ iterations, the tester remains with a requirement graph that has no interesting components, and the iterative part of the tester ends. Then the tester fixes a legal partition of U with respect to the requirements found in *all* iterations, and as in Algorithm 15, it seeks Π-violating vertex-pairs. By Claim 4.14, if

G is bipartite, then no Π-violating vertex-pairs exist. And as we will show, if G is ϵ-far from bipartiteness, then it is likely that at least $\frac{\epsilon}{6}N^2$ vertex-pairs are Π-violating.

As always, before we introduce the tester, we begin with some new notations and definitions. Denote a (n, m)-partition of U, as $\Pi = (V_1, V_2, \ldots, V_n; W_1, W_2, \ldots, W_m)$. This is a partition in the usual manner, where all sets are disjoint and their union is U $\left(\text{i.e. } U = \bigcup_{i=1}^{n} V_i \cup \bigcup_{j=1}^{m} W_j\right)$. The fact that Π has of two different types of sets, V-sets and W-sets, is to distinguish between the *interesting* sets of Π and the *non-interesting* sets. The interesting sets will be the V-sets, and the non-interesting sets will be the W-sets. Note that the notions from the previous section, of legal partitions (Definition 4.12), and of a Π-violating vertex-pairs (Definition 4.13), are notions that do not depend on the names of the sets in Π. Therefore, they apply to (n, m)-partitions, as they apply to any partition.

The notion of a vertex-pair that forms a bypass, presented in Definition 4.9, originally refers to connected components of $G[U]$. We now extend this definition to hold for any two arbitrary subsets of U.

Definition 4.16. *Let A and B be any two disjoint subsets of $V(G)$. We say a vertex-pair $(v_1, v_2) \in V(G) \times V(G)$ forms a* bypass *between A and B if there exists some $u_1 \in A$ and $u_2 \in B$, such that one of the two following hold:*

- *For some $v \in \{v_1, v_2\}$, we have that u_1 is adjacent to v and u_2 is adjacent to v. By Case 2 of Example 4.5, we have that (v_1, v_2) yield the requirement $u_1 \sim u_2$.*
- *We have that u_1 is adjacent to v_1, that u_2 is adjacent to v_2, and that v_1 and v_2 are adjacent. By Case 3 of Example 4.5, we have that (v_1, v_2) yield the requirement $u_1 \nsim u_2$.*

We denote by $CP(A, B)$, standing for connecting pairs, the set of all vertex-pairs that form a bypass between A and B. Given a (n, m)-partition Π for U, and a set $A \in \Pi$, we denote by $CP^{\Pi}(A)$ the set of all vertex-pairs that form a bypass between A and some other set $B \in \Pi$. That is, $CP^{\Pi}(A) = \bigcup_{\{B \in \Pi\colon\, B \neq A\}} CP(A, B)$.

We now introduce two definitions, which are crucial for understanding the new tester. They discuss properties for the interesting sets (i.e. V-sets) of Π, and the non interesting sets (i.e. W-sets) of Π.

Definition 4.17. *Let Π be a (n, m)-partition of U. Denote $\Pi = (V_1, V_2, \ldots, V_n; W_1, W_2, \ldots, W_m)$. Fix some $V_j \in \Pi$. We say that V_j is* heavily aligned *with Π, if the number of vertex-pairs in $V \times V$ that form a bypass between V_j and any other $V_k \in \Pi$ is at least $\frac{\epsilon}{12|U|}N^2$. That is, $\left|\bigcup_{k \neq j} CP(V_j, V_k)\right| \geq \frac{\epsilon}{12|U|}N^2$.*

Let us clarify the definition. Fix some V_j, and then look at the set of the vertex-pairs that form a bypass between V_j and some other V-set of Π, denoted V_k, where $V_k \neq V_j$. These vertex-pairs are exactly those in $\bigcup_{k \neq j} CP(V_j, V_k)$. Then V_j is said to be heavily aligned with

Π, if the number of all of these vertex-pairs is at least $\frac{\epsilon}{12|U|}N^2$. Note that the definition of a V-set being heavily aligned with Π *ignores* all the W-sets of Π, and regards solely the vertex pairs that form a bypass between V_j and other V-sets of Π.

Definition 4.18. *Let* Π *be a* (n, m)*-partition of* U. *Denote* $\Pi = (V_1, V_2, \ldots, V_n; W_1, W_2, \ldots, W_m)$. *We say that a vertex-pair* (v_1, v_2) *is* W-affecting *if for some* j *it forms a bypass between* W_j *and any other set* $B \in \Pi$*, where* $B \neq W_j$. *Formally,* (v_1, v_2) *is* W*-affecting, if there exists a* j *such that* $(v_1, v_2) \in CP^{\Pi}(W_j)$*, or alternatively, if* $(v_1, v_2) \in \bigcup_{j=1}^{m} CP^{\Pi}(W_j)$. *We call* Π *a* δ-bounded *partition, if the number of vertex-pairs in* $V(G) \times V(G)$ *that are* W*-affecting, is at most* δN^2. *That is,* $\left|\bigcup_{j=1}^{m} CP^{\Pi}(W_j)\right| \leq \delta N^2$.

Let us emphasize which vertex-pairs we discuss here, or more formally, which vertex-pairs are in $\bigcup_{j=1}^{m} CP^{\Pi}(W_j)$. These vertex-pairs close a path of length 2 or 3 between some two vertices u_1 and u_2 in U, where u_1 belongs to some W_j, and u_2 belongs to any other set $B \in \Pi$ which isn't W_j, meaning $B \in \Pi \setminus \{W_j\}$. Again we stress: the set $\bigcup_{j=1}^{m} CP^{\Pi}(W_j)$ does *not* include vertex-pairs that close a path of length 2 or 3 only between some two vertices u_1 and u_2 in U, where *both* u_1 and u_2 belong to the *same* W_j. Note the the definition of a partition that is δ-bounded ignores all vertex-pairs that form a bypass between two V-sets.

Let us describe our tester informally. At the beginning of iteration i it has a set of requirements found thus far, T^i, and a (n, m)-partition Π^i of U which is T^i-legal. It focuses *only* on the V^i-sets of Π^i. For every V_j^i, it tries to hit a vertex-pair that belongs to $\bigcup_{k \neq j} CP(V_j^i, V_k^i)$. If V_j^i is heavily aligned with Π^i, then a vertex-pair inducing a requirement for V_j^i and some other V_k^i is likely to be found, and the requirement this vertex-pair yield is added to T^i (towards forming T_{i+1}). If V_j^i is not heavily aligned with Π, then it may "cast" V_j^i aside, and in the next iteration, V_j^i will be a non-interesting W^{i+1}-set, which is ignored in all of the following iterations. Once the tester reaches an iteration i where Π^i has no V^i-sets to find requirements for, then it fixes some 2-partition $\langle S, \overline{S} \rangle$ of U that satisfies T^i. It then seeks a Π^i-violating vertex-pair. If G is bipartite, then as Claim 4.14 shows, no Π^i-violating edges exist. If G is ϵ-far from bipartiteness and all the hitters that the tester uses do not fail, then, as we show, after at most $\log(|U|)$ iteration, the tester has a T^i-legal partition Π^i, for which that number of Π^i-violating vertex-pairs is at least $(\epsilon/6)N^2$.

Note that as in Algorithm 14 and Algorithm 15, at each iteration we use the same sample set to check for all V-sets together, whether each V-set is heavily aligned with Π or not. Here, however, like in Algorithm 13, at each iteration we use new random bits, independent from the random bits that were used in the previous iterations. Formally, the tester is as follows.

Algorithm 16 Bipartiteness Tester Combining Both Approaches

1: Pick a set $U \subset V$, using a hitter for sets of density at least $\epsilon/3$ and confidence at least $1-\epsilon/27$. Query $G[U]$.

2: Denote the connected components of U to be $C_1, \dots, C_l$. If any component isn't bipartite, output *"not bipartite"* and halt. Let T^0 be the set of requirements that the edges of $G[U]$ yield. Let Π^0 be the $(l, 0)$-partition: $(C_1, C_2, \dots, C_l; \emptyset)$, which is T^0-legal.

3: **for** $i = 0, 1, \dots t = \log_2(|U|)$ **do**

4: Denote the set of requirement found thus far by T^i, and the legal T^i-partition by $\Pi^i = \left(V_1^i, \dots, V_{n_i}^i; W_1^i, \dots, W_{m_i}^i\right)$.

5: Select randomly a set $B^i \subset V(G) \times V(G)$ using a hitter for sets of density at least $\frac{\epsilon}{12|U|}$ and confidence at least $1 - \frac{1}{9n_i \log(|U|)}$. For every $(u, v) \in B^i$ query all potential edges between u, v and $V_1^i, V_2^i, \dots, V_{n_i}^i$ to see whether (u, v) forms a bypass between some two sets, V_j^i and V_k^i (possibly $V_j^i = V_k^i$).

6: For every vertex-pair $(v_1, v_2) \in B^i$ that forms a bypass between V_j^i and V_k^i, find the requirement they yield over some two vertices $u_j \in V_j^i$ and $u_k \in V_k^i$. Unite T^i with all requirements found in the ith iteration, into a new set of requirements, T^{i+1}.

7: Let Π^{i+1} be the T^{i+1}-legal partition,[12] where every connected component of $R(U, T^{i+1})$ is a set in Π^{i+1}. Divide Π^{i+1} to V-sets and W-sets as follows:

- If C is a component of $R(U, T^{i+1})$ that is the union of two or more V-sets of Π^i, then it is a V-set in Π^{i+1}.
- If C is a component of $R(U, T^{i+1})$ that was a W-set in Π^i, then it is a W-set in Π^i.
- If C is a component of $R(U, T^{i+1})$ that was a V-set in Π^i, yet no requirement connecting it with any other component in $R(U, T^i)$ was found, then it is a W-set in Π^{i+1}.

8: **end for**

9: Denote the set of requirements T^{t+1} and the T^{t+1}-legal partition Π^{t+1} that were defined in the last iteration, as T^f and Π^f, respectively. These are the final set of requirements, and the final (n, m)-partition, that the iterations part of the tester ends with. If no 2-partition of U satisfies T^f, then output *"not bipartite"* and halt.

10: Fix an arbitrary 2-partition $\langle S, \overline{S} \rangle$ that satisfies all requirements in T^f. Select randomly a set $C \subset V(G) \times V(G)$ using a hitter for sets of density at least $\epsilon/6$ and confidence at least $8/9$. For every $(u, v) \in C$, query all potential edge between u, v and U, to see whether $\langle u, v \rangle$ is a $\langle S, \overline{S} \rangle$-violating edge, whose endpoints are Π^f-violating.

11: If a Π^f-violating vertex-pair is found, output *"not bipartite"*, otherwise, output *"bipartite"*.

Theorem 4.19. *Algorithm 16 is a 1-sided tester for bipartiteness:*

[12] Recall that by Definition 4.12, legal partitions refer solely to the components in $R(U, T)$. Therefore, a T^{i+1}-legal partition always exists, regardless of whether there exists a 2-partition of U that satisfies T^{i+1} or not.

Proof. The analysis of the algorithm is based on the following claim:

Claim 4.20. *For every iteration i:*

1. *n_i, the number of V-sets in Π^i, is at most $\left(\frac{1}{2}\right)^i |U|$.*
2. *If all the hitters in all iterations did not fail, then Π^i is a $\left(\epsilon \cdot \frac{1-(1/2)^i}{6}\right)$-bounded partition.*

We first show how Claim 4.20 implies the validity the theorem. If G is bipartite, then for every iteration i, any 2-coloring of $V(G)$ must satisfy the set of requirements T^i. Therefore, any 2-coloring of U induced from a 2-coloring of $V(G)$ must satisfy T^i. By Claim 4.20, after at most $\log(|U|)$ iterations, we have a set of requirements T^f and a T^f-legal partition, Π^f, such that the number of V-sets in Π^f is at most 1. No matter which edges and vertex-pair Algorithm 16 picks in Step 5, the set T^f that they yield is satisfied by any 2-coloring of $V(G)$. Since Π^f is a T^f-legal partition, then by Claim 4.14, we have that Π^f has no Π^f-violating vertex-pairs. Thus the tester never finds any Π^f-violating vertex-pair, and it must output *"bipartite"*.

Let us now show that for any G that is ϵ-far from being bipartite, the tester outputs *"not bipartite"* with probability at least 2/3. Fix any G which is ϵ-far from being bipartite. Assume that none of the hitters used in all of the stages failed. (The probability that one failed is upper bounded by: $\frac{1}{9} + \sum_{i=0}^{\log(|U|)} \frac{1}{9n_i \log(|U|)} n_i + \frac{1}{9} = 1/3$.) Let us show that if this is the case, the tester doesn't fail.

As shown in Claim 4.3 and in Section 4.2, if the hitter of the first Step did not fail, we have a good set U. By Claim 4.3, we have that that every 2-partition of U has at least $(\epsilon/3)N^2$ violating edges. In particular, we have that the 2-partition that Algorithm 16 fixes in Step 10, $\langle S, \overline{S}\rangle$, has at least $(\epsilon/3)N^2$ violating edges.

Since we assume Claim 4.20 holds, then after at most $\log(|U|)$ iterations the tester finds a partition with at most one V-set, and quits doing iterations. Thus, Π^f is a partition with no heavily aligned components, because it has at most one V-set. Look at the endpoints of a violating edge of $\langle S, \overline{S}\rangle$ that are not Π^f-violating. By definition, this vertex-pair forms a bypass between two different sets of Π^f. Since there is at most one V-set in Π^i, we deduce that one of the sets this vertex-pair forms a bypass for, is a W-set of Π^f. Therefore, this is a W-affecting vertex-pair. By Claim 4.20, we have that Π^i is a $\epsilon/6$-bounded partition, so the number of W-affecting vertex-pairs for Π^f is at most $(\epsilon/6)N^2$. We deduce that the number of Π^i-violating vertex-pairs is at least $\left(\frac{\epsilon}{3} - \frac{\epsilon}{6}\right)N^2 = \frac{\epsilon}{6}N^2$. We assume the last hitter also doesn't fail, so it hits one Π^f-violating vertex-pair. Therefore, if G is ϵ-far from bipartite, all hitters did not fail, and Claim 4.14 holds, then the tester outputs *"not bipartite"*. □

Proof of Claim 4.20. We show that the claim holds by induction. Obviously, Π^0 has $l \leq |U|$ components, and since by definition Π^0 has no W-sets, then there exists no W-affecting vertex-pair for Π^0, so Π^0 is trivially a 0-bounded partition.

We now prove the induction step for Item 1. That is, assuming that $n_i \leq 2^{-i}|U|$, we show that $n_{i+1} \leq 2^{-(i+1)}|U|$. We know that the number of V-sets in Π^i is n_i. Take H to be a graph over n_i vertices, each represents a V^i-set in Π^i. In Step 5 of Algorithm 16, we find vertex-pairs that form a bypass between two different V-sets. For each pair of V-sets, V^i_j and V^i_k, that a bypass was found for, add an edge to H that connects the two vertices representing V^i_j and V^i_k. Recall that each vertex-pair that forms a bypass between V^i_j and V^i_k, yields a requirement that connects V^i_j and V^i_k in the requirements graph $R(U, T^{i+1})$. Therefore, the edge we add to H indicates that V^i_j and V^i_k are united into a single V-set in Π^{i+1}. After adding all edges to H, every vertex that remains isolated from the rest of H represents a V-set of Π^i that turns into a W-set in Π^{i+1}. This means that the number of V-set in Π^{i+1} is the number of components in H that are of size at least 2. Since H has n_i vertices, then at most it has $\frac{n_i}{2}$ connected components of size at least 2. This means that the number of V-sets in Π^i is at most $\frac{1}{2} \cdot 2^{-i}|U| = 2^{-(i+1)}|U|$.

We now prove the induction step of Item 2. That is, assuming that Π^i is a $\left(\epsilon \cdot \frac{1-2^{-i}}{6}\right)$-bounded partition, we show that Π^{i+1} is a $\left(\epsilon \cdot \frac{1-2^{-(i+1)}}{6}\right)$-bounded partition. Recall that Π^i is δ-bounded if the number of W-affecting vertex-pairs is at most δN^2. Also recall that a vertex-pair is W-affecting if it yields a requirement for two vertices u_1 and u_2 in U, where u_1 belongs to some $W^i_j \in \Pi^i$ and u_2 belongs to any set $B \in \Pi^i \setminus \{W^i_j\}$.

First, look at the W-sets of the partition Π^{i+1}. They can be classified into two types. The first type is of W-sets that were W-sets already in Π^i, and we call them the *old* W-sets. The second type is of W-sets that were V-set in Π^i, and we call them the *new* W-sets. This classification helps us to classify the vertex-pairs in $\bigcup_j CP^{\Pi}(W^{i+1}_j)$ into two types, and thus upper bounds their number.

Let (v_1, v_2) be a vertex-pair that is W-affecting, for the W-sets of Π^{i+1}. As in the definition of a vertex-pair that forms a bypass, Definition 4.16, we assume v_1 and v_2 close a path of length 2 or 3 between some two different vertices of U, denoted u_1 and u_2. Since (v_1, v_2) is W-affecting, then either u_1 or u_2 belong to some W-set in Π^{i+1}. Since we have old W-sets and new W-sets, then we have two cases:

Case 1: Either u_1 or u_2 belong to some old W-set. Then in this case, we have that the vertex-pair (v_1, v_2) forms a bypass between some W^i_j set in Π^i, and some other set of Π^i. Therefore, (v_1, v_2) is a vertex-pair that is W-affecting for the W-sets of Π^i.

Case 2: Both u_1 and u_2 do not belong to an old W-set, and one of them belongs to a new W-set. Without loss of generality, u_1 belongs to a new W-set, which in Π was a V-set, denoted V^i_j. Since u_2 also does not belong to an old W-set, then it must belong to some other V-set in Π^i. This means that the vertex-pair (v_1, v_2) forms a bypass between V^i_j and some other V-set of Π^i.

If V_j^i were heavily aligned with Π^i, then $\bigcup_{k\neq j} CP(V_j^i, V_k^i)$ would be a set of size at least $\frac{\epsilon}{12|U|}N^2$. By the assumption that the hitter of the ith iteration did not fail, we would hit a vertex-pair in $\bigcup_{k\neq j} CP(V_j^i, V_k^i)$. Thus V_j^i would be united with some other V_k^i into a new V-set in Π^{i+1}. But since V_j^i is a W-set in Π^{i+1}, we have a contradiction. We deduce that V_j^i isn't heavily aligned with Π^i.

Therefore, Π^{i+1} has that the W-affecting vertex-pairs are of two types. Type one is of vertex-pairs that were W-affecting for Π^i. By the induction hypothesis, we have at most $\epsilon \cdot \frac{1-2^{-i}}{6}N^2$ vertex-pairs of type one. Type two is of vertex-pairs that form a bypass between two V-sets of Π^i, where one V-set is not heavily aligned with Π^i. Each non heavily aligned V_j^i set of Π^i contributes at most $\frac{\epsilon}{12|U|}N^2$ vertex-pair that form a bypass between V_j^i and some other V-set of Π^i. Since the number of V-sets in Π is n_i, then we have at most $n_i \cdot \frac{\epsilon}{12|U|}N^2$ vertex-pairs of type two. Since Item 1 holds, the number of V-sets in Π^i is $n_i \leq 2^{-i}|U|$. Therefore, we can upper bound the number of W-affecting vertex-pairs, for Π^{i+1}, by

$$\left(\epsilon \cdot \frac{1-2^{-i}}{6}N^2\right) + n_i \cdot \left(\frac{\epsilon}{12|U|}N^2\right) \leq \frac{\epsilon}{6}N^2 \cdot \left(1 - 2^{-i} + \frac{2^{-i}|U|}{2|U|}\right) =$$
$$\frac{\epsilon}{6}N^2\left(1 - (1-\frac{1}{2}) \cdot 2^{-i}\right) = \epsilon \cdot \frac{1-2^{-(i+1)}}{6}N^2$$

Item 2 is proven, and we are done. □

4.7.1 Implementation with a Hitter Linear in Confidence

As before, U will be of size $O(\epsilon^{-2})$ which means that in Step 1 we make $O(\epsilon^{-4})$ queries and use n random bits. Each time we performs Step 5, it requires $2n$ random bits to produce $s = O(\frac{12|U|}{\epsilon} \cdot 9n_i \log(|U|)) = O(|U|^2 \log(|U|)/\epsilon) = O(\frac{\log(1/\epsilon)}{\epsilon^5})$ samples of vertex-pairs. For each vertex-pair we make $(2|U|+1)$ queries to the edges of the graph, meaning each iteration takes $s \cdot O(|U|) = O(\frac{\log(1/\epsilon)}{\epsilon^7})$ queries and $2n$ random bits. We do at most $t = \log(|U|)$ iterations, each time with new random bits, so the iterations part of the algorithm we use $t \cdot 2n = 2\log(1/\epsilon) \cdot n$ random bits, to make $s \cdot (2|U|+1) \cdot t = O(\frac{\log^2(1/\epsilon)}{\epsilon^7})$ queries to the edges of the graph.

In Step 10, we use additional $2n$ random bits to produce another $O(\frac{1}{\epsilon})$ samples, for each we make $O(|U|)$ queries, which means we make $O(|U|/\epsilon) = O(\epsilon^{-3})$ queries. All in all, we make $O(\frac{\log^2(1/\epsilon)}{\epsilon^7})$ queries, and use $n + t \cdot (2n) + 2n = O(\log(1/\epsilon) \cdot n)$ random bits.

Total Complexity:
Query Complexity: $O(\frac{\log^2(1/\epsilon)}{\epsilon^7})$.
Randomness Complexity: $O(\log(1/\epsilon) \cdot n)$.
Q-R Complexity: $O(\frac{\log^3(1/\epsilon)}{\epsilon^7} \cdot n)$.

4.7.2 Implementation with a Hitter Logarithmic in Confidence

The size of U we pick is $\tilde{O}(\epsilon^{-1})$, thus in Step 1 we use $n+O(\log(1/\epsilon))$ random bits and make $\tilde{O}(\epsilon^{-2})$ queries.

Each time we perform Step 5, it requires $2n + O(\log(n_i \log(|U|))) = 2n + \tilde{O}(\log(1/\epsilon))$ random bits, to produce $s = O\left(\frac{|U|}{\epsilon} \cdot \log(n_i \log(|U|))\right) = \tilde{O}(\epsilon^{-2})$ samples of vertex-pairs. For each sample we make $(2|U|+1)$ queries to the edges of the graph, meaning each iteration takes $s \cdot O(|U|) = \tilde{O}(\epsilon^{-3})$ queries. We do at most $t = \log(|U|)$ iterations, so in the iterations part of the algorithm, we make $s \cdot O(|U|) \cdot t = \tilde{O}\left(1/\epsilon^3\right)$ queries, and uses $(2n+\tilde{O}(\log(1/\epsilon))) \cdot \log(|U|) = O(\log(1/\epsilon)) \cdot n$ random bits.

In Step 10, we use $2n + O(1)$ random bits to take another sample of $O(\frac{1}{\epsilon})$ vertex-pairs, where for each sample we make $O(|U|)$ queries, So in step 10 we make $\tilde{O}(\epsilon^{-2})$ queries. All in all, we make $\tilde{O}\left((1/\epsilon)^3\right)$ queries and use $n + t \cdot n + 2n + O(1) = O(\log(1/\epsilon)) \cdot n$ random bits.

Total Complexity:

Query Complexity: $\tilde{O}(\epsilon^{-3})$.

Randomness Complexity: $O(\log(1/\epsilon)) \cdot n$.

Q-R Complexity: $\tilde{O}\left(\epsilon^{-3}\right) \cdot n$.

4.8 Summation

In this section we have introduce several algorithms for testing bipartitness. We now sum up their complexities:

Tester	Query Complexity	Randomness Complexity	Q-R Complexity
Algorithm 12: Original GGR	$\tilde{O}(\epsilon^{-3})$	$\tilde{O}(\epsilon^{2}n)$	$\tilde{O}(\epsilon^{-5}) \cdot n$
GGR + hitter linear in confidence	$2^{O(\epsilon^{-2})}$	$3n$	$2^{O(\epsilon^{-2})} \cdot n$
GGR + hitter logarithmic in confidence	$\tilde{O}(\epsilon^{-3})$	$3n + \tilde{O}(\epsilon^{-1})$	$\tilde{O}(\epsilon^{-3}) \cdot n + \tilde{O}(\epsilon^{-4})$
Algorithm 13: Sequential Forcing	$\tilde{O}(\epsilon^{-3})$	$\tilde{O}(\epsilon^{-1}n)$	$\tilde{O}(\epsilon^{-4}) \cdot n$
Algorithm 15: Concurrent Forcing	$\tilde{O}(\epsilon^{-4})$	$5n + O(\log(\epsilon^{-1}))$	$\tilde{O}(\epsilon^{-4}) \cdot n$
Algorithm 16: Combining concurrent and sequential	$\tilde{O}(\epsilon^{-3})$	$O(\log(\epsilon^{-1})) \cdot n$	$\tilde{O}(\epsilon^{-3}) \cdot n$

5 Conclusions and Open Problems

Again, we would like to stress the importance of reducing the randomness complexity of property testers. Besides the fact that saving resources is a good paradigm, we have shown that when using a single weak random source, the randomness complexity of a tester affects the number of queries it makes.

Also, we believe that the techniques presented in Section 4 are of importance for other testers as well. In particular, we would like to stress the following notion. When faced with an exponential number of events, the standard way to rule out all of them is to reduce the probability of each event to be exponentially small. Instead, try and gather more information by querying other entries of the input, and reduce the number of bad events from exponential to polynomial. This approach requires only slightly more queries, and may reduce significantly the randomness complexity. It might even be possible that once can fix one event, with the following property: If the input is a "Yes" instance, then this event is likely to have relatively few refutations (or none at all). If the input is a far from "Yes" instance, then this event has many refutations. If this is to hold, then one can use a randomness-efficient sampler (or a hitter) to estimate the number of refutations this specific event has.

We end with open questions. The most important open question this thesis raises is how to explicitly find a set U that is (K, δ)-universal (see Definition 3.5). We made some attempts in that direction, following the work of Nisan [Nis90] and of Nisan and Zuckerman [NZ96], but were unsuccessful. The approach we tried was to think of a canonical property tester for some Π as a probabilistic space bounded TM that gets $K \log N$ random bits, indicating the K vertices of the subgraph that the canonical tester checks. In order to check whether the subgraph over these K vertices has some property Π', we need to save this subgraph into the tester's workspace and then use a lookup table to determine whether to accept or reject the input. Saving this subgraph requires $O(K^2)$ space. Note, however, that in order to query all *pairs of vertices* of the given K vertices, the tester has to either save its entire $K \log N$ random bits into its workspace, or read the random tape several times, instead of just once. In the first case, the workspace of the machine is of size at least $K \log N$, and so the known PRGs for such machines have seeds of length at least $K \log N$. In the second case, the known PRGs are for space bounded machines with *one-time* access to their random tape, and are not adaptable to machines that read the random tape several times. Still, it is tempting to consider property testing (and not just canonical testers for graphs, but also other testers fall into that category) as a type of space bounded machines, that only store the result of their queries to their workspace, and thus determining their output. If the locations of the entries that the tester queries were written explicitly on the random tape, then property testing would fall in the known framework of space bounded computations. However, the tester computes its queries from the random tape, and this computation either requires too much workspace, or requires that we read the random tape several times. In both cases, applying the space-bounded model, in a straightforward way, fails.

A second question is to try and find a tighter lower bound for the randomness complexity of samplers and property testers. In particular, we wish to add a term of $2\log(1/\epsilon)$ to the known lower bound of samplers. It is not clear what is the connection between the randomness complexity and the accuracy of the sampler/distance parameter of the tester (the parameter ϵ). Perhaps some "translation" between the term of "entropy loss" in extractors to the field of samplers is required.

Finally, we believe it to be very interesting, yet very difficult, to prove some bounds on the Q-R complexity of property testers, with emphasis on a lower bound (other than the trivial multiplication of the lower bound on Q with the lower bound on r). Perhaps techniques related to space-time tradeoffs are applicable towards finding such a lower bound.

6 Acknowledgements

I would like to thank wholeheartedly my advisor, Prof. Oded Goldreich, for his helpful ideas and suggestions, remarkably useful comments, endless patience with me, and the tremendous amount of time and effort he has put in this thesis. I consider myself extremely honored and privileged to work under his supervision.

References

[AK02] Noga Alon and Michael Krivelevich. Testing k-colorability. *SIAM J. Discret. Math.*, 15(2):211–227, 2002.

[BGG93] Mihir Bellare, Oded Goldreich, and Shafi Goldwasser. Randomness in interactive proofs. *Comput. Complex.*, 3(4):319–354, 1993.

[CEG95] Ran Canetti, Guy Even, and Oded Goldreich. Lower bounds for sampling algorithms for estimating the average. *Information Processing Letters*, 53(1):17–25, 1995.

[GGR98] Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *J. ACM*, 45(4):653–750, 1998.

[Gol97] Oded Goldreich. A sample of samplers - a computational perspective on sampling (survey). *Electronic Colloquium on Computational Complexity (ECCC)*, 4(020), 1997.

[GR98] Oded Goldreich and Dana Ron. A sublinear bipartiteness tester for bounded degree graphs. In *STOC '98: Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 289–298, New York, NY, USA, 1998. ACM Press.

[GR02] Oded Goldreich and Dana Ron. Property testing in bounded degree graphs. *Algorithmica*, 32(2):302–343, 2002.

[GT01] Oded Goldreich and Luca Trevisan. Three theorems regarding testing graph properties. In *IEEE Symposium on Foundations of Computer Science*, pages 460–469, 2001.

[GUV06] Venkatesan Guruswami, Christopher Umans, and Salil Vadhan. Extractors and condensers from univariate polynomials. *Electronic Colloquium on Computational Complexity (ECCC)*, 13, 2006.

[LRVW03] C. Lu, O. Reingold, S. Vadhan, and A. Wigderson. Extractors: Optimal up to constant factors, 2003.

[Nis90] N. Nisan. Pseudorandom generators for space-bounded computations. In *STOC '90: Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 204–212, New York, NY, USA, 1990. ACM Press.

[NZ96] Noam Nisan and David Zuckerman. Randomness is linear in space. *Journal of Computer and System Sciences*, 52(1):43–52, 1996.

[PR02] Michal Parnas and Dana Ron. Testing the diameter of graphs. *Random Structures and Algorithms*, 20(2):165–183, 2002.

[RTS97] Jaikumar Radhakrishnan and Amnon Ta-Shma. Tight bounds for depth-two superconcentrators. In *IEEE Symposium on Foundations of Computer Science*, pages 585–594, 1997.

[Zuc97] David Zuckerman. Randomness-optimal oblivious sampling. *Random Structures and Algorithms*, 11(4):345–367, 1997.

[Zuc06] David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. In *STOC '06: Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 681–690, New York, NY, USA, 2006. ACM Press.

A Samplers

A.1 Basic Definitions and Notations

Let N be a positive integer, and denote $n = \log_2 N$.Let $f : [N] \to [0,1]$ be any function. Denote $E[f] = avg(f) = \frac{1}{N}\sum_{x\in[N]} f(x)$.

Definition A.1. *A* (ϵ, δ)-sampler *S with query complexity Q and randomness complexity r, is a probabilistic oracle machine that for every n and every* $f : [N] = \{0,1\}^n \to [0,1]$ *tosses at most r coins and queries no more than Q of the entries of f, in order to output an estimation* $Est_S(f)$ *that satisfies*

$$Pr[|Est_S(f) - E[f]| > \epsilon] < \delta.$$

We call ϵ the *accuracy* of the sampler, and $1-\delta$ the *confidence* of the sampler.

We wish to stress that a sampler can be adaptive, meaning that its queries may depend on the function values it already queried. The number of queries made by a sampler may also vary according to its sample points and the function values over these points, but we assume that the number of queries never exceeds Q.

We call a sampler *oblivious* if it is non-adaptive and its estimation doesn't depend on the labeling of the Q points it queries. We call a sampler *averaging* if it is an oblivious tester whose estimation is $Est_S(f) = \frac{1}{Q}\sum_{i=1}^{Q} f(x_i)$.

Similarly, we define a $\{0,1\}$-sampler, of the same parameters, to be a sampler that for every $f : [N] \to \{0,1\}$, produces an ϵ-close estimation of $E[f]$, with confidence at least $1-\delta$.

We classify samplers into two major kinds, according to the dependence of their sample complexity in their confidence. To be exact, the dependence is between Q, the query complexity of the sampler, to δ, which is the complementary of the confidence of the sampler to 1 (or, even more formally, between Q and $1/\delta$).[13]

Definition A.2. *We say that a sampler S is* polynomial in confidence *if its sample complexity is proportional to* $poly(1/\delta)$*, and* linear in confidence *if it is proportional to* $O(1/\delta)$*. We say that a sample S is* logarithmic in confidence *if its sample complexity is proportional to* $O(\log(1/\delta))$.

The subject of samplers have been extensively studied, and there are many known samplers (See [Gol97] for survey, [Zuc97] for the connection between samplers and extractors

[13] We apologize for the confusion that may arise from the term dependence "in confidence". We assume however that the average reader if familiar with the common terminology, and understands the choice of words.

and [GUV06] for the best known construction of extractors). They vary from very simple samplers to extremely non-trivial one. Here, we wish to present just two samplers, one linear in confidence, and the other logarithmic in confidence.

A.2 Lower and Upper Bounds for Samplers

Before introducing our samplers, we wish to first discuss the known lower bounds and upper bounds on the query complexity and the randomness complexity of samplers.

The work of Canetti et al. [CEG95] gave a lower bound of $\Omega(\frac{1}{\epsilon^2}\log(\frac{1}{\delta}))$ on the query complexity of *any* sampler. This lower bound is matched by a simple sampler that performs $O(\frac{1}{\epsilon^2}\log(\frac{1}{\delta}))$ queries: The sampler that selects randomly, independently and uniformly Q points in N, and averages of the function values over these points. The Chernoff bound shows that it suffices to take $O(\frac{1}{\epsilon^2}\log(\frac{1}{\delta}))$ samples in order to produce an ϵ-close estimation of $E[f]$ with probability greater than $1-\delta$.

The same paper [CEG95] gave also a lower bound of $n-\log Q+\log(1/\delta)+\log(1-2\epsilon)-2$ on the randomness complexity of *any* sampler having query complexity Q. Since we may assume $\epsilon < 1/4$, we can intuitively interpret this lower bound as $n - \log Q + \log(1/\delta) - O(1)$. The trivial sampler that queries Q entry points, chosen randomly, independently and uniformly, has randomness complexity of $Q \cdot n = O(\frac{1}{\epsilon^2}\log(\frac{1}{\delta})) \cdot n$, which is very far from the lower bound. Zuckerman [Zuc97] introduced a sampler with randomness $(1+\alpha)(n+\log(1/\delta))$ for any $0 < \alpha < 1$. However, the query complexity of Zuckerman's sampler is far from the optimal, and depends on n.

Ultimately, we would like to construct a sampling algorithm, that both its query complexity and its randomness complexity match the lower bounds. Here, we construct a sampler that has the optimal query complexity of $O(\frac{1}{\epsilon}\log(1/\delta))$, and is very close to having an optimal randomness complexity - it has randomness complexity of $n + O(\log(1/\delta))$.

A.3 A Sampler Linear in Confidence

Let us introduce a new-old sampler. This sampler is linear in confidence, and has already been proven [Gol97] to be a $\{0,1\}$-sampler. Here we apply almost the same method to show that this is also a sampler, in the standard sense.

We wish to emphasize the randomness complexity of this sampler. It has randomness complexity $r = n$, so this sampler is as efficient as picking one sample point in N uniformly. Furthermore, for samplers linear in confidence (i.e. Q is proportional to $\frac{1}{\delta}$), the lower bound on the randomness of samplers linear in confidence is $r \geq n - \log Q + \log(1/\delta) - O(1) = n - O(1)$. This means that this sampler exactly meets the lower bound for samplers linear in confidence without any constant factor (as apposed to, say, the pair-wise independent sampler, which has randomness complexity of $2n$).

Let us now describe the sampling algorithm. Fix any $\epsilon, \delta > 0$. Let G be a (N, D, λ)-expander, where $D = O\left(\frac{1}{\epsilon^2\delta}\right)$ and $\frac{\lambda}{D} = O\left(\frac{1}{\sqrt{D}}\right)$. For any $v \in [N]$, denote the ith neighbor of v as $\Gamma_i(v)$.

Algorithm 17 Sampling Algorithm 1

1: Select randomly and uniformly a vertex $v \in_R [N] = [2^n] = \{0,1\}^n$.
2: Query f for the following D values: $f(\Gamma_1(v)), f(\Gamma_2(v)), \ldots, f(\Gamma_D(v))$.
3: Output $Est_v(f) = \frac{1}{D}\sum_{i=1}^{D} f(\Gamma_i(v))$.

Theorem A.3 (Expander Neighbors Sampler). *$\forall n, \epsilon, \delta > 0$, Algorithm 17 is a (ϵ, δ)-averaging sampler, with query complexity of $O\left(\frac{1}{\epsilon^2\delta}\right)$ and randomness complexity of n.*

Proof. First, let us state the trivial properties of Algorithm 17. Randomness-wise, all Algorithm 17 needs, is to select one vertex uniformly from G, therefore its randomness complexity is $r = n$. The query complexity of Algorithm 17 is $D = O\left(\frac{1}{\epsilon^2\delta}\right)$, since it queries all D neighbors of the vertex selected. By definition, Algorithm 17 outputs the average of its sampling points. So, all that is left for us to show is that Algorithm 17 is a (ϵ, δ)-sampler. That is, we wish to show that indeed $Pr_{v \leftarrow U_r}[|Est_v(f) - E[f]| > \epsilon] < \delta$.

Denote by B^+ the set of $\{v \in [N] : Est_v(f) - E[f] > \epsilon\}$, and $B^- = \{v \in [N] : Est_v(f) - E[f] < -\epsilon\}$. Our goal is to upper bound $|B^+ \cup B^-| = |B^+| + |B^-|$. We begin with bounding $|B^+|$.

Denote by b the indicating vector of B^+, and let a be the vector where $a_x = f(x)$. Both a and b are of length N. Let M the normalized adjacency matrix of G. Naturally, M is symmetric and thus has a basis of orthonormal eigenvectors, which we will denote as $\{u_1, u_2, \ldots, u_N\}$. Their matching eigenvalues are denotes as $\mu_1, \mu_2, ..., \mu_N$ respectively, where $|\mu_1| \geq |\mu_2| \geq ... \geq |\mu_N|$. As usual, $u_1 = \frac{1}{\sqrt{N}}\bar{1}$ and $\mu_1 = 1$, and by our notation $\mu_2 = \lambda/D$.

Let us look at $b^T Ma$. Since $M_{x,y} = 1/D$ for any x, y such that $y \in \Gamma(x)$ and otherwise $M_{x,y} = 0$, we have that $(Ma)_v = \sum_{i=1}^{D} \frac{a_{(\Gamma_i(v))}}{D} = Est_v(f)$, thus, by the definition of the set B^+,

$$b^T Ma = \sum_{v \in B^+} Est_v(f) \geq |B^+| \cdot (E[f] + \epsilon)$$

which implies that

$$\frac{b^T Ma}{N} - \frac{B^+}{N} E[f] \geq \rho(B^+) \cdot \epsilon \tag{i}$$

On the other hand, using the orthonormal basis, we can represent $a = \sum_x \hat{a_x} u_x$ and $b = \sum_x \hat{b_x} u_x$, where $\hat{a_x} = \langle a, u_x \rangle, \hat{b_x} = \langle b, u_x \rangle$. Therefore

$$
\begin{aligned}
b^T M a = \left(\sum_y \hat{b_y} u_y^T\right) M \left(\sum_x \hat{a_x} u_x\right) = \left(\sum_y \hat{b_y} u_y^T\right)\left(\sum_x \hat{a_x}\mu_x u_x\right) = \sum_{x,y} \hat{a_x}\hat{b_y}\mu_x u_y^T u_x = \\
= \sum_x \hat{a_x}\hat{b_x}\mu_x \le \hat{a_1}\hat{b_1} + |\mu_2| \left|\sum_{x=2}^{|N|} \hat{a_x}\hat{b_x}\right| \overset{(1)}{\le} \hat{a_1}\hat{b_1} + |\mu_2| \|\hat{a}\|_2 \|\hat{b}\|_2 \\
\overset{(2)}{=} \hat{a_1}\hat{b_1} + |\mu_2| \|a\|_2 \|b\|_2
\end{aligned}
\tag{ii}
$$

where the (1) inequality is simply the Cauchy Schwartz inequality, and the (2) equality is due to the fact that replacing one orthonormal basis in another doesn't change the norm of the vector.

Simple calculations show that

$$
\begin{aligned}
\hat{a_1} = a^T\left(\frac{1}{\sqrt{N}}\bar{1}\right) = \frac{1}{\sqrt{N}}\sum_x a_x = \frac{1}{\sqrt{N}}\sum_x f(x) = \sqrt{N}\cdot E[f] \\
\hat{b_1} = b^T\left(\frac{1}{\sqrt{N}}\bar{1}\right) = \frac{1}{\sqrt{N}}\sum_x b_x = \sqrt{N}\cdot\rho(B^+) \\
\|b\|_2^2 = \sum_x b_x^2 = \sum_{x\in B^+} 1^2 = |B^+| \\
\|a\|_2^2 = \sum_x a_x^2 = \sum_x f(x)^2 \overset{(3)}{\le} \sum_x f(x) = N\cdot E[f]
\end{aligned}
$$

where (3) holds because $0 \le f(x) \le 1$. This inequality is, in fact, the only place where this analysis differs from one in [Gol97].

By "plugging in" the values for $\hat{a_1}, \hat{b_1}, \|a\|, \|b\|$ in (ii) we get:

$$
b^T M a \le N\rho(B^+)\cdot E[f] + \frac{\lambda}{D}\cdot\sqrt{N|B^+|\cdot E[f]}
$$

which implies

$$
\left|\frac{b^T M a}{N} - \rho(B^+)E[f]\right| \le \frac{\lambda}{D}\cdot\sqrt{\frac{B^+}{N}\cdot E[f]} \tag{iii}
$$

Combining both bounds in (i) and in (iii) we deduce:

$$
\epsilon\cdot\rho(B^+) \le \left|\frac{b^T M a}{N} - \rho(B^+)E[f]\right| \le \frac{\lambda}{D}\cdot\sqrt{E[f]\rho(B^+)} \Rightarrow \rho(B^+) \le \left(\frac{\lambda}{D\epsilon}\right)^2 E[f]
$$

Since we set $\lambda/D \leq \epsilon\sqrt{\delta}$ we have that $\rho(B^+) \leq \delta \cdot E[f]$.

Now, in order to bound $|B^-|$ one can apply the method used to bound the size of B^+. Alternatively, one may use the following reduction. We look at the complementary function of f, which is the function defined as $\overline{f} = 1 - f$. Let $\overline{B^+}$ be the set of $v \in [N]$ such that $Est_v(\overline{f}) > E[\overline{f}] + \epsilon$. The calculation from above holds for $\overline{f}$ as well, thus $\rho(\overline{B^+}) \leq \delta \cdot E[\overline{f}]$. But note that $\overline{B^+} = B^-$, meaning that every vertex producing an estimation which is at least ϵ smaller than $E[f]$, produces also an estimation which is at least ϵ bigger than $E[\overline{f}]$. We deduce that $\rho(B^-) \leq \delta \cdot E[\overline{f}] = \delta \cdot (1 - E[f])$.

Thus $\rho(B) = \rho(B^+) + \rho(B^-) \leq \delta \cdot E[f] + \delta \cdot (1 - E[f]) = \delta$. This completes the proof. $\square$

A.4 A Sampler Logarithmic in Confidence

We now show a sampler that is logarithmic in confidence. This sampler is based on the sampler that is linear in confidence, built in the previous section.

Bellare et al. [BGG93] introduced a general scheme, to construct a sampler logarithmic in confidence, given any sampler and an expander of suitable size. Originally, they implemented it using a pair-wise independent sampler and an expander random walk. This yields a construction of a sampler logarithmic in confidence, that has randomness complexity of $2n + O(\log(1/\delta))$. Here, we apply the same scheme, using the sampler from the previous section and an expander of suitable size, to construct a sampler also logarithmic in confidence, which has randomness complexity of $n + O(\log(1/\delta))$.

For any $\epsilon, \delta > 0$, let $S(\epsilon, \delta)$ be the (ϵ, δ)-sampler from Theorem A.3. Let R be the set of all possible coin tosses for $S(\epsilon, \delta)$.[14] For every $x \in R$ denote the estimation that S outputs when fed x as a random string, as $Est_x(f)$.

Algorithm 18 Sampling Algorithm 2

1: Let S be the sampling algorithm in Algorithm 17 with accuracy ϵ and confidence 0.99. Denote $R = \{0,1\}^n$ as the set of all possible random strings that S uses as a random input.
2: Let G be a (R, D, λ)-expander, where $|R| = N$, $D = O(1)$, and $\frac{\lambda}{D} \leq 0.01$.
3: Denote $l = 4\log_2(1/\delta)$.
4: Select randomly and uniformly a sequence $v, i_1, i_2, \ldots, i_l$ in $[N] \times ([D])^l$.
5: Denote by $v_1, v_2, \ldots, v_{l+1}$ a walk of length l over G, where $v_1 = v$ and for every j we have that $v_{j+1} = \Gamma_{i_j}(v_j)$.
6: For every $i = 1, 2, \ldots, l+1$, simulate S with v_i as its random string.
7: Let Est be the median of the set $\{E_1, E_2, \ldots, E_{l+1}\}$, where for every i we denote $E_i = Est_{v_i}(f)$. Output Est.

We wish to show that indeed Algorithm 18 is a sampling algorithm. In order to show that, we first state, without proof, the expander random walk theorem:

[14]Since R is always $[N]$, no matter what ϵ, δ are, there is no need to define R to be dependent of ϵ, δ

Theorem A.4 (Expander Random Walk Theorem, see [Gol97]). *Let G be a (N, D, λ)-expander. Let $W_0, W_1, \dots, W_l$ be subsets of V, and denote for every $i = 0, 1, \dots, l$ their density as $\rho_i = \frac{|W_i|}{|V|}$. Then the fraction of random walks in G of length l which intersect $W_0 \times W_1 \times \dots \times W_l$ is at most*

$$\sqrt{\rho_0 \rho_l} \prod_{i=1}^{l} \sqrt{\rho_i + (1-\rho_i)\left(\frac{\lambda}{D}\right)}$$

Claim A.5. *For any $N, \epsilon, \delta > 0$, Algorithm 18 is a (ϵ, δ)-sampling algorithm with query complexity $Q = O\left(\frac{1}{\epsilon^2}\log(1/\delta)\right)$, and randomness complexity $r = n + O(\log(1/\delta))$.*

Proof. We begin by showing that Algorithm 18 is indeed a (ϵ, δ)-sampler, meaning that $Pr[|Est - E[f]| > \epsilon] < \delta$, where N, ϵ, δ and $f : [N] \to [0, 1]$ are fixed arbitrarily.

Since Algorithm 18 outputs the median of l estimations, then the median is ϵ-far from the actual average of f if at least $\lceil l/2 \rceil$ estimations are either all bigger than $E[f] + \epsilon$ or all smaller than $E[f] - \epsilon$. We bound the probability that Algorithm 18 performs at least $\lceil l/2 \rceil$ simulations of S that give estimations which are ϵ-far from $E[f]$.

Let $B = \{x \in R : \ |Est_x(f) - E[f]| \geq \epsilon\}$. We know that $|B|/R \leq \delta$. By the expander random walk theorem stated above and the union bound, the probability that at least $l/2$ vertices in the random walk that Algorithm 18 performs reside in B is upper-bounded by

$$\sum_{j=l/2}^{l} \binom{l}{j} \left(\frac{1}{100} + \frac{1}{100}\right)^{j/2} \leq 2^l \cdot (0.02)^{l/4} = (0.32)^{l/4} < \delta$$

since we set $l = 4\log_2(1/\delta)$.

This means that w.p. greater than $1-\delta$ taking the median of all estimations will produce an estimation which is ϵ-close to $E[f]$. This guarantees that T is indeed a (ϵ, δ)-sampler.

Obviously, Algorithm 18 uses $n + l \cdot \log_2(D)$ coins, and therefore its randomness complexity is $n + O(\log(1/\delta))$. Since S has query complexity of $O\left(\frac{1}{\epsilon^2}\right)$, then Algorithm 18 which simulates S at most $l+1$ different times, has query complexity of $(l+1) \cdot O\left(\frac{1}{\epsilon^2}\right) = O\left(\frac{1}{\epsilon^2}\log(1/\delta)\right)$. We are done. ☐

Comment A.6. In fact, the scheme described in the proof of Claim A.5, gives a general scheme that applies to the error amplification of *any* randomized algorithm, and in particular, for property testers. Let T be a property tester with query complexity Q, randomness complexity r and error probability $1/3$. Suppose we wish to reduce the error probability of T to be at most δ. Then usually, we perform l independent simulations of T, and then take a majority vote between the $l = O(\log(1/\delta))$ simulations. By the Chernoff inequality, the majority vote is wrong with probability at most δ. In each of the l simulations, we do Q queries and use r random bits, so this standard amplification technique has query complexity $l \cdot Q$

and randomness complexity $l \cdot r$. Note that when δ is a constant, then these complexities are asymptotically the same as Q and r, respectively. However, we can reduce the error probability of T, just as we have reduced the error probability of Algorithm 17 from 0.01 to δ. We use a random walk of length $l = O(\log(1/\delta))$ over some fixed (N, D, λ)-expander, whose vertices represent all possible random coin tosses of T. A similar proof to the one of Claim A.5, shows that simulating T with the vertices traversed by this walk, and taking a majority vote, gives a tester with error probability at most δ. This amplification has query complexity of $l \cdot Q$, just as in the standard amplification, but randomness complexity of $r + l \cdot \log(D) = r + O(\log(1/\delta))$. The fact that the number of queries increases by a factor of $l = O(\log(1/\delta))$ seems unavoidable, as all known techniques for amplification perform at least $\log(1/\delta)$ simulations of T.

A.5 Hitters

We end this part with a weaker notion of samplers, called hitters. The standard notion of hitters regards algorithms that produce a set of query points, out of which at least one point is likely to fall inside a set of sufficiently large size. Here we introduce a generalization of this notion.

Definition A.7. *For any $n, \epsilon, \delta > 0$ and $c \geq 1$, a c-*hitter *S for sets of density ϵ and confidence $1 - \delta$, is a probabilistic oracle machine with the following property. For every set $T \subset N = \{0,1\}^n$ with density $\rho(T) > \epsilon$, the TM S uses at most r random bits, and queries at most Q points in N, which are denoted as $x_1, x_2, \ldots, x_Q$, that satisfy*

$$Pr\left[\, |\{x_1, x_2, \ldots, x_Q\} \cap T| < c\right] < \delta$$

This means that for every set T with density at least ϵ, we are likely to hit at least c points that belong to T using a c-hitter. We usually think of c as a small constant. We mostly refer to $c = 1$, in which case a 1-hitter is simply called a *hitter*.

One may ask why we bother and introduce the notion of c-hitters, instead of discussing 1-hitters, or merely hitters. True, the typical case is when one only needs one sample that falls in some set. Even in this work, only once, for testing whether a given graph is Eulerian (Section 3.2.5), we require the use of c-hitter instead of a hitter, and even in this case, we deal with $c = 3$. Nevertheless, as we argue next, it is worth to consider cases in which we need to produce $c > 1$ different samples.

Trivially, when c different samples are needed, one can use the regular amplification, and repeat c independent times the procedure of hitting one sample. Suppose we have a hitter with query complexity Q and randomness complexity r. Then using this hitter c independent times gives query complexity of $c \cdot Q$, randomness complexity of $c \cdot r$, and Q-R complexity of $c^2 r \cdot Q$. Granted, for this work, $c = O(1)$, and so this multiplication does not "cost" us, asymptotically. But when c is not a constant, this amplification is very wasteful.

However, as we show, c-hitters have query complexity and randomness complexity that are asymptotically identical to the complexities of 1-hitters. That means, that we can put the same amount of effort as the amount of effort a 1-hitter required, and find c samples, and not just one.

This is best demonstrated by an example. The *Pairwise Independent Hitter* is an algorithm that produces Q entries in $[N]$ in a pairwise independent manner. Given any set $T \subset [N]$, where $\rho(T) > \epsilon$, and a sample of $Q = \max\{\frac{4c}{\epsilon}, \frac{2}{\epsilon\delta}\}$ entries in $[N]$, denote by X_i the random variable indicating whether the i sample point belongs to T or not. Since the Q query points are taken in a pairwise independent manner, then we apply the Chebyshev inequality, and deduce:

$$\begin{aligned} Pr\left[\sum_{i=1}^{Q} X_i < c\right] &\leq Pr\left[\left|\rho(T) \cdot Q - \sum_{i=1}^{Q} X_i\right| \geq (\rho(T) \cdot Q - c)\right] \\ &\leq \frac{\rho(T)(1-\rho(T)) \cdot Q}{\rho(T)^2 Q^2 - 2Q \cdot \rho(T) \cdot c + c^2} \overset{(1)}{\leq} \frac{\rho(T) \cdot Q}{\frac{1}{2}\rho(T)^2 Q^2} \leq \frac{2}{\epsilon Q} \overset{(2)}{\leq} \delta \end{aligned}$$

where (1) is due to the fact that $Q \geq \frac{4c}{\epsilon} \geq \frac{4c}{\rho(T)}$ and (2) is due to the fact that $Q \geq \frac{2}{\epsilon\delta}$.

Note that for any $c < \frac{1}{2\delta}$, the Pairwise Independent c-hitter has query complexity of $Q = \frac{2}{\epsilon\delta}$, and there is a known construction for a Pairwise independent hitter with randomness complexity of $2n$. These complexities do not depend on c, and they are the same for every possible value that c takes, between 1 and $\frac{1}{2\delta}$. If instead of using a c-hitter, we were to use the regular amplification method for a 1-hitter, we would have query complexity of $c \cdot \frac{2}{\epsilon\delta}$ and randomness complexity of $2c \cdot n$.

The Pairwise Independent Hitter also demonstrates another point. As a thumb rule, samplers have their analogous hitters, where the construction of the hitter is very similar to the construction of the sampler. Usually, the difference between the hitter and the sampler lies in their query complexity. While the query complexity of samplers is usually proportional to $1/\epsilon^2$, the query complexity of hitters is usually proportional to $1/\epsilon$. The Pairwise Independent hitter demonstrate this point, because it is the analogous of the Pairwise Independent sampler, that queries a function $f : N \to [0, 1]$ over a set of sample points from $[N]$ taken in a pairwise independent manner. The Pairwise Independent Hitter has query complexity of $O(\frac{1}{\epsilon^2\delta})$, while, as we have shown, the query complexity of the Pairwise Independent Hitter is $O(\frac{1}{\epsilon\delta})$.

We also classify hitters in an analogous way to our classification of samplers:

Definition A.8. *We call a c-hitter* polynomial in confidence *if its query complexity Q is proportional to $p(1/\delta)$ for some polynomial p. We say it is* linear in confidence *if Q is* *proportional to* $1/\delta$. *We call a c-hitter* logarithmic in confidence *if* Q *is proportional to* $\log(1/\delta)$.

We show here the two hitters that are analogous to two samplers of Section A.3 and Section A.4.

A.5.1 A Hitter Linear in Confidence

The analog of the Expander Neighbors Sampler is the *Expander-Neighbors Hitter* (see [Gol97]).

Fix any $\epsilon, \delta > 0$. Let G be a (N, D, λ)-expander, where $D = O\left(\frac{1}{\epsilon\delta} + \frac{c}{\epsilon}\right)$ and $\frac{\lambda}{D} \le \sqrt{\frac{\epsilon\delta}{2}}$. For any $v \in [N]$, denote the ith neighbor of v as $\Gamma_i(v)$.

Algorithm 19 Hitting Algorithm 1

1: Select randomly and uniformly a vertex $v \in_R [N] = [2^n] = \{0,1\}^n$.
2: Output the following D entries: $f(\Gamma_1(v)), f(\Gamma_2(v)), \ldots, f(\Gamma_D(v))$.

Claim A.9. *For every* $n, \epsilon, \delta > 0$ *and* $c \ge 1$*, Algorithm 19 is a c-hitter with for sets of density at least* ϵ *and confidence* $1 - \delta$*. Its randomness complexity is* n *and its query complexity is* $O\left(\frac{1}{\epsilon\delta} + \frac{c}{\epsilon}\right)$.

As mentioned before, we assume that $c = O(1)$, therefore, we have that the query complexity of Algorithm 19 is $O(\frac{1}{\epsilon\delta})$.

In order to proof the claim, we use the Expander Mixing Lemma, which we state here without proof:[15]

Theorem A.10 (Expander Mixing Lemma). *Let* G *be a* (N, D, λ)*-expander. For any two* A, B *subsets of* $V(G)$*, we denote by* $E(A, B)$ *the number of edges with one endpoint in* A *and the other in* B*. Then the following holds that for any* A *and* B*:*

$$\left| \frac{|E(A,B)|}{N \cdot D} - \rho(A)\rho(B) \right| < \frac{\lambda}{D}\sqrt{\rho(A)\rho(B)}$$

Proof. The hitter selects uniformly a vertex in G, by tossing n coins, and queries all of its neighbors. Therefore, it is clear that its randomness complexity is $r = n$, and its query complexity is $Q = D = O\left(\frac{1}{\epsilon\delta} + \frac{c}{\epsilon}\right)$. Let us now show that with probability at least $1 - \delta$ it hits at least c times any set with density at least ϵ.

Fix any $T \subset [N]$ such that $\rho(T) \ge \epsilon$, and let $B \subset [N]$ be the set of all vertices $\{x \in [N] : |\Gamma(x) \cap T| < c\}$. Therefore, Algorithm 19 fails to produce a set of queries that hit T at least c times, if the vertex that Algorithm 19 select belongs to B. So let us bound the probability $Pr_{U_n}[v \in B] = \rho(B)$.

[15] In fact, its proof is implicit in the proof of Theorem A.3.

By the expander mixing lemma we have that

$$\left| \frac{|E(B,T)|}{N \cdot D} - \rho(B)\rho(T) \right| \leq \frac{\lambda}{D} \sqrt{\rho(B)\rho(T)}$$

The definition of B gives that $|E(B,T)| < c|B|$, and recall that $\frac{\lambda}{D} \leq \sqrt{\epsilon\delta/2}$. "Plugging in" these values, we have

$$\rho(B) \cdot (\rho(T) - \frac{c}{D}) \leq \sqrt{\frac{\epsilon\delta}{2} \cdot \rho(B)\rho(T)} \Rightarrow \rho(B) \leq \delta \left(\frac{\epsilon \cdot \rho(T)}{2(\rho(T) - \frac{c}{D})^2} \right) \leq \delta \left(\frac{\epsilon}{2(\rho(T) - \frac{c}{D})} \right)$$

Recall that $\rho(T) \geq \epsilon$ and that $D > \frac{2c}{\epsilon}$. This gives that $\rho(T) - \frac{c}{D} > \frac{\epsilon}{2}$, and so we have that $\rho(B) < \delta$.

Thus Algorithm 19 is indeed a c-hitter, since with probability at least $1-\delta$ it hits at least c times any set of density ϵ or more. $\square$

A.5.2 A Hitter Logarithmic in Confidence

Analogously to the construction of the sampler from Section A.4, we use the hitter linear in confidence, to construct a hitter logarithmic in confidence. To that end we also use random walks over expanders. Only this time, we use a weaker theorem regarding expander random walks:

Theorem A.11 (Expander Random Walk, see [Gol97]). *Given G, a (N, D, λ)-expander and a set $W \subset N$, the fraction of random walks of length l over G, that stay within W, is at most*

$$\rho(W) \left(\rho(W) + (1 - \rho(W)) \frac{\lambda}{D} \right)^l$$

Let us formally define the hitting algorithm. For any $\epsilon, \delta > 0$, let $S(\epsilon, \delta)$ be the c-hitter from Claim A.9 for sets of density at least ϵ and confidence at least $1-\delta$. Let R be the set of all possible coin tosses for $S(\epsilon, \delta)$. For every $v \in R$ denote the set of queries that S outputs when fed v as a random string, as P_v.

Algorithm 20 Hitting Algorithm 2

1: Let S be Algorithm 19 with accuracy ϵ and confidence 0.75. Denote $R = \{0,1\}^n$ as the set of all possible random strings that S uses as a random input.
2: Let G be a (R, D, λ)-expander, where $|R| = N$, $D = O(1)$, and $\frac{\lambda}{D} \leq 0.25$.
3: Denote $l = \log_2(1/\delta)$.
4: Select randomly and uniformly a sequence $v, i_1, i_2, \ldots, i_l$ in $[N] \times ([D])^l$.
5: Denote by $v_1, v_2, \ldots, v_{l+1}$ a walk of length l over G, where $v_1 = v$ and for every j we have that $v_{j+1} = \Gamma_{i_j}(v_j)$.
6: Output $\bigcup_{i=1}^{l+1} P_{v_i}$.

Claim A.12. *For any n, ϵ, δ and $c \geq 1$, Algorithm 20 is a c-hitter for sets of density at least ϵ and with confidence $1-\delta$. It has query complexity of $O\left(\frac{c}{\epsilon}\log(1/\delta)\right)$, and randomness complexity of $n + O(\log(1/\delta))$.*

Proof. Fix n, ϵ, δ and c. Since S has query complexity of $O\left(\frac{c}{\epsilon}\right)$, it is clear that Algorithm 20 has query complexity of $l \cdot |P_v| = O\left(\frac{c}{\epsilon}\log(1/\delta)\right)$. The randomness complexity of Algorithm 20 is $\log(|R|) + l \cdot \log_2(D) = n + O(\log(1/\delta))$. So we have to show that the probability that Algorithm 20 does not produce at least c queries into elements of a set of density at least ϵ, is upper bounded by δ.

Let T be any subset of $[N]$ of density at least ϵ. Denote, as before, the set $B = \{v \in R : |P_v \cap T| < c\}$. Since S is a c-hitter with confidence at least 0.75, we have that $\rho(B) < 0.25$.

The Expander Random Walk theorem assures us, the the probability that the random walk $v_1, v_2, \ldots, v_{l+1}$ stays within B is at most

$$\left(\rho(B) + \frac{\lambda}{D}\right)^l = \left(\frac{1}{4} + \frac{1}{4}\right)^l = 2^{-l} \leq \delta$$

Therefore, the probability that Algorithm 20 produces a set that does not contain c elements from T is at most δ. This proves the claim. ∎

Comment A.13. Just as in Comment A.6, we wish to stress that Claim A.12 describes a general scheme to reduce the error of *any* randomized one-sided algorithm. Specifically, let T is a one-sided property tester, with query complexity Q, randomness complexity r and error probability 1/3. We can perform a random-walk of length $l = O(\log(1/\delta))$ over a (N, D, λ)-expander, whose vertices represent the possible coin tosses of T, and simulate T with the vertices traversed by this walk. The probability that T accepts some input that is ϵ-far from the property, can be upper bounded by δ, just as in the proof of Claim A.12. This amplification has query complexity of $l \cdot Q$ and randomness complexity of $r + l \cdot \log(D) = r + O(\log(1/\delta))$. Again, we stress that all known amplification techniques perform at least $\log(1/\delta)$ simulations of T, and thus increasing the original query complexity of T by a factor of $\log(1/\delta)$.