Lower Bounds for Tolerant Junta and Unateness Testing via Rejection Sampling of Graphs

Amit Levi[1]
University of Waterloo, Canada
amit.levi@uwaterloo.ca

Erik Waingarten[2]
Columbia University, USA
eaw@cs.columbia.edu

Abstract

We introduce a new model for testing graph properties which we call the *rejection sampling model*. We show that testing bipartiteness of n-nodes graphs using rejection sampling queries requires complexity $\widetilde{\Omega}(n^2)$. Via reductions from the rejection sampling model, we give three new lower bounds for tolerant testing of Boolean functions of the form $f\colon \{0,1\}^n \to \{0,1\}$:

- Tolerant k-junta testing with *non-adaptive* queries requires $\widetilde{\Omega}(k^2)$ queries.
- Tolerant unateness testing requires $\widetilde{\Omega}(n)$ queries.
- Tolerant unateness testing with *non-adaptive* queries requires $\widetilde{\Omega}(n^{3/2})$ queries.

Given the $\widetilde{O}(k^{3/2})$-query non-adaptive junta tester of Blais [7], we conclude that non-adaptive tolerant junta testing requires more queries than non-tolerant junta testing. In addition, given the $\widetilde{O}(n^{3/4})$-query unateness tester of Chen, Waingarten, and Xie [19] and the $\widetilde{O}(n)$-query non-adaptive unateness tester of Baleshzar, Chakrabarty, Pallavoor, Raskhodnikova, and Seshadhri [3], we conclude that tolerant unateness testing requires more queries than non-tolerant unateness testing, in both adaptive and non-adaptive settings. These lower bounds provide the first separation between tolerant and non-tolerant testing for a natural property of Boolean functions.

2012 ACM Subject Classification Theory of computation → Probabilistic computation

Keywords and phrases Property Testing, Juntas, Tolerant Testing, Boolean functions

Digital Object Identifier 10.4230/LIPIcs.ITCS.2019.52

Related Version A full version of the paper is available at `https://arxiv.org/abs/1805.01074v1`.

1 Introduction

Over the past decades, property testing has emerged as an important line of research in sublinear time algorithms. The goal is to understand randomized algorithms for approximate decision making, where the algorithm needs to decide (with high probability) whether a huge object has some property by making a few queries to the object. Many different types of objects and properties have been studied from this property testing perspective (see the surveys by Ron [35, 36] and the recent textbook by Goldreich [26] for overviews of contemporary property testing research). This paper deals with property testing of Boolean functions $f\colon \{0,1\}^n \to \{0,1\}$ and property testing of graphs with vertex set $[n]$.

[1] Research supported by NSERC Discovery grant and the David R. Cheriton Graduate Scholarship. Part of this work was done while the author was visiting Columbia University.
[2] This work is supported in part by the NSF Graduate Research Fellowship under Grant No. DGE-16-44869, CCF-1703925, CCF-1563155, and CCF-1420349.

In this paper we describe a new model of graph property testing, which we call the *rejection sampling model*. For $n \in \mathbb{N}$ and a subset $\mathcal{P}$ of graphs on the vertex set $[n]$, we say a graph G on vertex set $[n]$ has property $\mathcal{P}$ if $G \in \mathcal{P}$ and say G is ε-far from having property $\mathcal{P}$ if all graphs $H \in \mathcal{P}$ differ on at least εn^2 edges[3]. The problem of ε-testing $\mathcal{P}$ with *rejection sampling queries* is the following task:

> Given some $\varepsilon > 0$ and access to an unknown graph $G = ([n], E)$, output "accept" with probability at least $\frac{2}{3}$ if G has property $\mathcal{P}$, and output "reject" with probability at least $\frac{2}{3}$ if G is ε-far from having property $\mathcal{P}$. The access to G is given by the following oracle queries: given a query set $L \subseteq [n]$, the oracle samples an edge $(\boldsymbol{i}, \boldsymbol{j}) \sim E$ uniformly at random and returns $\{\boldsymbol{i}, \boldsymbol{j}\} \cap L$.

We measure the complexity of algorithms with rejection sampling queries by considering the sizes of the queries. The complexity of an algorithm making queries $L_1, \ldots, L_t \subseteq [n]$ is $\sum_{i=1}^{t} |L_i|$.

The rejection sampling model allows us to study testers which rely on random sampling of edges, while providing the flexibility of making lower-cost queries. This type of query access strikes a delicate balance between simplicity and generality: queries are constrained enough for us to show high lower bounds, and at the same time, the flexibility of making queries allows us to reduce the rejection sampling model to Boolean function testing problems. Specifically, we reduce to tolerant junta testing and tolerant unateness testing (see Subsection 1.1).

Our main result in the rejection sampling model is regarding *non-adaptive* algorithms. These algorithms need to fix their queries in advance and are not allowed to depend on answers to previous queries (in the latter case we say that the algorithm is *adaptive*). We show a lower bound on the complexity of testing whether an unknown graph G is bipartite using non-adaptive queries.

▶ **Theorem 1.** *There exists a constant $\varepsilon > 0$ such that any non-adaptive ε-tester for bipartiteness in the rejection sampling model has cost $\widetilde{\Omega}(n^2)$.*[4]

More specifically, Theorem 1 follows from applying Yao's principle to the following lemma.

▶ **Lemma 2.** *Let $\mathcal{G}_1$ be the uniform distribution over the union of two disjoint cliques of size $n/2$, and let $\mathcal{G}_2$ be the uniform distribution over complete bipartite graphs with each part of size $n/2$. Any deterministic non-adaptive algorithm that can distinguish between $\mathcal{G}_1$ and $\mathcal{G}_2$ with constant probability using rejection sampling queries, must have complexity $\widetilde{\Omega}(n^2)$.*

We discuss a number of applications of the rejection sampling model (specifically, of Lemma 2) in the next subsection. In particular, we obtain new lower bounds in the *tolerant testing framework* introduced by Parnas, Ron, and Rubinfeld in [34] for two well-studied properties of Boolean functions (specifically, k-juntas and unateness; see the next subsection for definitions of these properties). These lower bounds are obtained by a reduction from the rejection sampling model; we show that too-good-to-be-true Boolean function testers for these properties imply the existence of rejection sampling algorithms which distinguish $\mathcal{G}_1$ and $\mathcal{G}_2$ with $\tilde{o}(n^2)$ complexity. Therefore, we may view the rejection sampling model as a useful abstraction in studying the hard instances of tolerant testing k-juntas and unateness.

---

[3] The distance definition can be modified accordingly when one considers bounded degree or sparse graphs.

[4] We use the notations $\widetilde{O}, \widetilde{\Omega}$ to hide polylogarithmic dependencies on the argument, i.e. for expressions of the form $O(f \log^c f)$ and $\Omega(f/\log^c f)$ respectively (for some absolute constant c).

1.1 Applications to Tolerant Testing: Juntas and Unateness

Given $n \in \mathbb{N}$ and a subset $\mathcal{P}$ of n-variable Boolean functions, a Boolean function $f\colon \{0,1\}^n \to \{0,1\}$ has property $\mathcal{P}$ if $f \in \mathcal{P}$. The distance between Boolean functions $f, g\colon \{0,1\}^n \to \{0,1\}$ is $\text{dist}(f,g) = \mathbf{Pr}_{\boldsymbol{x}\sim\{0,1\}^n}[f(\boldsymbol{x}) \neq g(\boldsymbol{x})]$. The distance of f to the property $\mathcal{P}$ is $\text{dist}(f,\mathcal{P}) = \min_{g\in\mathcal{P}} \text{dist}(f,g)$. We say that f is ε-close to $\mathcal{P}$ if $\text{dist}(f,\mathcal{P}) \leq \varepsilon$ and f is ε-far from $\mathcal{P}$ if $\text{dist}(f,\mathcal{P}) > \varepsilon$. The problem of *tolerant property testing* [34] of $\mathcal{P}$ asks for query-efficient randomized algorithms for the following task:

> Given parameters $0 \leq \varepsilon_0 < \varepsilon_1 < 1$ and black-box query access to a Boolean function $f\colon \{0,1\}^n \to \{0,1\}$, accept with probability at least $\frac{2}{3}$ if f is ε_0-close to $\mathcal{P}$ and reject with probability at least $\frac{2}{3}$ if f is ε_1-far from $\mathcal{P}$.

An algorithm which performs the above task is an $(\varepsilon_0,\varepsilon_1)$-tolerant tester for $\mathcal{P}$. A $(0,\varepsilon_1)$-tolerant tester is a *standard* property tester or a *non-tolerant* tester. As noted in [34], tolerant testing is not only a natural generalization, but is also very often the desirable attribute of testing algorithms. This motivates the high level question: how does the requirement of being tolerant affect the complexity of testing the properties studied? We make progress on this question by showing query-complexity separations for two well-studied properties of Boolean functions: k-juntas, and unate functions.

- (k-junta) A function $f\colon \{0,1\}^n \to \{0,1\}$ is a *k-junta* if it depends on at most k of its variables, i.e., there exists k distinct indices $i_1, \dots i_k \in [n]$ and a k-variable function $g\colon \{0,1\}^k \to \{0,1\}$ where $f(x) = g(x_{i_1}, \dots, x_{i_k})$ for all $x \in \{0,1\}^n$.
- (unateness) A function $f\colon \{0,1\}^n \to \{0,1\}$ is *unate* if f is either non-increasing or non-decreasing in every variable. Namely, there exists a string $r \in \{0,1\}^n$ such that the function $f(x \oplus r)$ is monotone with respect to the bit-wise partial order on $\{0,1\}^n$.

While separations between tolerant and non-tolerant testing of Boolean function properties were known for an (artificial) property (see Subsection 1.2), these results are the first to give such lower bounds for a natural class of well-studied properties of Boolean functions. The first such theorem we state concerns non-adaptive tolerant testers for k-juntas.

▶ **Theorem 3.** *For any $\alpha < 1$, there exists constants $0 < \varepsilon_0 < \varepsilon_1 < 1$ such that for any $k = k(n) \leq \alpha n$, any non-adaptive $(\varepsilon_0,\varepsilon_1)$-tolerant k-junta tester must make $\widetilde{\Omega}(k^2)$ queries.*

We give a noteworthy consequences of the Theorem 3. In [7], Blais gave a non-adaptive $\widetilde{O}(k^{3/2})$-query tester for (non-tolerant) testing of k-juntas, which was shown to be optimal for non-adaptive algorithms by Chen, Servedio, Tan, Waingarten and Xie in [17]. Combined with Theorem 3, this shows a polynomial separation in the query complexity of non-adaptive tolerant junta testing and non-adaptive junta testing.

The next two theorems concern tolerant testers for unateness.

▶ **Theorem 4.** *There exists constants $0 < \varepsilon_0 < \varepsilon_1 < 1$ such that any (possibly adaptive) $(\varepsilon_0,\varepsilon_1)$-tolerant unateness tester must make $\widetilde{\Omega}(n)$ queries.*

▶ **Theorem 5.** *There exists constant $0 < \varepsilon_0 < \varepsilon_1 < 1$ such that any non-adaptive $(\varepsilon_0,\varepsilon_1)$-tolerant unateness tester must make $\widetilde{\Omega}(n^{3/2})$ queries.*

A similar separation in tolerant and non-tolerant testing occurs for the property of unateness as a consequence of Theorem 4 and Theorem 5. Recently, in [3], Baleshzar, Chakrabarty, Pallavoor, Raskhodnikova, and Seshadhri gave a non-adaptive $\widetilde{O}(n)$-query tester for (non-tolerant) unateness testing, and Chen, Waingarten and Xie [18] gave an

(adaptive) $\widetilde{O}(n^{3/4})$-query tester for (non-tolerant) unateness testing. We thus, conclude that by Theorem 4 and Theorem 5, tolerant unateness testing is polynomially harder than (non-tolerant) unateness testing, in both adaptive and non-adaptive settings.

1.2 Related Work

The properties of k-juntas and unateness have received much attention in property testing research ([24, 20, 7, 8, 10, 37, 17, 9] study k-juntas, and [27, 31, 14, 3, 18, 19] study unateness). We briefly review the current state of affairs in (non-tolerant) k-junta testing and unateness testing, and then discuss tolerant testing of Boolean functions and the rejection sampling model.

Testing k-juntas. The problem of testing k-juntas, introduced by Fischer, Kindler, Ron, Safra, and Samorodnitsky [24], is now well understood up to poly-logarithmic factors. Chockler and Gutfreund [20] show that any tester for k-juntas requires $\Omega(k)$ queries (for a constant ε_1). Blais [8] gave a junta tester that uses $O(k \log k + k/\varepsilon_1)$ queries, matching the bound of [20] up to a factor of $O(\log k)$ for constant ε_1. When restricted to non-adaptive algorithms, [24] gave a non-adaptive tester making $\widetilde{O}(k^2/\varepsilon_1)$ queries, which was subsequently improved in [7] to $\widetilde{O}(k^{3/2})/\varepsilon_1$. In terms of lower bounds, Buhrman, Garcia-Soriano, Matsliah, and de Wolf [10] gave a $\Omega(k \log k)$ lower bound for $\varepsilon = \Omega(1)$, and Servedio, Tan, and Wright [37] gave a lower bound which showed a separation between adaptive and non-adaptive algorithms for $\varepsilon_1 = \frac{1}{\log k}$. These results were recently improved in [17] to $\widetilde{\Omega}(k^{3/2}/\varepsilon_1)$, settling the non-adaptive query complexity of the problem up to poly-logarithmic factors.

Testing unateness. The problem of testing unateness was introduced alongside the problem of testing monotonicity in Goldreich, Goldwasser, Lehman, Ron, and Samorodnitsky [27], where they gave the first $O(n^{3/2}/\varepsilon_1)$-query non-adaptive tester. Khot and Shinkar [31] gave the first improvement by giving a $\widetilde{O}(n/\varepsilon_1)$-query adaptive algorithm. A non-adaptive algorithm with $\widetilde{O}(n/\varepsilon_1)$ queries was given in [13, 3]. Recently, [18, 2] show that $\widetilde{\Omega}(n)$ queries are necessary for non-adaptive one-sided testers. Subsequently, [19] gave an adaptive algorithm testing unateness with query complexity $\widetilde{O}(n^{3/4}/\varepsilon_1^2)$. The current best lower bound for general adaptive testers appears in [18], where it was shown that any adaptive two-sided tester must use $\widetilde{\Omega}(n^{2/3})$ queries.

Tolerant testing. Once we consider tolerant testing, i.e., the case $\varepsilon_0 > 0$, the picture is not as clear. In the paper introducing tolerant testing, [34] observed that standard algorithms whose queries are uniform (but not necessarily independent) are inherently tolerant to some extent. Nevertheless, achieving $(\varepsilon_0, \varepsilon_1)$-tolerant testers for constants $0 < \varepsilon_0 < \varepsilon_1$, can require applying different methods and techniques (see e.g, [30, 34, 25, 1, 32, 33, 22, 11, 6, 5, 38]).

By applying the observation from [34] to the unateness tester in [3], the tester accepts functions which are $O(\varepsilon_1/n)$-close to unate with constant probability. We similarly obtain weak guarantees for tolerant testing of k-juntas. Diakonikolas, Lee, Matulef, Onak, Rubinfeld, Servedio, and Wan [21] observed that one of the (non-adaptive) junta testers from [24] accepts functions that are $\text{poly}(\varepsilon_1, 1/k)$-close to k-juntas. Chakraborty, Fischer, Garcia-Soríano, and Matsliah [15] noted that the analysis of the junta tester of Blais [8] implicitly implies an $\exp(k/\varepsilon_1)$-query complexity tolerant tester which accepts functions that are ε_1/c-close to some k-junta (for some constant $c > 1$) and rejects functions that are ε_1-far from every k-junta. Recently, Blais, Canonne, Eden, Levi and Ron [9] showed that when required to distinguish between the cases that f is $\varepsilon_1/10$-close to a k-junta, or is ε_1-far from a $2k$-junta, $\text{poly}(k, 1/\varepsilon_1)$ queries suffice.

For general properties of Boolean functions, tolerant testing could be much harder than standard testing. Fischer and Fortnow [23] used PCPs in order to construct a property of Boolean functions $\mathcal{P}$ which is $(0, \varepsilon_1)$-testable with a constant number of queries (depending on ε_1), but any $(1/4, \varepsilon_1)$-tolerant test for $\mathcal{P}$ requires n^c queries for some $c > 0$. While [23] presents a strong separation between tolerant and non-tolerant testing, the complexity of tolerant testing of many natural properties remains open. We currently neither have a $\text{poly}(k, \frac{1}{\varepsilon_1})$-query tester which $(\varepsilon_0, \varepsilon_1)$-tests k-juntas, nor a $\text{poly}(n, \frac{1}{\varepsilon_1})$-query tester that $(\varepsilon_0, \varepsilon_1)$-tests unateness or monotonicity when $\varepsilon_0 = \Theta(\varepsilon_1)$.

Testing graphs with rejection sampling queries. Even though the problem of testing graphs with rejection sampling queries has not been previously studied, the model shares characteristics with previous studied frameworks. These include sample-based testing studied by Goldreich, Goldwasser, and Ron in [28, 29], where the oracle receives random samples from the input. One crucial difference between rejection sampling algorithms (which always query $[n]$) and sample-based testers is the fact that rejection sampling algorithms only receive *positive* examples (in the form of edges), as opposed to random positions in the adjacency matrix (which may be a *negative* example indicated the non-existence of an edge).

The rejection sampling model for graph testing also bears some resemblance to the conditional sampling framework for distribution testing introduced in Canonne, Ron, and Servedio, as well as Chakraborty, Fischer, Goldhirsh, and Matsliah [12, 16], where the algorithm specifies a query set and receives a sample conditioned on it lying in the query set.

1.3 Techniques and High Level Overview

We first give an overview of how the lower bound in the rejection sampling model (Lemma 2) implies lower bounds for tolerant testing of k-juntas and unateness, and then we give an overview of how Lemma 2 is proved.

Reducing Boolean Function Testing from Rejection Sampling. This work should be considered alongside some recent works showing lower bounds for testing the properties of monotonicity, unateness, and juntas in the standard property testing model [4, 18, 17]. At a high level, the lower bounds for Boolean function testing proceed in three steps:

1. First, design a randomized indexing function $\mathbf{\Gamma}\colon \{0,1\}^n \to [N]$ that partitions the Boolean cube $\{0,1\}^n$ into roughly equal parts in a way compatible with the property (either junta, or unateness). We want to ensure that algorithms that make few queries cannot learn too much about $\mathbf{\Gamma}$, and that queries falling in the same part are close in Hamming distance.
2. Second, define two distributions over functions $\boldsymbol{h}_i\colon \{0,1\}^n \to \{0,1\}$ for each $i \in [N]$. The hard functions are defined by $\boldsymbol{f}(x) = \boldsymbol{h}_{\mathbf{\Gamma}(x)}(x)$, so that one distribution corresponds to functions with the property, and the other distribution corresponds to functions far from the property.
3. Third, show that any testing algorithm for the property is actually solving some algorithmic task (determined by the distributions of $\boldsymbol{h}_i$) which is hard when queries are close in Hamming distance.

The first step in the above-mentioned plan is standard (given familiarity with [18] and [17]). We will use a construction from [17] for the junta lower bound and a Talagrand-based construction (similar to [18], but somewhat simpler) for the unateness lower bounds. The novelty in this work lies in steps 2 and 3. We will define the distributions over sub-functions $\boldsymbol{h}_i$ such that the resulting Boolean functions $\boldsymbol{f}(x) = \boldsymbol{h}_{\mathbf{\Gamma}(x)}(x)$ either is ε_0-close to desired

property (k-juntas and unateness), or is ε_1-far from having the desired property (k-juntas and unateness). Then, we will show that any algorithm for tolerant testing of k-juntas or unateness must be able to solve a hard instance of bipartiteness testing in the rejection sampling model.

At a very high level, our reductions will follow by associating to each distribution of Boolean functions $\boldsymbol{f}\colon \{0,1\}^n \to \{0,1\}$ a distribution over graphs $\mathbf{G}$ defined on a subset of $[n]$ (these will be $\mathcal{G}_1$ and $\mathcal{G}_2$). The edges of a graph $\mathbf{G}$ sampled from $\mathcal{G}_1$ or $\mathcal{G}_2$ will encode how the variables of $\boldsymbol{f}$ interact with one another, and the distance of $\boldsymbol{f}$ to k-junta (or unateness) will depend on a global parameter of the $\mathbf{G}$.[5] In addition, Boolean function queries on $\boldsymbol{f}$ will be interpreted as rejection sampling queries to $\mathbf{G}$, so that tests distinguishing the distributions of Boolean functions will give rise to rejection sampling algorithms which distinguish between $\mathcal{G}_1$ and $\mathcal{G}_2$. Since we will show a lower bound in the rejection sampling model, we will obtain a lower bound for tolerant testing of k-juntas and unateness.

For a more detailed discussion of the distributions and the reductions see Sections 3 and 4.

Distinguishing $\mathcal{G}_1$ and $\mathcal{G}_2$ with Rejection Sampling Queries. In order to prove Lemma 2, one needs to rule out any deterministic non-adaptive algorithm which distinguishes between $\mathcal{G}_1$ and $\mathcal{G}_2$ with rejection sampling queries of complexity $\widetilde{o}(n^2)$. In order to keep the discussion at a high level, we identify three possible "strategies" for determining whether an underlying graph is a complete bipartite graph, or a union of two disjoint cliques:

1. One approach is for the algorithm to sample edges and consider the subgraph obtained from edges returned by the oracle. For instance, the algorithm may make all rejection sampling queries to be $[n]$. These queries are expensive in the rejection sampling model, but they guarantee that an edge from the graph will be observed. If the algorithm is lucky, and there exists a triangle in the subgraph observed, the graph must not be bipartite, so it must come from $\mathcal{G}_2$.
2. Another sensible approach is for the algorithm to forget about the structure of the graph, and simply view the distribution on the edges generated by the randomness in the rejection sampling oracle as a distribution testing problem. Suppose for simplicity that the algorithm makes rejection sampling queries $[n]$. Then, the corresponding distributions supported on edges from $\mathcal{G}_1$ and $\mathcal{G}_2$ will be $\Omega(1)$-far from each other, so a distribution testing algorithm can be used.
3. A third, more subtle, approach is for the algorithm to use the fact that $\mathcal{G}_1$ and $\mathcal{G}_2$ correspond to the union of two cliques and a complete bipartite graph, and extract knowledge about the non-existence of edges when making queries which return either $\emptyset$ or a single vertex. More specifically, suppose that by having observed some edges, the algorithm observes two connected components L_1 and L_2. If when querying $L_1 \cup L_2$ multiple times, we do not observe an edge, it is more likely the underlying graph comes from $\mathcal{G}_1$ than $\mathcal{G}_2$. Specifically, if $\mathbf{G} \sim \mathcal{G}_1$ and L_1 lies in one clique and L_2 lies in the other clique, there would be no edges with edges from L_1 and L_2; on the other hand, if $\mathbf{G} \sim \mathcal{G}_2$, then L_1 and L_2 would always have some edges between them.

[5] The relevant graph parameter in k-juntas and unateness will be different. Luckily, both graph parameters will have gaps in their value depending on the distribution the graphs were drawn from (either $\mathcal{G}_1$ or $\mathcal{G}_2$). This allows us to reuse the work of proving Lemma 2 to obtain Theorem 3, Theorem 4, and Theorem 5.

The three strategies mentioned above all fail to give $\widetilde{o}(n^2)$ rejection sampling algorithms. The first approach fails because with a budget of $\widetilde{o}(n^2)$, rejection sampling algorithms will observe subgraphs which consist of various trees of size at most $\log n$, thus we will not observe cycles. The second approach fails since the distributions are supported on $\Omega(n^2)$ edges, so distribution testing algorithms will require $\Omega(n)$ edges (which costs $\Omega(n^2)$) to distinguish between $\mathcal{G}_1$ and $\mathcal{G}_2$. Finally, the third approach fails since algorithms will only observe $o(n)$ responses from the oracle corresponding to lone vertices which will be split roughly evenly among the unknown parts of the graph, so these observations will not be enough to distinguish between $\mathcal{G}_1$ and $\mathcal{G}_2$.

Our lower bound rules out the three strategies sketched above when the complexity is $\widetilde{o}(n^2)$, and shows that if the above three strategies do not work (in any possible combination with each other as well), then no non-adaptive algorithm of complexity $\widetilde{o}(n^2)$ will work. The main technical challenge is to show that the above strategies are the *only* possible strategies to distinguish $\mathcal{G}_1$ and $\mathcal{G}_2$. In Section 5, we give a more detailed, yet still high-level discussion of the proof of Lemma 2.

Finally, the analysis of Lemma 2 is tight; there is a non-adaptive rejection sampling algorithm which distinguishes $\mathcal{G}_1$ and $\mathcal{G}_2$ with complexity $\widetilde{O}(n^2)$. The algorithm (based on the first approach mentioned above) is simple: make $\widetilde{O}(n)$ queries $L = [n]$, and if we observe an odd-length cycle, we output "$\mathcal{G}_1$", otherwise, output "$\mathcal{G}_2$".

1.4 Preliminaries

We use boldfaced letters such as $\mathbf{A}, \mathbf{M}$ to denote random variables. Given a string $x \in \{0,1\}^n$ and $j \in [n]$, we write $x^{(j)}$ to denote the string obtained from x by flipping the j-th coordinate. An edge along the j-th direction in $\{0,1\}^n$ is a pair (x, y) of strings with $y = x^{(j)}$. In addition, for $\alpha \in \{0,1\}$ we use the notation $x^{(j \to \alpha)}$ to denote the string x where the jth coordinate is set to α. Given $x \in \{0,1\}^n$ and $S \subseteq [n]$, we use $x|_S \in \{0,1\}^S$ to denote the projection of x on S. For a distribution $\mathcal{D}$ we write $\boldsymbol{d} \sim \mathcal{D}$ to denote an element d drawn according to the distribution. We sometimes write $a \approx b \pm c$ to denote $b - c \leq a \leq b + c$.

2 The Rejection Sampling Model

In this section, we define the rejection sampling model and the distributions over graphs we will use throughout this work. We define the rejection sampling model tailored to our specific application of proving Lemma 2.

▶ **Definition 6.** Consider two distributions, $\mathcal{G}_1$ and $\mathcal{G}_2$ supported on graphs with vertex set $[n]$. The problem of distinguishing $\mathcal{G}_1$ and $\mathcal{G}_2$ with a rejection sampling oracle aims to distinguish between the following two cases with a specific kind of query:

- Cases: We have an unknown graph $\mathbf{G} \sim \mathcal{G}_1$ or $\mathbf{G} \sim \mathcal{G}_2$.
- Rejection Sampling Oracle: Each query is a subset $L \subseteq [n]$; an oracle samples an edge $(\boldsymbol{j}_1, \boldsymbol{j}_2)$ from $\mathbf{G}$ uniformly at random, and the oracle returns $\boldsymbol{v} = \{\boldsymbol{j}_1, \boldsymbol{j}_2\} \cap L$. The complexity of a query L is given by $|L|$.

We say a non-adaptive algorithm Alg for this problem is a sequence of query sets $L_1, \ldots, L_q \subseteq [n]$, as well as a function $\mathrm{Alg}\colon ([n] \cup ([n] \times [n]) \cup \{\emptyset\})^q \to \{\text{"}\mathcal{G}_1\text{"}, \text{"}\mathcal{G}_2\text{"}\}$. The algorithm sends each query to the oracle, and for each query L_i, the oracle responds $\boldsymbol{v}_i \in [n] \cup ([n] \times [n]) \cup \{\emptyset\}$, which is either a single element of $[n]$, an edge in $\mathbf{G}$, or $\emptyset$. The algorithm succeeds if:

$$\Pr_{\substack{\mathbf{G} \sim \mathcal{G}_1, \\ \boldsymbol{v}_1, \ldots, \boldsymbol{v}_q}} [\mathrm{Alg}(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_q) \text{ outputs "}\mathcal{G}_1\text{"}] - \Pr_{\substack{\mathbf{G} \sim \mathcal{G}_2, \\ \boldsymbol{v}_1, \ldots, \boldsymbol{v}_q}} [\mathrm{Alg}(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_q) \text{ outputs "}\mathcal{G}_1\text{"}] \geq \frac{1}{3}.$$

The complexity of Alg is measured by the sum of the complexity of the queries, so we let $\text{cost}(\text{Alg}) = \sum_{i=1}^{q} |L_i|$.

While our interest in this work is primarily on lower bounds for the rejection sampling model, an interesting direction is to explore upper bounds of various natural graph properties with rejection sampling queries. Our specific applications only require ruling out non-adaptive algorithms, but one may define adaptive algorithms in the rejection sampling model and study the power of adaptivity in this setting as well.

2.1 The Distributions $\mathcal{G}_1$ and $\mathcal{G}_2$

Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be two distributions supported on graphs with vertex set $[n]$ defined as follows. Let $\mathbf{A} \subseteq [n]$ be a uniform random subset of size $\frac{n}{2}$.

$$\mathcal{G}_1 = \left\{ K_{\mathbf{A}} \cup K_{\overline{\mathbf{A}}} : \mathbf{A} \subseteq [n] \text{ random subset size } \frac{n}{2} \right\}$$
$$\mathcal{G}_2 = \left\{ K_{\mathbf{A},\overline{\mathbf{A}}} : \mathbf{A} \subseteq [n] \text{ random subset size } \frac{n}{2} \right\},$$

where for a subset A, K_A is the complete graph on vertices in A and $K_{A,\overline{A}}$ is the complete bipartite graph whose sides are A and $\overline{A}$.

3 Tolerant Junta Testing

In this section, we will prove that distinguishing the two distributions $\mathcal{G}_1$ and $\mathcal{G}_2$ using a rejection sampling oracle reduces to distinguishing two distributions $\mathcal{D}_{\text{yes}}$ and $\mathcal{D}_{\text{no}}$ over Boolean functions, where $\mathcal{D}_{\text{yes}}$ is supported on functions that are close to k-juntas and $\mathcal{D}_{\text{no}}$ is supported on functions that are far from any k-junta with high probability.

3.1 High Level Overview

We start by providing some intuition of how our constructions and reduction implement the plan set forth in Subsection 1.3 for the property of being a k-junta. We define two distributions supported on Boolean functions, $\mathcal{D}_{\text{yes}}$ and $\mathcal{D}_{\text{no}}$, so that functions in $\mathcal{D}_{\text{yes}}$ are ε_0-close to being k-juntas and functions in $\mathcal{D}_{\text{no}}$ are ε_1-far from being k-juntas (where ε_0 and ε_1 are appropriately defined constants and $k = \frac{3n}{4}$).

As mentioned in the introduction, our distributions are based on the indexing function used in [17]. We draw a uniform random subset $\mathbf{M} \subseteq [n]$ of size $n/2$ and our function $\mathbf{\Gamma} = \Gamma_{\mathbf{M}} \colon \{0,1\}^n \to [2^{n/2}]$ projects the points onto the variables in $\mathbf{M}$. Thus, it remains to define the sequence of functions $\mathbf{H} = (\boldsymbol{h}_i \colon \{0,1\}^n \to \{0,1\} : i \in [2^{n/2}])$.

We will sample a graph $\mathbf{G} \sim \mathcal{G}_1$ (in the case of $\mathcal{D}_{\text{yes}}$), and a graph $\mathbf{G} \sim \mathcal{G}_2$ (in the case of $\mathcal{D}_{\text{no}}$) supported on vertices in $\overline{\mathbf{M}}$. Each function $\boldsymbol{h}_i \colon \{0,1\}^n \to \{0,1\}$ is given by first sampling an edge $(\boldsymbol{j}_1, \boldsymbol{j}_2) \sim \mathbf{G}$ and letting $\boldsymbol{h}_i$ be a parity (or a negated parity) of the variables $x_{\boldsymbol{j}_1}$ and $x_{\boldsymbol{j}_2}$. Thus, a function $\boldsymbol{f}$ from $\mathcal{D}_{\text{yes}}$ or $\mathcal{D}_{\text{no}}$ will have all variables being relevant, however, we will see that functions in $\mathcal{D}_{\text{yes}}$ have a group of $\frac{n}{4}$ variables which can be eliminated efficiently[6].

We think of the sub-functions $\boldsymbol{h}_i$ defined with respect to edges from $\mathbf{G}$ as implementing a sort of *gadget*: the gadget defined with respect to an edge (j_1, j_2) will have the property that if $\boldsymbol{f}$ eliminates the variable j_1, it will be "encouraged" to eliminate variable j_2 as well.

[6] We say that a variable is eliminated if we change the function to remove the dependence of the variable.

In fact, each time an edge $(\boldsymbol{j}_1, \boldsymbol{j}_2) \sim \mathbf{G}$ is used to define a sub-function $\boldsymbol{h}_i$, any k-junta $g\colon \{0,1\}^n \to \{0,1\}$ where variable $\boldsymbol{j}_1$ or $\boldsymbol{j}_2$ is irrelevant will have to change half of the corresponding part indexed by $\boldsymbol{\Gamma}$. Intuitively, a function $\boldsymbol{f} \sim \mathcal{D}_{\text{yes}}$ or $\mathcal{D}_{\text{no}}$ (which originally depends on all n variables) wants to eliminate its dependence of $n-k$ variables in order to become a k-junta. When $\boldsymbol{f}$ picks a variable $j \in \overline{\mathbf{M}}$ to eliminate (since variables in $\mathbf{M}$ are too expensive), it must change points in parts where the edge sampled is incident on j. The key observation is that when $\boldsymbol{f}$ needs to eliminate multiple variables, if $\boldsymbol{f}$ picks the variables j_1 and j_2 to eliminate, whenever a part samples the edge (j_1, j_2), the function changes the points in one part and eliminates two variables. Thus, $\boldsymbol{f}$ eliminates two variables by changing the same number of points when there are edges between j_1 and j_2.

At a high level, the gadgets encourage the function $\boldsymbol{f}$ to remove the dependence of variables within a group of edges, i.e., the closest k-junta will correspond to a function g which eliminates groups of variables with edges within each other and few outgoing edges. More specifically, if we want to eliminate $\frac{n}{4}$ variables from $\boldsymbol{f}$, we must find a bisection of the graph $\mathbf{G}$ whose cut value is small; in the case of $\mathcal{G}_1$, one of the cliques will have cut value 0, whereas any bisection of a graph from $\mathcal{G}_2$ will have a high cut value, which makes functions in $\mathcal{D}_{\text{yes}}$ closer to $\frac{3n}{4}$-juntas than functions in $\mathcal{D}_{\text{no}}$.

The reduction from rejection sampling is straight-foward. We consider all queries which are indexed to the same part, and if two queries indexed to the same part differ on a variable j, then the algorithm "explores" direction j. Each part $i \in [2^{n/2}]$ where some query falls in has a corresponding rejection sampling query L_i, which queries the variables explored by the Boolean function testing algorithm.

3.2 The Distributions $\mathcal{D}_{\text{yes}}$ and $\mathcal{D}_{\text{no}}$

The goal of this subsection is to define the two distributions $\mathcal{D}_{\text{yes}}$ and $\mathcal{D}_{\text{no}}$, supported over Boolean functions with n variables. Functions $f \in \mathcal{D}_{\text{yes}}$ will be *close* to being a k-junta with high probability, and functions $f \sim \mathcal{D}_{\text{no}}$ will be *far* from any k-junta with high probability. We note that it suffices to consider $k = \frac{3n}{4}$ to obtain Theorem 3. We refer the reader to the full version of the paper for the reduction from arbitrary k to $k = \frac{3n}{4}$.

Distribution $\mathcal{D}_{\text{yes}}$. A function f from $\mathcal{D}_{\text{yes}}$ is generated from a tuple of three random variables, $(\mathbf{M}, \mathbf{A}, \mathbf{H})$, and we set $f = f_{\mathbf{M},\mathbf{A},\mathbf{H}}$. The tuple is drawn according to the following randomized procedure:

1. Sample a uniformly random subset $\mathbf{M} \subseteq [n]$ of size $m \stackrel{\text{def}}{=} \frac{n}{2}$. Let $N = 2^m$ and $\Gamma_{\mathbf{M}} : \{0,1\}^n \to [N]$ be the function that maps $x \in \{0,1\}^n$ to a number encoded by $x|_{\mathbf{M}} \in [N]$.
2. Sample $\mathbf{A} \subseteq \overline{\mathbf{M}}$ of size $\frac{n}{4}$ uniformly at random, and consider the graph $\mathbf{G}$ defined on vertices $[\overline{\mathbf{M}}]$ with $\mathbf{G} = K_{\mathbf{A}} \cup K_{\overline{\mathbf{A}}}$, i.e., $\mathbf{G}$ is a uniformly random graph drawn according to $\mathcal{G}_1$.
3. Define a sequence of N functions $\mathbf{H} = \{\boldsymbol{h}_i\colon \{0,1\}^n \to \{0,1\} : i \in [N]\}$ drawn from a distribution $\mathcal{E}(\mathbf{G})$. For each $i \in \{1, \ldots, N/2\}$, we let $\boldsymbol{h}_i(x) = \bigoplus_{\ell \in \mathbf{M}} x_\ell$.
   For each $i \in \{N/2+1, \ldots, N\}$, we will generate $\boldsymbol{h}_i$ independently by sampling an edge $(\boldsymbol{j}_1, \boldsymbol{j}_2) \sim \mathbf{G}$ uniformly at random, as well as a uniform random bit $\boldsymbol{r} \sim \{0,1\}$. We let

   $$\boldsymbol{h}_i(x) = x_{\boldsymbol{j}_1} \oplus x_{\boldsymbol{j}_2} \oplus \boldsymbol{r}.$$

4. Using $\mathbf{M}, \mathbf{A}$ and $\mathbf{H}$, define $f_{\mathbf{M},\mathbf{A},\mathbf{H}} = \boldsymbol{h}_{\Gamma_{\mathbf{M}}(x)}(x)$ for each $x \in \{0,1\}^n$.

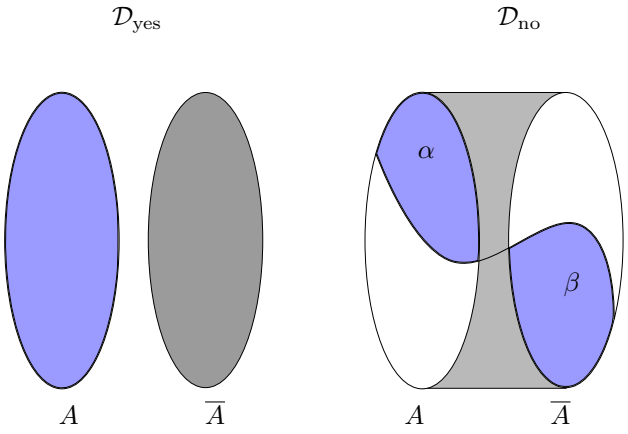

Figure 1 Example of graphs $\mathbf{G}$ from $\mathcal{D}_{\text{yes}}$ and $\mathcal{D}_{\text{no}}$. On the left, the graph $\mathbf{G}$ is the union of two cliques of size $\frac{n}{4}$, corresponding to $\mathcal{D}_{\text{yes}}$. We note that $\chi(G) = \frac{1}{2}$, since if we let $S = \mathbf{A}$ (pictured as the blue set), we see that S contains half of the edges. On the right, the graph $\mathbf{G}$ is the complete bipartite graph with side sizes $\frac{n}{4}$, corresponding to $\mathcal{D}_{\text{no}}$. We note that $\chi(G) = \frac{3}{4}$: consider any set $S \subseteq \overline{M}$ of size at least $\frac{n}{4}$ pictured in the blue region, and let $\alpha = |S \cap A|$ and $\beta = |S \cap \overline{A}|$, where $\alpha + \beta \geq \frac{n}{4}$, so $E(S,S) + E(S,\overline{S}) \geq (\frac{n}{4})^2 - \alpha\beta \geq (\frac{n}{4})^2(1 - \frac{1}{4})$.

Distribution $\mathcal{D}_{\text{no}}$. A function f drawn from $\mathcal{D}_{\text{no}}$ is also generated by first drawing the tuple $(\mathbf{M}, \mathbf{A}, \mathbf{H})$ and setting $f = f_{\mathbf{M},\mathbf{A},\mathbf{H}}$. Both $\mathbf{M}$ and $\mathbf{A}$ are drawn using the same procedure; the only difference is that the graph $\mathbf{G} = K_{\mathbf{A},\overline{\mathbf{A}}}$, i.e., $\mathbf{G}$ is a uniformly random graph drawn according to $\mathcal{G}_2$. Then $\mathbf{H} \sim \mathcal{E}(\mathbf{G})$ is sampled from the modified graph $\mathbf{G}$.

We let $k \stackrel{\text{def}}{=} \frac{3n}{4}, \varepsilon_0 \stackrel{\text{def}}{=} \frac{1}{8}$, and $\varepsilon_1 \stackrel{\text{def}}{=} \frac{3}{16}$. Consider a fixed subset $M \subseteq [n]$ which satisfies $|M| = \frac{n}{2}$, and a fixed subset $A \subseteq \overline{M}$ which satisfies $|A| = \frac{n}{4}$. Let G be a graph defined over vertices in $\overline{M}$, and for any subsets $S_1, S_2 \subseteq \overline{M}$, let $E_G(S_1, S_2) = |\{(j_1, j_2) \in G : j_1 \in S_1, j_2 \in S_2\}|$, be the number of edges between sets S_1 and S_2. Additionally, we let

$$\chi(G) = \min\left\{ \frac{E_G(S,S) + E_G(S,\overline{S})}{E_G(\overline{M},\overline{M})} : S \subseteq \overline{M}, |S| \geq \frac{n}{4} \right\} \tag{1}$$

be the minimum fraction of edges adjacent to a set S of size at least $\frac{n}{4}$. The following lemma relates the distance of a function $\boldsymbol{f} = f_{M,A,\mathbf{H}}$ where $\mathbf{H} \sim \mathcal{E}(G)$ to being a k-junta to $\chi(G)$. We then apply this lemma to the graph in $\mathcal{D}_{\text{yes}}$ and $\mathcal{D}_{\text{no}}$ to show that functions in $\mathcal{D}_{\text{yes}}$ are ε_0-close to being k-juntas, and functions in $\mathcal{D}_{\text{no}}$ are ε_1-far from being k-juntas.

▶ **Lemma 7.** *Let G be any graph defined over vertices in A. If $\boldsymbol{f} = f_{M,A,\mathbf{H}}$, where $\mathbf{H} \sim \mathcal{E}(G)$, then with probability at least $1 - o(1)$,*

$$\frac{1}{4} \cdot \chi(G) - o(1) \leq \text{dist}(\boldsymbol{f}, k\text{-Junta}) \leq \frac{1}{4} \cdot \chi(G) + o(1).$$

▶ **Corollary 8.** *We have that $\boldsymbol{f} \sim \mathcal{D}_{yes}$ has $\text{dist}(\boldsymbol{f}, k\text{-Junta}) \leq \varepsilon_0 + o(1)$ with probability $1 - o(1)$, and that $\boldsymbol{f} \sim \mathcal{D}_{no}$ has $\text{dist}(\boldsymbol{f}, k\text{-Junta}) \geq \varepsilon_1 - o(1)$ with probability $1 - o(1)$.*

The proof shows that distinguishing the two distributions $\mathcal{G}_1$ and $\mathcal{G}_2$ using rejection sampling oracle reduces to distinguishing the two distributions $\mathcal{D}_{\text{yes}}$ and $\mathcal{D}_{\text{no}}$.

▶ **Lemma 9.** *Suppose there exists a deterministic non-adaptive algorithm* Alg *making* q *queries to Boolean functions* $f\colon \{0,1\}^{2n} \to \{0,1\}$*. Then, there exists a deterministic non-adaptive algorithm* Alg$'$ *making rejection sampling queries to an* n*-vertex graph such that:*

$$\Pr_{\boldsymbol{f}\sim\mathcal{D}_{yes}}[\text{Alg}(\boldsymbol{f}) \text{ ``accepts''}] = \Pr_{\mathbf{G}\sim\mathcal{G}_1}[\text{Alg}'(\mathbf{G}) \text{ outputs ``}\mathcal{G}_1\text{''}], \quad \text{and}$$
$$\Pr_{\boldsymbol{f}\sim\mathcal{D}_{no}}[\text{Alg}(\boldsymbol{f}) \text{ ``accepts''}] = \Pr_{\mathbf{G}\sim\mathcal{G}_2}[\text{Alg}'(\mathbf{G}) \text{ outputs ``}\mathcal{G}_1\text{''}].$$

and has $\text{cost}(\text{Alg}') = O(q \log n)$ *with probability* $1 - o(1)$ *over the randomness in* Alg$'$*.*

4 Tolerant Unateness Testing

In this section, we show how to reduce distinguishing distributions $\mathcal{G}_1$ and $\mathcal{G}_2$ to distinguishing between Boolean functions which are close to unate and Boolean functions which are far from unate. We start with a high level overview of the constructions and reduction, and then proceed to give formal definitions and the reductions for adaptive and non-adaptive tolerant testing.

4.1 High Level Overview

We now describe how our constructions and reduction implement the plan set forth in Subsection 1.3 for the property of unateness. Similarly to Section 3, we define two distributions $\mathcal{D}_{\text{yes}}$ and $\mathcal{D}_{\text{no}}$ supported on Boolean functions, so that functions in $\mathcal{D}_{\text{yes}}$ are ε_0-close to being unate, and functions in $\mathcal{D}_{\text{no}}$ are ε_1-far from being unate (where ε_0 and ε_1 are appropriately defined constants).

We will use a randomized indexing function $\mathbf{\Gamma}\colon \{0,1\}^n \to [N] \cup \{0^*, 1^*\}$ based on the Talagrand-style constructions from [4, 18] to partition $\{0,1\}^n$ in a unate fashion, specifically, $\mathbf{\Gamma}$ will satisfy that for all $i \neq j \in [N]$, if $x, y \in \{0,1\}^n$ have $\mathbf{\Gamma}(x) = i$ and $\mathbf{\Gamma}(y) = j$, then x and y are *incomparable*, $x \not\prec y$ and $y \not\prec x$. Again, we will then use a graph $\mathbf{G} \sim \mathcal{G}_1$ or $\mathcal{G}_2$ to define the sequence of sub-function $\mathbf{H} = (\boldsymbol{h}_i\colon \{0,1\}^n \to \{0,1\} : i \in [N])$. The sub-functions $\boldsymbol{h}_i$ will be given by a parity (or negated parity) of three variables: two variables will correspond to the end points of an edge sampled $(\boldsymbol{j}_1, \boldsymbol{j}_2) \sim \mathcal{G}$, the third variable will be one of two pre-specified variables, which we call m_1 and m_2. Consider for simplicity the case when $\boldsymbol{h}_i(x) = x_{\boldsymbol{j}_1} \oplus x_{\boldsymbol{j}_2} \oplus x_{m_1}$, and assume that we require that variable m_1 is non-decreasing.

Similarly to Section 3, the functions $\boldsymbol{h}_i$ are thought of as gadgets. We will have that if $\boldsymbol{h}_i$ is defined with respect to an edge (j_1, j_2) and m_1, then the function $\boldsymbol{f}$ will be "encouraged" to make variables j_1 and j_2 have opposite directions, i.e., either j_1 is non-increasing and j_2 is non-decreasing, or j_1 is non-decreasing and j_2 is non-increasing. In order to see why the three variable parity implements this gadget, we turn our attention to Figure 2 and Figure 3.

Intuitively, the function $\boldsymbol{f}$ needs to change some of its inputs to be unate, and it must choose whether the variables j_1 and j_2 will be monotone (non-decreasing) or anti-monotone (non-increasing). Suppose $\boldsymbol{f}$ decides that the variable j_1 should be monotone and j_2 be anti-monotone, and m_1 will always be monotone (since it will be too expensive to make it anti-monotone). Then, when $\boldsymbol{h}_i(x) = x_{j_1} \oplus x_{j_2} \oplus x_{m_1}$, $\boldsymbol{h}_i$ will have some *violating edges*, i.e., edges in direction j_1 which are decreasing, or edges in direction j_2 which are increasing, or edges in direction m_1 which are decreasing (see Figure 2, where these violating edges are marked in red). In this case, there exists a way that $\boldsymbol{f}$ may change $\frac{1}{4}$-th fraction of the points and remove all violating edges (again, this procedure is shown in Figure 2).

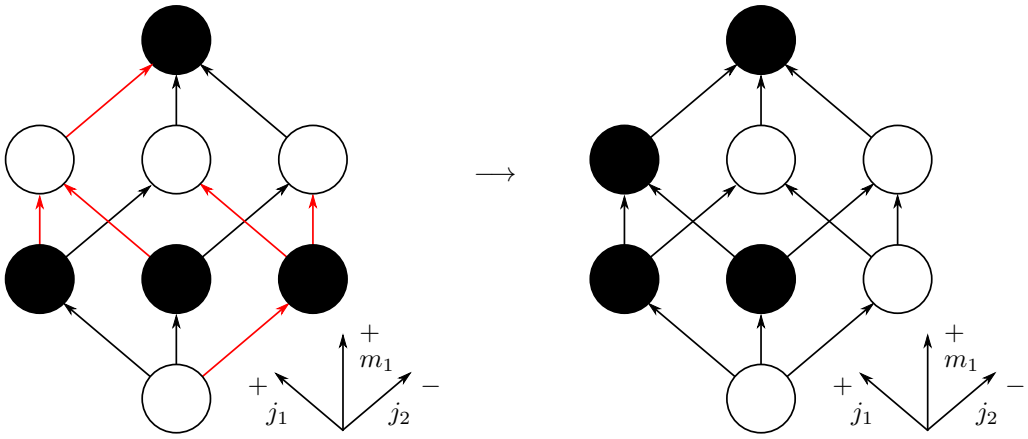

Figure 2 Example of a function $\boldsymbol{h}_i\colon \{0,1\}^n \to \{0,1\}$ with $\boldsymbol{h}_i(x) = x_{j_1} \oplus x_{j_2} \oplus x_{m_1}$ with variable j_1 (which ought to be monotone), j_2 (which ought to be anti-monotone), and m_1 (which is always monotone). The image on the left-hand side represents $\boldsymbol{h}_i$, and the red edges correspond to violating edges for variables j_1, j_2 and m_1. In other words, the red edges correspond to anti-monotone edges in variables j_1, monotone edges in variables j_2, and anti-monotone edges in direction m_1. On the right-hand side, we show how such a function can being "fixed" into a function $\boldsymbol{h}'_i\colon \{0,1\}^n \to \{0,1\}$ by changing $\frac{1}{4}$-fraction of the points.

In contrast, suppose that $\boldsymbol{f}$ decides that the variables j_1 and j_2 both should be monotone. Then, when $\boldsymbol{h}_i(x) = x_{j_1} \oplus x_{j_2} \oplus x_{m_1}$, the violating edges (shown in Figure 3) form vertex-disjoint cycles of length 6 in $\{0,1\}^n$, thus, the function $\boldsymbol{f}$ will have to change $\frac{3}{8}$-th fraction of the points in order to remove all violating edges. In other words, when there is an edge (j_1, j_2) sampled in $\boldsymbol{h}_i$, the function $\boldsymbol{f}$ is "encouraged" to make j_1 and j_2 have opposite directions, and "discouraged" to make j_1 and j_2 have the same direction. The other cases are presented in Figures 4, 5, and 6.

In order for $\boldsymbol{f}$ to become unate, it must first choose whether each variable will be monotone or anti-monotone. $\boldsymbol{f}$ will choose all variables in $\mathbf{M}$ to be monotone, the variable m_1 to be monotone, and m_2 to be anti-monotone, but will have to make a choice for each variable in $\overline{\mathbf{M}}$, corresponding to each vertex of the graph $\mathbf{G}$. As discussed above, for each edge (j_1, j_2) in the graph, $\boldsymbol{f}$ is encouraged to make these orientations opposite from each other, so $\boldsymbol{f}$ will want to look for the maximum cut on the graph, whose value will be different in $\mathcal{G}_1$ and $\mathcal{G}_2$.

Similarly to the case in Section 3, the reduction will follow by defining the rejection sampling queries L_i corresponding to variables explored in sub-function $\boldsymbol{h}_i$. The unate indexing functions $\boldsymbol{\Gamma}$ are not as strong as the indexing functions from the Section 3, so for each query in the Boolean function testing algorithm, our reduction will lose some cost in the rejection sampling algorithm. In particular, the adaptive reduction loses n cost for each Boolean function query, since adaptive algorithms can efficiently explore variables with a binary search; this gives the $\widetilde{\Omega}(n)$ lower bound for tolerant unateness testing. The non-adaptive reduction loses $O(\sqrt{n}\log n)$ cost for each Boolean function query since queries falling in the same part may be $\Omega(\sqrt{n})$ away from each other (the same scenario occurs in the non-adaptive monotonicity lower bound of [18]). The non-adaptive reduction is more complicated than the adaptive reduction since it is not exactly a black-box reduction (we require a lemma from Section 5). This gives the $\widetilde{\Omega}(n^{3/2})$ lower bound for non-adaptive tolerant unateness testing.

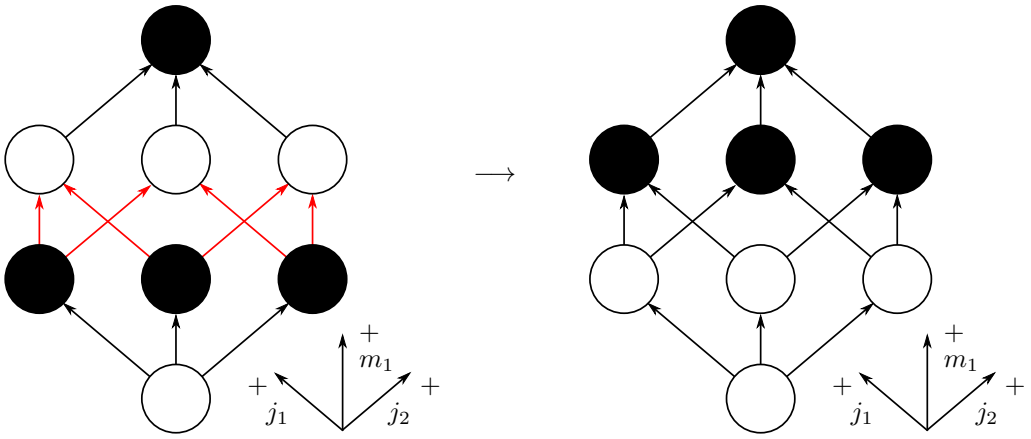

Figure 3 Example of a function $\boldsymbol{h}_i\colon \{0,1\}^n \to \{0,1\}$ with $\boldsymbol{h}_i(x) = x_{j_1} \oplus x_{j_2} \oplus x_{m_1}$ with variables j_1 and j_2 (which ought to be monotone), and m_1 (which ought to be monotone). On the left side, we indicate the violating edges with red arrows, and note that the functions in the left and right differ by $\frac{3}{8}$-fraction of the points. We also note that any function $\boldsymbol{h}'_i\colon \{0,1\}^n \to \{0,1\}$ which has j_1, j_2 and m_1 monotone must differ from $\boldsymbol{h}_i$ on at least $\frac{3}{8}$-fraction of the points because the violating edges of $\boldsymbol{h}_i$ form a cycle of length 6.

4.2 The Distributions $\mathcal{D}_{\text{yes}}$ and $\mathcal{D}_{\text{no}}$

We now turn to describing a pair of distributions $\mathcal{D}_{\text{yes}}$ and $\mathcal{D}_{\text{no}}$ supported on Boolean functions $f\colon \{0,1\}^n \to \{0,1\}$. These distributions will have the property that for some constants ε_0 and ε_1 with $0 < \varepsilon_0 < \varepsilon_1$,

$$\Pr_{\boldsymbol{f}\sim\mathcal{D}_{\text{yes}}}[\text{dist}(\boldsymbol{f}, \text{Unate}) \leq \varepsilon_0] = 1 - o(1) \qquad \text{and} \qquad \Pr_{\boldsymbol{f}\sim\mathcal{D}_{\text{no}}}[\text{dist}(\boldsymbol{f}, \text{Unate}) \geq \varepsilon_1] = 1 - o(1).$$

We first define a function $\boldsymbol{f} \sim \mathcal{D}_{\text{no}}$, where we fix the parameter $N = 2^{\sqrt{n}}$.

1. Sample some set $\mathbf{M} \subseteq [n]$ of size $|\mathbf{M}| = \frac{n}{2}$ uniformly at random and let $\boldsymbol{m}_1, \boldsymbol{m}_2 \sim \mathbf{M}$ be two distinct indices.
2. We let $\mathbf{T} \sim \mathcal{E}(\mathbf{M} \setminus \{\boldsymbol{m}_1, \boldsymbol{m}_2\})$ (which we describe next). $\mathbf{T}$ is a sequence of terms $(\mathbf{T}_i : i \in [N])$ which is used to defined a multiplexer map $\Gamma_{\mathbf{T}}\colon \{0,1\}^n \to [N] \cup \{0^*, 1^*\}$.
3. We sample $\mathbf{A} \subseteq \overline{\mathbf{M}}$ of size $|\mathbf{A}| = \frac{n}{2}$ and define a graph as $\mathbf{G} = K_{\mathbf{A}} \cup K_{\overline{\mathbf{A}}}$.
4. We now define the distribution over sub-functions $\mathbf{H} = (\boldsymbol{h}_i : i \in [N]) \sim \mathcal{H}(\boldsymbol{m}_1, \boldsymbol{m}_2, \mathbf{G})$. For each function $\boldsymbol{h}_i\colon \{0,1\}^n \to \{0,1\}$, we generate $\boldsymbol{h}_i$ independently:
   - When $i \leq 3N/4$, we sample $\boldsymbol{j} \sim \{\boldsymbol{m}_1, \boldsymbol{m}_2\}$ and we let:

$$\boldsymbol{h}_i(x) = \begin{cases} x_{\boldsymbol{j}} & \boldsymbol{j} = \boldsymbol{m}_1 \\ \neg x_{\boldsymbol{j}} & \boldsymbol{j} = \boldsymbol{m}_2 \end{cases}.$$

   - Otherwise, if $i > 3N/4$, we sample an edge $(\boldsymbol{j}_1, \boldsymbol{j}_2) \sim \mathbf{G}$ and an index $\boldsymbol{j}_3 \sim \{\boldsymbol{m}_1, \boldsymbol{m}_2\}$ we let:

$$\boldsymbol{h}_i(x) = \begin{cases} x_{\boldsymbol{j}_1} \oplus x_{\boldsymbol{j}_2} \oplus x_{\boldsymbol{j}_3} & \boldsymbol{j}_3 = \boldsymbol{m}_1 \\ \neg x_{\boldsymbol{j}_1} \oplus x_{\boldsymbol{j}_2} \oplus x_{\boldsymbol{j}_3} & \boldsymbol{j}_3 = \boldsymbol{m}_2 \end{cases}.$$

The function $\boldsymbol{f}\colon \{0,1\}^n \to \{0,1\}$ is given by $\boldsymbol{f}(x) = f_{\mathbf{T},\mathbf{A},\mathbf{H}}(x)$ where:

$$f_{\mathbf{T},\mathbf{A},\mathbf{H}}(x) = \begin{cases} 1 & |x_{|\mathbf{M}}| > \frac{n}{4} + \sqrt{n} \\ 0 & |x_{|\mathbf{M}}| < \frac{n}{4} - \sqrt{n} \\ 1 & \Gamma_{\mathbf{T}}(x) = 1^* \\ 0 & \Gamma_{\mathbf{T}}(x) = 0^* \\ \boldsymbol{h}_{\Gamma_{\mathbf{T}}(x)}(x) & \text{otherwise} \end{cases} . \tag{2}$$

We now turn to define the distribution $\mathcal{E}(M)$ supported on terms $\mathbf{T}$, as well as the multiplexer map $\Gamma_{\mathbf{T}}\colon \{0,1\}^n \to [N]$. As mentioned above, $\mathbf{T} \sim \mathcal{E}(M)$ will be a sequence of N terms $(\mathbf{T}_i : i \in [N])$, where each $\mathbf{T}_i$ is given by a DNF term: $\mathbf{T}_i(x) = \bigwedge_{j \in \mathbf{T}_i} x_j$, where the set $\mathbf{T}_i \subseteq M$ is a uniformly random $\sqrt{n}$-element subset. Given the sequence of terms $\mathbf{T}$, we let:

$$\Gamma_{\mathbf{T}}(x) = \begin{cases} 0^* & \forall i \in [N], \mathbf{T}_i(x) = 0 \\ 1^* & \exists i_1 \neq i_2 \in [N], \mathbf{T}_{i_1}(x) = \mathbf{T}_{i_2}(x) = 1 \\ i & \mathbf{T}_i(x) = 1 \text{ for a unique } i \in [N] \end{cases} .$$

It remains to define the distribution $\mathcal{D}_{\text{yes}}$ supported on Boolean functions. The function $\boldsymbol{f} \sim \mathcal{D}_{\text{yes}}$ will be defined almost exactly the same. We still have $\boldsymbol{f} = f_{\mathbf{T},\mathbf{A},\mathbf{H}}$ as defined above, however, the graph $\mathbf{G}$ will be different. In particular, we will let $\mathbf{G} = K_{\mathbf{A},\overline{\mathbf{A}}}$.

Fix any set $M \subseteq [n]$ of size $\frac{n}{2}$ and let $m_1, m_2 \in M$ be two distinct indices and $M' = M \setminus \{m_1, m_2\}$. For any $\mathbf{T} \sim \mathcal{E}(M')$, let $\mathbf{X} \subseteq \{0,1\}^n$ be the subset of points indexed to some subfunction $\boldsymbol{h}_i$:

$$\mathbf{X} \stackrel{\text{def}}{=} \left\{x \in \{0,1\}^n : |x_{|M}| \in [n/4 - \sqrt{n}, n/4 + \sqrt{n}] \text{ and } \Gamma_T(x) \in [N]\right\},$$

and define $\gamma \in (0,1)$ be the parameter: $\gamma \stackrel{\text{def}}{=} \mathbb{E}_{\mathbf{T} \sim \mathcal{E}(M')}\left[\frac{|\mathbf{X}|}{2^n}\right]$.

In addition, let $X_i \subseteq X$ be the subset of points $x \in X$ with $\Gamma_T(x) = i$, and note that the subsets $X_1, \ldots, X_N$ partition X, where each $|X_i| \leq 2^{n-\sqrt{n}}$. With probability $1 - o(1)$ over the draw of $\mathbf{T} \sim \mathcal{E}(M)$, we have:

$$\sum_{i=1}^{3N/4} |\mathbf{X}_i| = 2^n \cdot \frac{3\gamma}{4}\left(1 \pm \frac{1}{n}\right) \qquad \text{and} \qquad \sum_{i=3N/4+1}^{N} |\mathbf{X}_i| = 2^n \cdot \frac{\gamma}{4}\left(1 \pm \frac{1}{n}\right). \tag{3}$$

Thus, we only consider functions $\boldsymbol{f} \sim \mathcal{D}_{\text{yes}}$ (or $\sim \mathcal{D}_{\text{no}}$) where the sets M, and T satisfy (3).

We consider any set $A \subseteq \overline{M}$ of size $\frac{n}{4}$. Now, consider any graph G defined over vertices in $\overline{M}$, and we let:

$$\chi(G) = \min\left\{\frac{E_G(S,S) + E_G(\overline{S},\overline{S})}{E_G(\overline{M},\overline{M})} : S \subseteq \overline{M}\right\}.$$

In other words, we note that $\chi(G)$ is one minus the fractional value of the maximum cut, and the value of $\chi(G)$ is minimized for the set S achieving the maximum cut of G. The following lemma relates the distance to unateness of a function $\boldsymbol{f} = f_{T,A,\mathbf{H}}$ with $\mathbf{H} \sim \mathcal{H}(m_1, m_2, G)$, where G is an underlying graph defined on vertices in $\overline{M}$.

▶ **Lemma 10.** *Let G be any graph defined over vertices in $\overline{M}$. If $\boldsymbol{f} = f_{T,A,\mathbf{H}}$ where $\mathbf{H} \sim \mathcal{H}(m_1, m_2, G)$, then with probability at least $1 - o(1)$,*

$$\frac{\gamma}{16}\left(1 + \frac{1}{2} \cdot \chi(G)\right) - o(1) \leq \text{dist}(\boldsymbol{f}, \text{Unate}) \leq \frac{\gamma}{16}\left(1 + \frac{1}{2} \cdot \chi(G)\right) + o(1).$$

We consider the constants $\varepsilon_0 = \frac{\gamma}{16}$ and $\varepsilon_1 = \frac{5\gamma}{64}$.

▶ **Corollary 11.** *We have that $\boldsymbol{f} \sim \mathcal{D}_{yes}$ has $\text{dist}(\boldsymbol{f}, \text{Unate}) \leq \varepsilon_0 + o(1)$ with high probability, and $\boldsymbol{f} \sim \mathcal{D}_{no}$ has $\text{dist}(\boldsymbol{f}, \text{Unate}) \geq \varepsilon_1 - o(1)$ with high probability.*

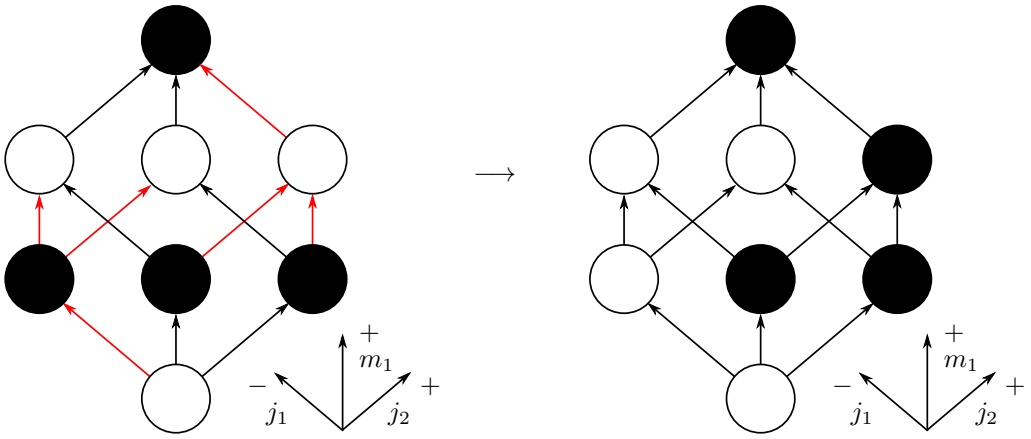

Figure 4 Similarly to Figure 2, this is an example of a function $\boldsymbol{h}_i\colon \{0,1\}^n \to \{0,1\}$ with $\boldsymbol{h}_i(x) = x_{j_1} \oplus x_{j_2} \oplus x_{m_1}$ variables j_1 (which ought to be anti-monotone), j_2 (which ought to be monotone), and m_1 (which is always monotone) being "fixed" into a function $\boldsymbol{h}'_i\colon \{0,1\}^n \to \{0,1\}$ defined on the right-hand side.

4.3 Reducing from Rejection Sampling

In order to reduce from rejection sampling, we need the following two lemmas.

▶ **Lemma 12.** *Suppose there exists a deterministic algorithm* Alg *making q queries to Boolean functions $f\colon \{0,1\}^{2n} \to \{0,1\}$. Then, there exists a deterministic non-adaptive algorithm* Alg$'$ *making rejection sampling queries to an n-vertex graph with* $\mathrm{cost}(\mathrm{Alg}') = qn$ *such that:*

$$\Pr_{\boldsymbol{f}\sim\mathcal{D}_{yes}}[\mathrm{Alg}(\boldsymbol{f}) \text{ ``accepts''}] = \Pr_{\mathbf{G}\sim\mathcal{G}_2}[\mathrm{Alg}'(\mathbf{G}) \text{ outputs ``}\mathcal{G}_2\text{''}], \qquad \text{and}$$
$$\Pr_{\boldsymbol{f}\sim\mathcal{D}_{no}}[\mathrm{Alg}(\boldsymbol{f}) \text{ ``accepts''}] = \Pr_{\mathbf{G}\sim\mathcal{G}_1}[\mathrm{Alg}'(\mathbf{G}) \text{ outputs ``}\mathcal{G}_2\text{''}].$$

▶ **Lemma 13.** *Suppose there exists a deterministic non-adaptive algorithm* Alg *making q queries to Boolean functions $f\colon \{0,1\}^{2n} \to \{0,1\}$ where $q \le \frac{n^{3/2}}{\log^8 n}$. Then, there exists a deterministic non-adaptive algorithm* Alg$'$ *making rejection sampling queries to an n-vertex graph such that:*

$$\Pr_{\boldsymbol{f}\sim\mathcal{D}_{yes}}[\mathrm{Alg}(\boldsymbol{f}) \text{ ``accepts''}] \approx \Pr_{\mathbf{G}\sim\mathcal{G}_2}[\mathrm{Alg}'(\mathbf{G}) \text{ outputs ``}\mathcal{G}_2\text{''}] \pm o(1), \qquad \text{and}$$
$$\Pr_{\boldsymbol{f}\sim\mathcal{D}_{no}}[\mathrm{Alg}(\boldsymbol{f}) \text{ ``accepts''}] \approx \Pr_{\mathbf{G}\sim\mathcal{G}_1}[\mathrm{Alg}'(\mathbf{G}) \text{ outputs ``}\mathcal{G}_2\text{''}] \pm o(1).$$

and has $\mathrm{cost}(\mathrm{Alg}') \le q\sqrt{n}\log n$ *with probability* $1 - o(1)$ *over the randomness in* Alg$'$.

Combining Lemma 12 with Theorem 1, we conclude Theorem 4, and combining Lemma 13 with Theorem 1, we conclude Theorem 5.

5 A lower bound for distinguishing $\mathcal{G}_1$ and $\mathcal{G}_2$ with rejection samples

In this section, we derive a lower bound for distinguishing $\mathcal{G}_1$ and $\mathcal{G}_2$ with rejection samples.

▶ **Lemma 14.** *Any deterministic non-adaptive algorithm* Alg *with* $\mathrm{cost}(\mathrm{Alg}) \le \frac{n^2}{\log^6 n}$, *has:*

$$\Pr_{\mathbf{G}\sim\mathcal{G}_1}[\mathrm{Alg} \text{ outputs ``}\mathcal{G}_1\text{''}] \le (1 + o(1)) \Pr_{\mathbf{G}\sim\mathcal{G}_2}[\mathrm{Alg} \text{ outputs ``}\mathcal{G}_1\text{''}] + o(1).$$

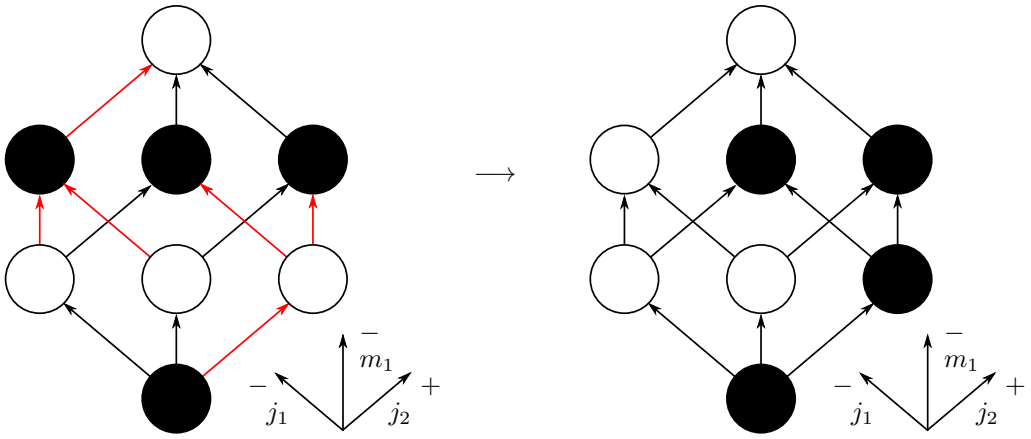

Figure 5 Similarly to Figure 2, this is an example of a function $\boldsymbol{h}_i\colon \{0,1\}^n \to \{0,1\}$ with $\boldsymbol{h}_i(x) = \neg x_{j_1} \oplus x_{j_2} \oplus x_{m_2}$ variables j_1 (which ought to be anti-monotone), j_2 (which ought to be monotone), and m_2 (which is always anti-monotone) being "fixed" into a function $\boldsymbol{h}'_i\colon \{0,1\}^n \to \{0,1\}$ defined on the right-hand side.

We assume Alg is a deterministic non-adaptive algorithm with $\text{cost}(\text{Alg}) \leq \frac{n^2}{\log^6 n}$. Alg makes queries $L_1, \ldots, L_t \subseteq [n]$ and the oracle returns $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_t$, some of which are edges, some are lone vertices, and some are $\emptyset$. Let $\mathbf{G}_o \subseteq \mathbf{G}$ be the graph observed by the algorithm by considering all edges in $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_t$. We let $|\mathbf{G}_o|$ be the number of edges.

Before going on to prove the lower bound, we use the following simplification. First, we assume that any algorithm Alg has all its queries $L_1, \ldots, L_t$ satisfying that either $|L_i| \leq \frac{n}{\log n}$, or $L_i = [n]$. Thus, it suffices to show for this restricted class of algorithms, the cost must be at least $\frac{n^2}{\log^5 n}$.

5.1 High Level Overview

We will argue outcome-by-outcome; i.e., we consider the possible ways the algorithm can act, which depends on the responses to the queries the algorithm gets. Consider some responses $v_1, \ldots, v_t \in [n] \cup ([n] \times [n]) \cup \{\emptyset\}$, where each v_i may be either a lone vertex, an edge, or $\emptyset$. Suppose that upon observing this outcome, the algorithm outputs "$\mathcal{G}_1$". There will be two cases:

- The first case is when the probability of observing this outcome from $\mathcal{G}_2$ is not too much lower than the probability of observing this outcome from $\mathcal{G}_1$. In these outcomes, we will not get too much advantage in distinguishing $\mathcal{G}_1$ and $\mathcal{G}_2$.
- The other case is when the probability of observing this outcome from $\mathcal{G}_2$ is substantially lower than the probability of observing this outcome from $\mathcal{G}_1$. These cases do help us distinguish between $\mathcal{G}_1$ and $\mathcal{G}_2$; thus, we will want to show that collectively, the probability that we observe these outcomes from $\mathcal{G}_1$ is $o(1)$.

We will be able to characterize the outcomes which fall into the first case and the second case by considering a sequence of events. In particular we define five events which depend on $v_1, \ldots, v_t$, as well as the random choice of $\mathbf{A}$. Consider the outcome $v_1, \ldots, v_t$ which together form components $C_1, \ldots, C_\alpha$. The events are the following[7]:

[7] We note that the first two event are not random and depends on the values $v_1, \ldots, v_t$, and the rest are random variables depending on the partition $\mathbf{A}$ and the oracle responses $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_t$.

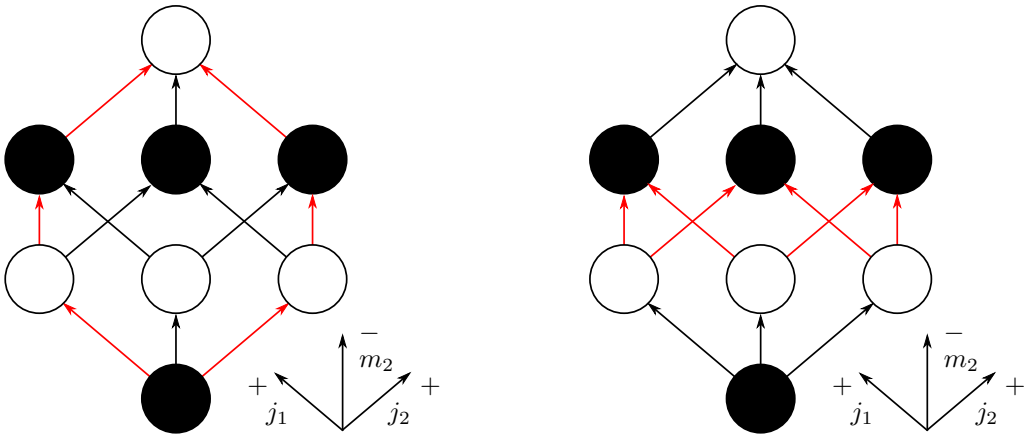

Figure 6 Examples of functions $\boldsymbol{h}_i\colon \{0,1\}^n \to \{0,1\}$ with orientations on the variables and violating edges. On the left-hand side, $\boldsymbol{h}_i(x) = \neg x_{j_1} \oplus x_{j_2} \oplus x_{m_2}$ with variables j_1 and j_2 (which ought to be monotone), and m_2 (which is always anti-monotone). On the right-hand side, $\boldsymbol{h}_i(x) = \neg x_{j_1} \oplus x_{j_2} \oplus x_{m_2}$ with variables j_1 and j_2 (which ought to be anti-monotone), and m_2 (which is always anti-monotone). We note that the violating edges form a cycle of length 6, so any unate function whose orientations on j_1 and j_2 are as indicated (both monotone on the left-hand side, and both anti-monotone on the right-hand side) must disagree on a $\frac{3}{8}$-fraction of the points.

1. $\mathcal{E}_T$ (Observe small trees): this is the event where the values of $v_1, \ldots, v_t$ form components $C_1, \ldots, C_\alpha$ which are all trees of size at most $\log n$.
2. $\mathcal{E}_F$ (Observe few non-empty responses): this is the event where the values of $v_1, \ldots, v_t$ have at most $\frac{n}{\log^4 n}$ non-$\emptyset$ responses. This event implies that the total number of vertices in the responses $v_1, \ldots, v_t$ is at most $\frac{n}{\log^4 n}$.
3. $\boldsymbol{\mathcal{E}}_{C,\text{yes}}$ and $\boldsymbol{\mathcal{E}}_{C,\text{no}}$ (Consistency condition of the components observed): these are the events where $\mathbf{A} \subseteq [n]$ partitions the components $C_1, \ldots, C_\alpha$ in a manner consistent with $\mathcal{G}_1$ in $\boldsymbol{\mathcal{E}}_{C,\text{yes}}$ or $\mathcal{G}_2$ in $\boldsymbol{\mathcal{E}}_{C,\text{no}}$, i.e., either every C_i is contained within $\mathbf{A}$ or $\overline{\mathbf{A}}$ (in the case of $\mathcal{G}_1$, or edges in every C_i cross the partition on vertices induced by $\mathbf{A}$ (in the case of $\mathcal{G}_2$). These events are random variables that depend only on $\mathbf{A}$. It will become clear that in order to observe the outcome $v_1, \ldots, v_t$ in $\mathcal{G}_1$, event $\boldsymbol{\mathcal{E}}_{C,\text{yes}}$ must be triggered, and in $\mathcal{G}_2$, event $\boldsymbol{\mathcal{E}}_{C,\text{no}}$ must be triggered. See Figure 7 for an illustration.
4. $\boldsymbol{\mathcal{E}}_O$ (Observe specific responses): this event is over the randomness in $\mathbf{A}$, as well as the randomness in the responses of the oracle $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_t$. The event is triggered when the responses of the oracle are exactly those dictated by $v_1, \ldots, v_t$; i.e., for all $i \in [t]$, $\boldsymbol{v}_i = v_i$.
5. $\boldsymbol{\mathcal{E}}_B$ (Balanced lone vertices condition): this event is over the randomness in $\mathbf{A}$, as well as the responses $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_t$. The event occurs when the queries L_i corresponding to lone vertices $\boldsymbol{v}_i$ have $|L_i \cap \mathbf{A}|$ and $|L_i \cap \overline{\mathbf{A}}|$ roughly equal, and roughly half of $\boldsymbol{v}_i$ fall in $\mathbf{A}$.

Having defined these events, the lower bound follows by the following three lemmas. The first lemma says that for any outcomes satisfying $\mathcal{E}_T$ and $\mathcal{E}_F$, the probability over $\mathbf{A}$ of being consistent in $\mathcal{G}_1$ cannot be much higher than in $\mathcal{G}_2$. The second lemma says that the outcomes satisfying the events described above do not help in distinguishing $\mathcal{G}_1$ and $\mathcal{G}_2$. The third lemma says that good outcomes occur with high probability over $\mathcal{G}_1$.

▶ **Lemma 15** (Consistency Lemma). *Consider a fixed $v_1, \ldots, v_t \in [n] \cup ([n] \times [n]) \cup \{\emptyset\}$ forming components $C_1, \ldots, C_\alpha$ where events $\mathcal{E}_T$ and $\mathcal{E}_F$ are satisfied. Then, we have:*

$$\Pr_{\substack{\mathbf{G}\sim\mathcal{G}_1 \\ \boldsymbol{v}_1,\ldots,\boldsymbol{v}_t}}[\boldsymbol{\mathcal{E}}_{C,yes}] \leq (1+o(1)) \Pr_{\substack{\mathbf{G}\sim\mathcal{G}_2 \\ \boldsymbol{v}_1,\ldots,\boldsymbol{v}_t}}[\boldsymbol{\mathcal{E}}_{C,no}].$$

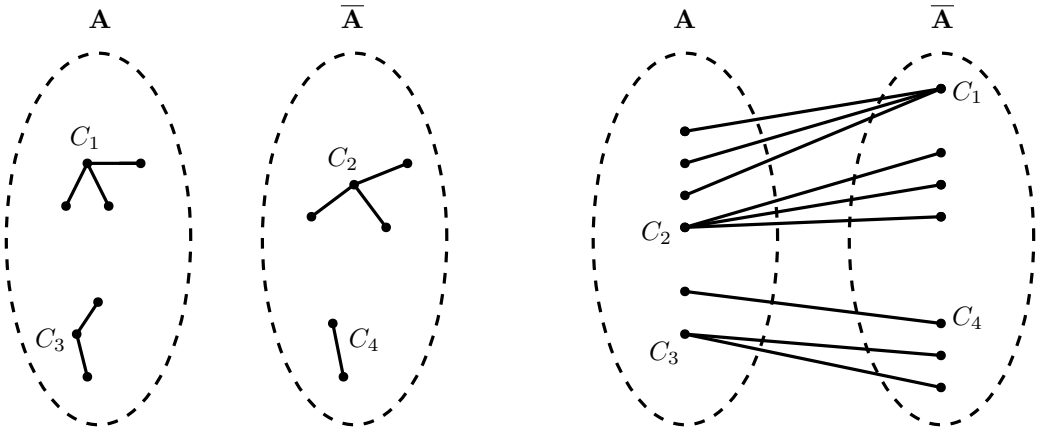

Figure 7 **A** consistently partition of the components C_1, C_2, C_3 and C_4 according to $\mathcal{G}_1$ (on the left) and $\mathcal{G}_2$ (on the right).

▶ **Lemma 16** (Good Outcomes Lemma). *Consider a fixed $v_1, \ldots, v_t \in [n] \cup ([n] \times [n]) \cup \{\emptyset\}$ forming components $C_1, \ldots, C_\alpha$ where events $\mathcal{E}_T$ and $\mathcal{E}_F$ are satisfied. Then, we have:*

$$\Pr_{\substack{\mathbf{G}\sim\mathcal{G}_1 \\ \boldsymbol{v}_1,\ldots,\boldsymbol{v}_t}} [\boldsymbol{\mathcal{E}}_O \wedge \boldsymbol{\mathcal{E}}_B \mid \boldsymbol{\mathcal{E}}_{C,yes}] \leq (1 + o(1)) \Pr_{\substack{\mathbf{G}\sim\mathcal{G}_2 \\ \boldsymbol{v}_1,\ldots,\boldsymbol{v}_t}} [\boldsymbol{\mathcal{E}}_O \mid \boldsymbol{\mathcal{E}}_{C,no}].$$

▶ **Lemma 17** (Bad Outcomes Lemma). *We have that:*

$$\Pr_{\substack{\mathbf{G}\sim\mathcal{G}_1 \\ \boldsymbol{v}_1,\ldots,\boldsymbol{v}_t}} [\neg\boldsymbol{\mathcal{E}}_T \vee \neg\boldsymbol{\mathcal{E}}_F \vee \neg\boldsymbol{\mathcal{E}}_B] = o(1).$$

Assuming the above three lemmas, we may prove Lemma 14.

References

1. Nir Ailon, Bernard Chazelle, Seshadhri Comandur, and Ding Liu. Estimating the distance to a monotone function. *Random Structures and Algorithms*, 31(3):371–383, 2007.
2. Roksana Baleshzar, Deeparnab Chakrabarty, Ramesh Krishnan S. Pallavoor, Sofya Raskhodnikova, and C. Seshadhri. A lower bound for nonadaptive, one-sided error testing of unateness of Boolean functions over the hypercube. *arXiv preprint*, 2017. `arXiv:1706.00053`.
3. Roksana Baleshzar, Deeparnab Chakrabarty, Ramesh Krishnan S. Pallavoor, Sofya Raskhodnikova, and C. Seshadhri. Optimal Unateness Testers for Real-Values Functions: Adaptivity Helps. In *Proceedings of the 44th International Colloquium on Automata, Languages and Programming (ICALP '2017)*, 2017.
4. Aleksandrs Belovs and Eric Blais. A polynomial lower bound for testing monotonicity. In *Proceedings of the 48th ACM Symposium on the Theory of Computing (STOC '2016)*, pages 1021–1032, 2016.
5. Piotr Berman, Meiram Murzabulatov, and Sofya Raskhodnikova. Tolerant Testers of Image Properties. In *Proceedings of the 43th International Colloquium on Automata, Languages and Programming (ICALP '2016)*, pages 90:1–90:14, 2016.
6. Piotr Berman, Sofya Raskhodnikova, and Grigory Yaroslavtsev. L_p-testing. In *Proceedings of the 46th ACM Symposium on the Theory of Computing (STOC '2014)*, 2014.
7. Eric Blais. Improved bounds for testing juntas. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, pages 317–330. Springer, 2008.

8 Eric Blais. Testing juntas nearly optimally. In *Proceedings of the 41st ACM Symposium on the Theory of Computing (STOC '2009)*, pages 151–158, 2009.

9 Eric Blais, Clément L Canonne, Talya Eden, Amit Levi, and Dana Ron. Tolerant junta testing and the connection to submodular optimization and function isomorphism. In *Proceedings of the 29th ACM-SIAM Symposium on Discrete Algorithms (SODA '2018)*, pages 2113–2132, 2018.

10 Harry Buhrman, David Garcıa-Soriano, Arie Matsliah, and Ronald de Wolf. The non-adaptive query complexity of testing k-parities. *Chicago Journal of Theoretical Computer Science*, 6:1–11, 2013.

11 Andrea Campagna, Alan Guo, and Ronitt Rubinfeld. Local reconstructors and tolerant testers for connectivity and diameter. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 411–424. Springer, 2013.

12 Clément L. Canonne, Dana Ron, and Rocco A. Servedio. Testing probability distributions using conditional samples. *SIAM Journal on Computing*, 44(3):540–616, 2015.

13 Deeparnab Chakrabarty and Seshadhri Comandur. An o(n) monotonicity tester for boolean functions over the hypercube. *SIAM Journal on Computing*, 45(2):461–472, 2016.

14 Deeparnab Chakrabarty and C. Seshadhri. A $\widetilde{O}(n)$ non-adaptive tester for unateness. *arXiv preprint*, 2016. `arXiv:1608.06980`.

15 Sourav Chakraborty, Eldar Fischer, David García-Soriano, and Arie Matsliah. Junto-symmetric functions, hypergraph isomorphism and crunching. In *Proceedings of the 27th Conference on Computational Complexity (CCC '2012)*, pages 148–158. IEEE, 2012.

16 Sourav Chakraborty, Eldar Fischer, Yonatan Goldhirsh, and Arie Matsliah. On the power of conditional samples in distribution testing. *SIAM Journal on Computing*, 45(4):1261–1296, 2016.

17 Xi Chen, Rocco A. Servedio, Li-Yang Tan, Erik Waingarten, and Jinyu Xie. Settling the query complexity of non-adaptive junta testing. In *Proceedings of the 32nd Conference on Computational Complexity (CCC '2017)*, 2017.

18 Xi Chen, Erik Waingarten, and Jinyu Xie. Beyond Talagrand functions: new lower bounds for testing monotonicity and unateness. In *Proceedings of the 49th ACM Symposium on the Theory of Computing (STOC '2017)*, 2017.

19 Xi Chen, Erik Waingarten, and Jinyu Xie. Boolean Unateness Testing with $\widetilde{O}(n^{3/4})$ Adaptive Queries. In *Proceedings of the 58th Annual IEEE Symposium on Foundations of Computer Science (FOCS '2017)*, 2017.

20 Hana Chockler and Dan Gutfreund. A lower bound for testing juntas. *Information Processing Letters*, pages 301–305, 2004.

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

22 Shahar Fattal and Dana Ron. Approximating the distance to monotonicity in high dimensions. *ACM Transactions on Algorithms*, 6(3):52, 2010.

23 Eldar Fischer and Lance Fortnow. Tolerant versus intolerant testing for Boolean properties. *Theory of Computing*, 2(9):173–183, 2006.

24 Eldar Fischer, Guy Kindler, Dana Ron, Shmuel Safra, and Alex Samorodnitsky. Testing juntas. *Journal of Computer and System Sciences*, 68(4):753–787, 2004. `doi:10.1016/j.jcss.2003.11.004`.

25 Eldar Fischer and Ilan Newman. Testing versus estimation of graph properties. *SIAM Journal on Computing*, 37(2):482–501, 2007.

26 Oded Goldreich. *Introduction to property testing*. Cambridge University Press, 2017.

27 Oded Goldreich, Shafi Goldwasser, Eric Lehman, Dana Ron, and Alex Samordinsky. Testing Monotonicity. *Combinatorica*, 20(3):301–337, 2000.

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

29 Oded Goldreich and Dana Ron. On sample-based testers. *ACM Transactions on Computation Theory*, 8(2), 2016.

30 Venkatesan Guruswami and Atri Rudra. Tolerant locally testable codes. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, pages 306–317. Springer, 2005.

31 Subhash Khot and Igor Shinkar. An $\widetilde{O}(n)$ queries adaptive tester for unateness. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, pages 37:1–37:7, 2016.

32 Swastik Kopparty and Shubhangi Saraf. Tolerant linearity testing and locally testable codes. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 601–614. Springer, 2009.

33 Sharon Marko and Dana Ron. Approximating the distance to properties in bounded-degree and general sparse graphs. *ACM Transactions on Algorithms*, 5(2):22, 2009.

34 Michal Parnas, Dana Ron, and Ronitt Rubinfeld. Tolerant property testing and distance approximation. *Journal of Computer and System Sciences*, 72(6):1012–1042, 2006.

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

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

37 Rocco A Servedio, Li-Yang Tan, and John Wright. Adaptivity helps for testing juntas. In *Proceedings of the 30th Conference on Computational Complexity (CCC '2015)*, pages 264–279, 2015.

38 Roei Tell. A Note on Tolerant Testing with One-Sided Error. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 23, page 32, 2016.