Distribution-Free Testing of Decision Lists with a Sublinear Number of Queries

Abstract

We give a distribution-free testing algorithm for decision lists with $\tilde{O}(n^{11/12}/\varepsilon^3)$ queries. This is the first sublinear algorithm for this problem, which shows that, unlike halfspaces, testing is strictly easier than learning for decision lists. Complementing the algorithm, we show that any distribution-free tester for decision lists must make $\tilde{\Omega}(\sqrt{n})$ queries, or take $\tilde{\Omega}(n)$ samples when the algorithm is sample-based.

1 Introduction

A Boolean function $f : \{0,1\}^n \to \{0,1\}$ is called a *decision list* (or 1-decision list) if there exists a list of pairs $(\alpha_1, \beta_1), \ldots, (\alpha_k, \beta_k)$ where each α_i is a literal and $\beta_i \in \{0,1\}$, such that $f(x)$ is set to be β_j of the smallest index j such that α_j is satisfied by x and is set to be a default value $\beta_{k+1} \in \{0,1\}$ if no literal is satisfied. Decision lists were first introduced by Rivest [Riv87], and have been one of the most well studied classes of Boolean functions in computational learning theory. In particular, the fundamental theorem of Statistical Learning [SSBD14] shows that the VC dimension of a class essentially captures the number of samples needed for its PAC learning [Val84], which gives a tight bound of $\Theta(n)$ for learning decision lists.

In this paper, we study the *distribution-free testing* of decision lists, where the goal of a tester is to determine whether an unknown function $f : \{0,1\}^n \to \{0,1\}$ is a decision list or ε-far from decision lists with respect to an unknown distribution $\mathcal{D}$ over $\{0,1\}^n$ (i.e., $\Pr_{x\sim\mathcal{D}}[f(x) \neq g(x)] \geq \varepsilon$ for any decision list g), given oracle (query) access to f and sampling access to $\mathcal{D}$. Inspired by the PAC learning model, distribution-free testing was first introduced by Goldreich, Goldwasser, and Ron [GGR98] and has been studied extensively [AC06, HK07, GS09, HK08a, HK08b, DR11, CX16, BFH21, CP22]. While testing is known to be no harder than proper learning [GGR98], much of the work is motivated by understanding whether concept classes well studied in learning theory can be tested more efficiently under the distribution-free testing model.

In [GS09], Glasner and Servedio obtained a lower bound of $\tilde{\Omega}(n^{1/5})$[1] on the query complexity[2] of distribution-free testing of conjunctions, decision lists and halfspaces. In [DR11], Dolev and Ron obtained a distribution-free testing algorithm for conjunctions with $\tilde{O}(\sqrt{n})$ queries. Later [CX16], Chen and Xie gave a tight bound of $\tilde{\Theta}(n^{1/3})$ for conjunctions; their $\tilde{\Omega}(n^{1/3})$ lower bound applies to decision lists and halfspaces as well. For sample-based distribution-free testing,[3] on the other hand, Blais, Ferreira Pinto Jr. and Harms [BFH21] obtained strong lower bounds for a number of concept classes based on a variant of VC dimension they proposed called the "lower VC" dimension. In

[1]For convenience we focus on the case when ε is a constant in the discussion of related work.

[2]For distribution-free testers, the query complexity refers to the number of queries made on f plus the number of samples drawn from $\mathcal{D}$. In many cases we simply refer to it as the number of queries made by the algorithm.

[3]A tester is sample-based if it can only draw samples $x_1, \ldots, x_q \sim \mathcal{D}$ and receive $f(x_1), \ldots, f(x_q)$.

particular, they showed that the distribution-free testing of halfspaces requires $\tilde{\Omega}(n)$ samples. Indeed even for general testers with queries, Chen and Patel [CP22] recently showed that $\tilde{\Omega}(n)$ queries are necessary, which implies that testing halfspaces is as hard as PAC learning.

To summarize, before this work, there remains wide gaps in our understanding of distribution-free testing of decision lists. In particular, it is not known whether sample-based distribution-free testing requires $\tilde{\Omega}(n)$ samples, and it is not known, when queries are allowed, whether there exists a distribution-free tester for decision lists with query complexity sublinear in n.

Our Contribution. We give the first sublinear distribution-free tester for decision lists:

Theorem 1.1. *There is a two-sided, adaptive distribution-free testing algorithm for decision lists that makes $\tilde{O}(n^{11/12}/\varepsilon^3)$ queries and has the same running time.*[4]

Theorem 1.1 is obtained by first giving an $\tilde{O}(n^{11/12}/\varepsilon^2)$-query algorithm for *monotone* decision lists in Section 4 (where a decision list is said to be monotone [GLR01] if all literals α_i in the list are positive), and combining it with a reduction to testing general decision lists in Section 5.

On the lower bound side, we show that any distribution-free testing algorithm for decision lists must make $\tilde{\Omega}(\sqrt{n})$ queries, and must draw $\tilde{\Omega}(n)$ samples when the algorithm is sample-based.

Theorem 1.2. *Any two-sided, adaptive distribution-free testing algorithm for decision lists must make $\tilde{\Omega}(\sqrt{n})$ queries when ε is a sufficiently small constant.*

Theorem 1.3. *Any two-sided, sample-based distribution-free testing algorithm for decision lists must draw $\tilde{\Omega}(n)$ samples when ε is a sufficiently small constant.*

As a warm-up for Theorem 1.1, we give an optimal distribution-free testing algorithm for *total orderings*, which highlights, in a much simplified setting, some of the most crucial ideas behind the main algorithm for monotone decision lists. The input consists of 1) oracle access to a tournament graph[5] G over $[n]$ (i.e., one can pick any $u \neq v \in [n]$ and query whether (u, v) or (v, u) is in G; and 2) sampling access to a probability distribution $\mathcal{D}$ over edges of G. The goal is then to determine whether G is acyclic, or ε-far from acyclic with respect to $\mathcal{D}$ (i.e., any feedback edge set[6] of G has probability mass at least ε in $\mathcal{D}$).

Theorem 1.4. *There is a two-sided, adaptive distribution-free testing algorithm for total orderings that makes $\tilde{O}(\sqrt{n}/\varepsilon)$ queries. On the other hand, any such algorithm for total orderings must make $\Omega(\sqrt{n})$ queries when ε is a sufficiently small constant.*

1.1 Technical Overview

We begin by describing our lower bound for total orderings (cf. Figure 1). To prove the statement, we use Yao's lemma and construct two distributions $\mathcal{D}_{YES}$ and $\mathcal{D}_{NO}$. Note that a draw from $\mathcal{D}_{YES}$ and $\mathcal{D}_{NO}$ consists of both a function $f : \{0,1\}^n \to \{0,1\}$ and a distribution $\mathcal{D}$ over $\{0,1\}^n$. Most of the relevant ideas come up in just a one-sided bound[7], so we focus on this case and only describe the $\mathcal{D}_{NO}$ distribution. To draw a function and distribution from $\mathcal{D}_{NO}$, we first choose a random permutation π. We then create a set of 4-cycles i.e. $\pi(4i+1) <_\sigma \pi(4i+2) <_\sigma \pi(4i+3) <_\sigma$

[4] For the running time we assume that standard bitwise operations such as bitwise AND, OR and XOR over n-bit strings each cost one step.

[5] These are directed graphs obtained by assigning a direction to each edge in an undirected complete graph.

[6] A feedback edge set of a directed graph is a set of edges such that the graph becomes acyclic after its removal.

[7] Recall that a tester for decision-lists is one-sided if it never rejects when f is a decision-list. Such testers reject when they find a violation, so to prove a one-sided lower bounds it suffices to give a distribution $\mathcal{D}_{NO}$ where any deterministic algorithm fails to find a violation.

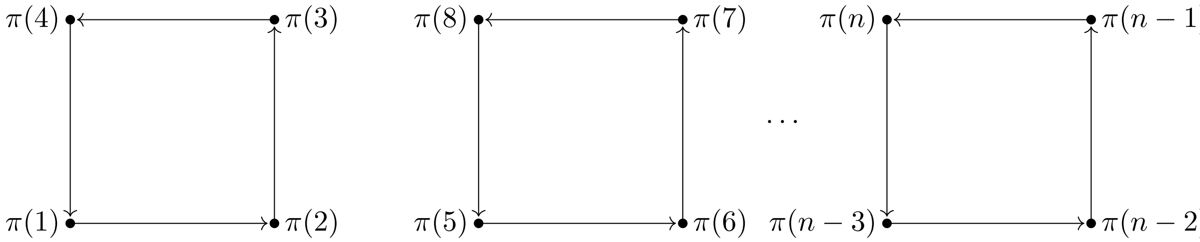

Figure 1: Lower Bound Construction for Total Orderings. An edge from x to y indicates that $x <_\sigma y$. Note that we only show edges in the support of $\mathcal{D}$ in the figure.

$\pi(4i+4) <_\sigma \pi(4i+1)$ for $i = 0, 1, \ldots, \frac{n}{4} - 1$. Note that we are using the notation $x <_\sigma y$ to denote the fact that the edge (x, y) appears in the tournament graph. The ordering across cycles is made to be consistent with a total ordering. Namely, $\pi(x) <_\sigma \pi(y)$ if $\lceil \frac{y}{4} \rceil \geq \lceil \frac{x}{4} \rceil$. Finally the distribution $\mathcal{D}$ is uniform over the edges of the four cycles i.e. $\pi(1)\pi(2), \pi(2)\pi(3), \pi(3)\pi(4), \pi(4)\pi(1), \pi(5)\pi(6), \ldots$.

Clearly, in order to make $<_\sigma$ into a total ordering, we must change at least one edge in each cycle, so each function drawn from $\mathcal{D}_{NO}$ is $\frac{1}{4}$-far from total orderings. On the other hand, in order to find a violation, we must find three vertices from a single cycle, which takes $\Omega(\sqrt{n})$ queries by a birthday paradox argument. Our lower bound for decision lists follows a similar high-level scheme, but with extra care to handle the case where the tester queries a string with large support.

We now use this lower bound as motivation for designing our tester for total orderings. In particular, consider a tester that correctly rejects on the lower bound instance. It must draw $\Omega(\sqrt{n})$ samples. After doing so, it is likely to have drawn two samples from the same cycle. That said, the algorithm does not know which pair of edges this is and querying all pairs would require $\Omega(n)$ queries.[8]

To circumvent this issue, we create a "sketch" of the ordering. At a high level, the sketch buckets the vertices into $\sqrt{n}$ blocks $B_1, \ldots, B_{\sqrt{n}}$ with the guarantee that if σ were a total ordering then any vertex in B_i ought to be less than any vertex in B_j if $j > i$. We also have that the probability of drawing an edge incident to some vertex in a block B_i is at most $\tilde{O}(n^{-1/2})$ and that we can quickly determine which block a vertex belongs to.

With this sketch in hand, we can build a $\tilde{O}(\sqrt{n})$ tester. To do so, we consider cycles in the tournament graph corresponding to the ordering. We can then split these into two types: Those involving edges across blocks and those completely contained within a block. To handle cycles spanning multiple blocks, we simply check that the sketch looks to be consistent with the ordering. On the other hand, if there were many cycles completely contained in a block, we show that $O(\sqrt{n})$ samples suffices to sample one. Since each block is unlikely to be incident to an edge, no block will have more than $\tilde{O}(1)$ sampled edges with high probability. We can then query every pair in a block to find the violation.

Unfortunately, several aspects of this approach break when attempting to adapt the result to monotone decision lists. Note that a monotone decision list f naturally induces an ordering over strings $x \in \{0,1\}^n$ based the rule in x that fires in f. That said, in this setting, one can only compare elements x, y with $f(x) \neq f(y)$. To accommodate this, we bucket the elements $[n]$ into sets $B_1, \ldots, B_k$ that now have alternating values i.e. for all $i \in B_j$ we have that $f(e_i) = i \mod 2$, where $e_i \in \{0,1\}^n$ is ith standard basis vector. That said no longer have the guarantee that all elements in B_i are smaller than those in B_{i+1}, but they are guaranteed to be smaller than B_j for

[8]Technically, for the lower bound presented we can hope to draw two edges from a cycle that share a vertex, but one can modify the construction so that these types of pairs of edges won't yield a violation.

$j > i + 1$.

The primary challenge when testing monotone decision lists is determining what constitutes a violation. In the case of total orderings, each comparison provided a concrete bit indicating that one element was larger than another under σ, and a violation was clearly defined as a cycle. However, in the case of a monotone decision list, querying a string x with, say, $f(x) = 0$, only tells you that some zero rule in x is greater than all the one rules in x.

To address this, we design a procedure that determines the value of the maximum element $i \in \text{supp}(x)$. However, this procedure is effective only for blocks B_i that contain at most n^δ indices. Once we identify the maximum elements, violations naturally correspond to cycles in an associated hypergraph.

Nevertheless, this procedure is insufficient for testing, as many of the blocks in our sketch may have $\Omega(n)$ indices. For instance, if f is a conjunction, there are only 2 blocks and at least one must be large. To handle such large blocks, we prove that if f is a decision list, then most elements of B_i are smaller than those in B_{i+1}. If we could check that this property holds for a general f, which may not be a decision list, then we are in a similar setting to that of the total ordering case and can easily control violations involving elements from any fixed set of $n^{1-\delta}$ blocks. Verification of this property turns out to be somewhat tricky, but we demonstrate that it can be achieved with an argument similar in spirit to Dolev and Ron's conjunction tester, but crucially modified to use an asymmetric version of the birthday paradox.

Finally to extend our result to general decision lists, we note that given an arbitrary decision list f, if we know the default string r, then $f(x \oplus r)$ is now a monotone decision list. While we are unable to find r, we show that it suffices to find a string whose firing rule has sufficiently low priority in the decision list. We can then draw many samples and guess each string as the candidate low priority element.

2 Preliminaries

Notation. Given a positive integer n, we write $[n]$ to denote $\{1, \dots, n\}$. Given two integers $a \le b$, we write $[a : b]$ to denote the set of integers $\{a, \dots, b\}$ between a and b. Given a probability distribution $\mathcal{D}$ over a finite set S, we write $\mathcal{D}(p)$ to denote the probability mass of $p \in S$ in $\mathcal{D}$, and write $\mathcal{D}(P)$ for a given subset $P \subseteq S$ to denote $\sum_{p \in P} \mathcal{D}(p)$. Throughout the paper, drawing a set T of m samples from $\mathcal{D}$ always means to draw m independent samples from $\mathcal{D}$ (with replacements) and take T to be the set they form (so in general $|T|$ could be smaller than m).

Given two strings x and $y \in \{0,1\}^n$, we write $x \vee y$ to denote the bitwise OR of x and y, i.e., $x \vee y \in \{0,1\}^n$ with $(x \vee y)_i = x_i \vee y_i$ for all $i \in [n]$, and $x \oplus y$ to denote their bitwise XOR, with $(x \oplus y)_i = x_i \oplus y_i$ for all $i \in [n]$. Given $i \in [n]$ we write e_i to denote the string in $\{0,1\}^n$ such that $(e_i)_i = 1$ and $(e_i)_j = 0$ for all $j \neq i$. Given a probability distribution $\mathcal{D}$ over $\{0,1\}^n$ and $r \in \{0,1\}^n$, we write $\mathcal{D} \oplus r$ to denote the distribution over $\{0,1\}^n$ with $\mathcal{D} \oplus r(x) = \mathcal{D}(x \oplus r)$.

Given $f : \{0,1\}^n \to \{0,1\}$, $x \in \{0,1\}^n$ is a 1-string of f if $f(x) = 1$ and a 0-string if $f(x) = 0$.

Monotone Decision Lists. A function $f : \{0,1\}^n \to \{0,1\}$ is said to be a monotone decision list if it can be represented by a pair (π, ν)[9], where π is a permutation over $[n]$ and $\nu \in \{0,1\}^{n+1}$, such that $f(x) = \nu_j$ if j is the smallest integer in $[n]$ such that $x_{\pi(j)=1}$, and $f(x) = \nu_{n+1}$ when $x = 0^n$. Variable $i \in [n]$ is said to be a b-rule variable if $\nu_j = b$ for $j = \pi^{-1}(i)$, where $b \in \{0,1\}$. We write MonoDL to denote the class of monotone decision lists.

Given π and $x \in \{0,1\}^n$, we write $\min_\pi(x)$ to denote the smallest $j \in [n]$ such that $x_{\pi(j)} = 1$ and it is set to $n+1$ when $x = 0^n$. Let f be an *arbitrary* Boolean function and x, y be two strings

[9] Note though that the representation is not unique in general.

with $f(x) \neq f(y)$. We write $x \succ_f y$ (or $y \prec_f x$) if $f(x \vee y) = f(x)$. Note that when f is a monotone decision list, we have $x \succ_f y$ if and only if $\min_\pi(x) < \min_\pi y$.

Decision Lists. A function $f : \{0,1\}^n \to \{0,1\}$ is said to be a decision list if $g := f(x \oplus r)$ is a monotone decision list for some $r \in \{0,1\}^n$. Equivalently, f is a decision list if it can be represented by a triple (π, μ, ν), where $\pi : [n] \to [n]$ is a permutation over $[n]$, $\mu \in \{0,1\}^n$, and $\nu \in \{0,1\}^{n+1}$, such that $f(x) = \nu_j$ if j is the smallest integer in $[n]$ such that $x_{\pi(j)} = \mu_{\pi(j)}$, and $f(x) = \nu_{n+1}$ if no such j exists. Similarly, we say variable $i \in [n]$ is a b-rule variable if $\nu_j = b$ for $j = \pi^{-1}(i)$. Given π, μ and $x \in \{0,1\}^n$, we let $\min_{\pi,\mu}(x)$ denote the smallest j with $x_{\pi(j)} = \mu_{\pi(j)}$, and it is set to $n+1$ if no such j exists.

Distribution-free Testing. We review the model of distribution-free property testing. Let $f, g : \{0,1\}^n \to \{0,1\}$ denote two Boolean functions and $\mathcal{D}$ denote a distribution over $\{0,1\}^n$.

We define the distance between f and g with respect to $\mathcal{D}$ as

$$\mathrm{dist}_{\mathcal{D}}(f,g) = \Pr_{x \in \mathcal{D}}\big[f(x) \neq g(x)\big].$$

Given a class $\mathfrak{C}$ of Boolean functions (such as the class of (monotone) decision lists), we define

$$\mathrm{dist}_{\mathcal{D}}(f,\mathfrak{C}) = \min_{g \in \mathfrak{C}} \Big(\mathrm{dist}_{\mathcal{D}}(f,g) \Big)$$

as the distance between f and $\mathfrak{C}$ with respect to $\mathcal{D}$. We also say that f is ϵ-far from $\mathfrak{C}$ with respect to $\mathcal{D}$ for some $\epsilon \geq 0$ if $\mathrm{dist}_{\mathcal{D}}(f,\mathfrak{C}) \geq \epsilon$. Now we define distribution-free testing algorithms.

Let $\mathfrak{C}$ be a class of Boolean functions over $\{0,1\}^n$. A distribution-free testing algorithm ALG for $\mathfrak{C}$ has access to a pair $(f,\mathcal{D})$, where f is an unknown Boolean function $f : \{0,1\}^n \to \{0,1\}$ and $\mathcal{D}$ is an unknown probability distribution over $\{0,1\}^n$, via

1. a black-box oracle that returns the value $f(x)$ when $x \in \{0,1\}^n$ is queried; and
2. a sampling oracle that returns a sample $x \sim \mathcal{D}$ drawn independently each time.

The algorithm ALG takes $(f,\mathcal{D},\varepsilon)$ as input, where $\varepsilon > 0$ is a distance parameter, and satisfies:

1. If $f \in \mathfrak{C}$, then ALG accepts with probability at least $2/3$; and
2. If f is ε-far from $\mathfrak{C}$ with respect to $\mathcal{D}$, then ALG rejects with probability at least $2/3$.

We say an algorithm is sample-based if it can only receive a sequence of samples $z_1, \ldots, z_q \sim \mathcal{D}$ together with $f(z_1), \ldots, f(z_q)$.

2.1 Birthday Paradox Lemmas

As highlighted earlier in the sketch of our algorithms, birthday paradox arguments play an important role in the analysis. We include the proof of two birthday paradox lemmas in Section 8, one for bipartite graphs and one for hypergraphs. The bipartite graph lemma (Lemma 2.1 below) has been previously incorporated as a crucial component of the analysis in [DR11] for the distribution-free testing of monomials, though without an explicit statement. In Section 8, we provide an alternative proof of this lemma, drawing upon the classical birthday paradox from probability theory, and then extend the proof to work for hypergraphs (Lemma 2.2).

Lemma 2.1. *Let $G = (U,V,E)$ be a bipartite graph, with probability distributions μ on $U \cup \{\#\}$ and ν on $V \cup \{\#\}$. Assume that any vertex cover $C = C_1 \sqcup C_2$ of G, where $C_1 \subset U$ and $C_2 \subset V$, has $\mu(C_1) + \nu(C_2) \geq \varepsilon$. Let S be a set of m independent samples from μ and S' be a set of m' independent samples from ν, with m and m' satisfying $m \cdot m' \geq 100|U|/\varepsilon^2$ and $m, m' \geq 100/\varepsilon$. With probability at least 0.99, there exist $x \in S$ and $y \in S'$ such that (x,y) is an edge in G.*

Lemma 2.2. *Let $G = (V, E)$ be a k-uniform hypergraph and let μ be a probability distribution over $V \cup \{\#\}$ such that any vertex cover C of G has $\mu(C) \geq \varepsilon$. Let S be a set of m sample from μ with*

$$m \geq \frac{10k^2|V|^{(k-1)/k}}{\varepsilon}.$$

Then S contains an edge in G with probability at least 0.99.

3 Warm-up: Testing Total Orderings

In this section, we present a distribution-free testing algorithm for *total orderings* as a warm-up to demonstrate some of the ideas (such as the use of *sketches* and the classification of cycles into *long cycles* and *local cycles*) that will play important roles in our algorithm for monotone decision lists.

In the problem of testing total orderings, we are given query access to a comparison function $<_\sigma$ over $[n]$ and sampling access to a distribution $\mathcal{D}$ over $\binom{[n]}{2}$. For any $u \neq v \in [n]$, the tester can query $<_\sigma$ on $\{u, v\}$ to reveal if $u <_\sigma v$ or $v <_\sigma u$. Given $<_\sigma, \mathcal{D}$ and ε, the goal of the tester is to

1. accept with probability at least 2/3 if the comparison function $<_\sigma$ is a total ordering; and
2. reject with probability at least 2/3 if $<_\sigma$ is ε-far from total orderings with respect to $\mathcal{D}$, i.e.,

$$\Pr_{\{u,v\}\sim\mathcal{D}}\Big[\big[u <_\sigma v \text{ and } u >_\tau v\big] \text{ or } \big[u >_\sigma v \text{ and } u <_\tau v\big]\Big] \geq \varepsilon, \quad \text{for any total ordering } <_\tau.$$

We will prove the following theorem for the distribution-free testing of total orderings:

Theorem 3.1. *There is a distribution-free tester for total orderings with $\tilde{O}(\sqrt{n}/\varepsilon)$ queries.*

We remark that our tester is optimal up to logarithmic factors. Indeed one can easily modify the lower bound from Section 6 to show that any tester must make $\tilde{\Omega}(\sqrt{n})$ many queries when ε is a sufficiently small constant.

3.1 Sketches

The backbone of our tester for total orderings (as well as monotone decision lists in Section 4) are *sketches*, which, roughly speaking, can help us compare elements that are far in the ordering.

Definition 3.2 (Sketch)**.** A sketch $\mathcal{S} = (s^{(1)}, \ldots, s^{(k)})$ is a tuple of distinct elements from $[n]$ for some $k \geq 1$. We say $\mathcal{S}$ is *consistent* with a comparison function $<_\sigma$ if $s^{(i)} <_\sigma s^{(i+1)}$ for all $i \in [k-1]$.

Note that when $<_\sigma$ is a total ordering, one can infer from a consistent sketch $\mathcal{S}$ that $s^{(i)} <_\sigma s^{(j)}$ for all $i < j$. This, however, does not hold for general comparison functions.

The procedure Sketch described in Algorithm 1 efficiently builds a sketch by simply sampling and sorting elements drawn from $\mathcal{D}^*$, where $\mathcal{D}^*$ is a distribution over $[n]$ defined using $\mathcal{D}$ as follows

$$\mathcal{D}^*(i) := \frac{1}{2} \cdot \sum_{j \neq i} \mathcal{D}(\{i, j\}), \quad \text{for each } i \in [n].$$

Note that sampling access to $\mathcal{D}^*$ can be simulated using sampling access to $\mathcal{D}$, query by query, by first sampling from $\mathcal{D}$ and returning one of the two elements uniformly at random.

We summarize performance guarantees of Sketch in the following lemma:

Algorithm 1 SKETCH($<_\sigma, \mathcal{D}, \varepsilon$)

Input: Oracle access to $<_\sigma$, sampling access to $\mathcal{D}$ and $\varepsilon > 0$

1: Draw $O(\sqrt{n}/\varepsilon)$ samples $\mathcal{D}^*$ and let S be the set of elements sampled

2: Sort elements in S into $s^{(1)}, \ldots, s^{(k)}$ by running MergeSort with $<_\sigma$, where $k = |S| \geq 1$

3: Query $\{s^{(i)}, s^{(i+1)}\}$ and **reject** if $s^{(i)} >_\sigma s^{(i+1)}$ for any $i \in [k-1]$

4: **return** $\mathcal{S} := (s^{(1)}, \ldots, s^{(k)})$

Lemma 3.3. SKETCH *makes* $\tilde{O}(\sqrt{n}/\varepsilon)$ *queries. It rejects or returns a sketch consistent with* $<_\sigma$.

Suppose that $<_\sigma$ *is a total ordering. Then* SKETCH *always returns a sketch* $\mathcal{S} = (s^{(1)}, \ldots, s^{(k)})$ *that is consistent with* $<_\sigma$. *Moreover, with probability at least* $1 - o_n(1)$, $\mathcal{S}$ *satisfies for all* $i \in [0:k]$,

$$\Pr_{u \sim \mathcal{D}^*}\left[s^{(i)} <_\sigma u <_\sigma s^{(i+1)}\right] < \frac{100\varepsilon \log n}{\sqrt{n}}, \tag{3.1}$$

where the event above is $u <_\sigma s^{(1)}$ *when* $i = 0$ *and is* $s^{(k)} <_\sigma u$ *when* $i = k$.

Proof. The only nontrivial part of the lemma is to show that the event described at the end occurs with probability at least $1 - o_n(1)$. Without loss of generality, take $<_\sigma$ to be the total ordering with $1 <_\sigma 2 <_\sigma \cdots <_\sigma n$. For each i, let j_i denote the smallest integer with $\mathcal{D}^*([i, j_i]) \geq 100\varepsilon \log n/\sqrt{n}$. Note that

$$\Pr\Big[\text{SKETCH does not sample any element in } [i, j_i]\Big] \leq \left(1 - \frac{100\varepsilon \log n}{\sqrt{n}}\right)^{\sqrt{n}/\varepsilon} \leq n^{-100}.$$

So by a union bound, SKETCH samples a point from each interval $[i, j_i]$ with high probability. The lemma follows because, if there exists an i such that

$$\Pr_{u \sim \mathcal{D}^*}\left[s^{(i)} <_\sigma u <_\sigma s^{(i+1)}\right] \geq \frac{100\varepsilon \log n}{\sqrt{n}}$$

then it must be the case that SKETCH did not sample any point in some interval $[i, j_i]$. □

Given a sketch $\mathcal{S}$ that is consistent with a total ordering $<_\sigma$, FINDBLOCK($<_\sigma, \mathcal{S}, u$) (described in Algorithm 2) returns the unique $i \in [0:k]$ such that

1. $i = 0$ if $u <_\sigma s^{(1)}$;
2. $i \in [k-1]$ if either $u = s^{(i)}$ or $s^{(i)} <_\sigma u <_\sigma s^{(i+1)}$; and
3. $i = k$ if either $u = s^{(k)}$ or $s^{(k)} <_\sigma u$.

Indeed, FINDBLOCK returns such an i for u even when $<_\sigma$ is an arbitrary comparison function.

We summarize its performance guarantees below:

Lemma 3.4. FINDBLOCK($<_\sigma, \mathcal{S}, u$) *is deterministic and makes* $O(\log n)$ *queries. It always returns an* $i \in [0:k]$ *that satisfies the conditions above for* u.

Given any $<_\sigma$ and a sketch $\mathcal{S}$ consistent with $<_\sigma$, FINDBLOCK (which is deterministic) uses $\mathcal{S}$ to induce a partition of $[n]$ into *blocks*. We say $u \in [n]$ lies in the ℓ-th block (with respect to $\mathcal{S}$) for some $\ell \in [0:k]$ if $\ell = \text{FINDBLOCK}(<_\sigma, \mathcal{S}, u)$.

Algorithm 2 FindBlock($<_\sigma, \mathcal{S}, u$)

Input: Oracle access to $<_\sigma$, a sketch $\mathcal{S} = (s^{(1)}, \ldots, s^{(k)})$ consistent with $<_\sigma$ and $u \in [n]$

return i if $u = s^{(i)}$ for some $i \in [k]$
return 0 if $u <_\sigma s^{(1)}$; **return** k if $s^{(k)} <_\sigma u$
Set upper $\leftarrow k$ and lower $\leftarrow 1$
while upper $-$ lower > 1 **do**
    Set mid $\leftarrow \lfloor(\text{upper} + \text{lower})/2\rfloor$
    If $s^{(\text{mid})} >_\sigma u$, set upper $\leftarrow$ mid; otherwise, lower $\leftarrow$ mid
end while
return mid

3.2 The order graph and classification of cycles

We now move to discuss how we will reject comparison functions that are far from total orderings. Towards this goal, we define the *order graph* and introduce some notation:

Definition 3.5 (Order graph)**.** Given a comparison function $<_\sigma$, the order graph G_σ is an orientation of the complete graph K_n, where edge (u, v) is oriented towards v if $u <_\sigma v$. The distribution $\mathcal{D}$ naturally induces a distribution over edges of G_σ: the probability mass of an edge (u, v) in G_σ is given by $\mathcal{D}(\{u, v\})$. For convenience we will still use $\mathcal{D}$ to denote the distribution over edges of G_σ and write $\mathcal{D}(R)$ to denote the total probability of a set of edges R in G_σ.

It's easy to see that if the order graph is acyclic if and only if $<_\sigma$ is a total ordering. Moreover, we can connect distance between $<_\sigma$ and total orderings with feedback edge sets of G_σ:

Lemma 3.6. *If $<_\sigma$ is ε-far from total orderings with respect to $\mathcal{D}$, then any set R of edges of G_σ such that G_σ is acyclic after removing R (i.e., R is a feedback edge set) must satisfy $\mathcal{D}(R) \geq \varepsilon$.*

Proof. Let R be such a set. As G_σ after removing R is acyclic, there is a total ordering of $[n]$ that is consistent with all edges of G_σ except those in R. As a result, the distance between $<_\sigma$ and total orderings under $\mathcal{D}$ is at most $\mathcal{D}(R)$, from which we have that $\mathcal{D}(R) \geq \varepsilon$. □

Consider $(<_\sigma, \mathcal{D})$ such that $<_\sigma$ is ε-far from total orderings with respect to $\mathcal{D}$. We use a sketch $\mathcal{S}$ to classify cycles of $\mathcal{D}$ into two types: *long* cycles and *local* cycles.

Definition 3.7 (Long and local cycles)**.** Given a sketch $\mathcal{S} = (s^{(1)}, \ldots, s^{(k)})$, we say a directed edge (u, v) in G_σ (which means that $u <_\sigma v$) is a *long* edge (with respect to $\mathcal{S}$) if

$$\textsc{FindBlock}(<_\sigma, \mathcal{S}, u) > \textsc{FindBlock}(<_\sigma, \mathcal{S}, v).$$

A cycle in G_σ is said to be a *long* cycle if it contains at least one long edge. A cycle in G_σ is said to be a *local* cycle if it does not contain any long edges.

Given that every cycle is either long or local, we have the following corollary of Lemma 3.6:

Corollary 3.8. *Suppose $<_\sigma$ is ε-far from total orderings with respect to $\mathcal{D}$, and $\mathcal{S}$ is a sketch that is consistent with $<_\sigma$. Then either any feedback edge set R for long cycles of G_σ has $\mathcal{D}(R) \geq \varepsilon/2$, or any feedback edge set R for local cycles of G_σ has $\mathcal{D}(R) \geq \varepsilon/2$.*

TestLongCycles (see Algorithm 3) is the procedure that helps reject $(<_\sigma, \mathcal{D})$ when $\mathcal{D}(R) \geq \varepsilon/2$ for any feedback edge set R of long cycles of G_σ. It simply draws edges from $\mathcal{D}$ and rejects when a long edge is found. Given that a total ordering has no long edges, TestLongCycles is trivially one-sided. Its performance guarantees are stated in the following lemma:

Algorithm 3 TestLongCycles($<_\sigma, \mathcal{D}, \varepsilon, \mathcal{S}$)

Input: Oracle access to $<_\sigma$, sampling access to $\mathcal{D}$, $\varepsilon > 0$ and a sketch $\mathcal{S}$ consistent with $<_\sigma$

1: Draw $100/\varepsilon$ samples from $\mathcal{D}$

2: For each $\{u, v\}$ sampled with $u <_\sigma v$, **reject** if FindBlock($<_\sigma, \mathcal{S}, u$) $>$ FindBlock($<_\sigma, \mathcal{S}, v$)

3: **accept**

Lemma 3.9. *TestLongCycles($<_\sigma, \mathcal{D}, \varepsilon, \mathcal{S}$) makes $O(\log n/\varepsilon)$ queries.*

When $<_\sigma$ is a total ordering, TestLongCycles always accepts.

Suppose that any feedback edge set R for long cycles in G_σ satisfies $\mathcal{D}(R) \geq \varepsilon/2$. Then TestLongCycles rejects with probability at least 0.99.

Proof. Note that the set of long edges forms a feedback edge set for long cycles. It follows that we sample a long edge on line 1 with probability at least $1 - (1 - \varepsilon/2)^{100/\varepsilon} \geq 0.99$. □

Next we consider the case when any feedback edge set for local cycles of G_σ has mass at least $\varepsilon/2$. It follows from the definition that a cycle C is local if and only if all of its vertices lie in the same block, i.e., FindBlock($<_\sigma, \mathcal{S}, u$) is the same for all $u \in C$. The following lemma motivates the procedure TestLocalCycles for this case. To state the lemma, we let H denote the following undirected bipartite graph: the left side of H consists of edges of G_σ; the right side of H consists of vertices $[n]$ of G_σ; (u, v) and w has an edge iff $v <_\sigma w <_\sigma u$ and

$$\textsc{FindBlock}(<_\sigma, \mathcal{S}, u) = \textsc{FindBlock}(<_\sigma, \mathcal{S}, v) = \textsc{FindBlock}(<_\sigma, \mathcal{S}, w).$$

Combining with $u <_\sigma v$ as (u, v) is an edge in G_σ, an edge between (u, v) and w in H implies that u, v, w form a directed triangle, a violation to $<_\sigma$ being a total ordering.

We are now ready to state the lemma:

Lemma 3.10. *Suppose that any feedback edge set R for local cycles in G_σ has $\mathcal{D}(R) \geq \varepsilon/2$. Then any vertex cover $C = C_1 \sqcup C_2$ of H must have $\mathcal{D}(C_1) + \mathcal{D}^*(C_2) \geq \varepsilon/2$.*

Proof. First we show that for any local cycle $C = c_0 \dots c_{k-1}$ in G_σ, there must exist i and j such that $c_i c_{(i+1) \bmod k} c_j$ forms a directed triangle. Assume for a contradiction that this is not the case. We start by noting that we must have that $c_0 <_\sigma c_j$ for all j. Indeed, assume that $c_0 <_\sigma c_i$. If $c_{i+1} <_\sigma c_0$ then $c_0 c_i c_{(i+1) \bmod k}$ forms a directed triangle. Since the base case holds ($c_0 <_\sigma c_1$), we conclude that $c_0 <_\sigma c_j$ for all j. But now we have reached a contradiction since $c_{k-1} <_\sigma c_0$.

Therefore, for any local cycle $C = c_0 \dots c_{k-1}$ of G_σ, there exist i and j such that there is an edge bewteen $(c_i, c_{i+1 \bmod k})$ and c_j in H. We now claim that if $C = C_1 \sqcup C_2$ is a vertex cover of H, then there is a feedback edge set R for local cycles in G_σ with $\mathcal{D}(R) \leq \mathcal{D}(C_1) + \mathcal{D}^*(C_2)$, from which the lemma follows. To see this is the case, we set R to be the following set of edges in G_σ: all edges in C_1 and all edges that are incident to a vertex in C_2. It is easy to verify that R is a feedback edge set for local cycles of G_σ. This finishes the proof of the lemma. □

Based on Lemma 3.10, TestLocalCycles (Algorithm 4) mimics the bipartite birthday paradox Lemma 2.1 by drawing $\sqrt{n}/\varepsilon$ samples S from $\mathcal{D}$ and $\sqrt{n}/\varepsilon$ samples T from $\mathcal{D}^*$. Then for any edge (u, v) in S and any vertex w in T with all u, v, w lying in the same block, we query $\{u, w\}$ and $\{v, w\}$ to see if they form a directed triangle. Naively, however, this could lead to $\Omega(n)$ queries (e.g., consider the worst case when all elements lie in the same block). However, by Lemma 3.3, this is unlikely to occur when $<_\sigma$ is truly a total ordering so TestLocalCycles rejects when too many samples lie in the same block. This is where the algorithm makes two-sided errors though.

We state performance guarantees of TestLocalCycles in the following lemma:

Algorithm 4 TestLocalCycles($<_\sigma, \mathcal{D}, \varepsilon, \mathcal{S}$)

Input: Oracle access to $<_\sigma$, sampling access to $\mathcal{D}$, $\varepsilon > 0$ and a sketch $\mathcal{S}$ consistent with $<_\sigma$

1: Draw $O(\sqrt{n}/\varepsilon)$ edges S from $\mathcal{D}$ and draw $O(\sqrt{n}/\varepsilon)$ elements T from $\mathcal{D}^*$
2: For every element u in T or an edge of S, run FindBlock($<_\sigma, \mathcal{S}, u$).
3: **reject** if any block has more than $1000 \log n$ elements from T
4: **for** every $(u, v) \in S$ and $w \in T$ such that FindBlock puts them in the same block **do**
5: Query $\{u, w\}$ and $\{v, w\}$ and **reject** if u, v, w form a directed triangle in G_σ
6: **end for**
7: **accept**

Algorithm 5 TestTotalOrdering($<_\sigma, \mathcal{D}, \varepsilon$)

Input: Oracle access $<_\sigma$, sampling access to $\mathcal{D}$ and $\varepsilon > 0$

Run Sketch($<_\sigma, \mathcal{D}, \varepsilon$) and **reject** if it rejects; otherwise let $\mathcal{S}$ be its output
Run TestLongEdges($<_\sigma, \mathcal{D}, \varepsilon, \mathcal{S}$) and **reject** if it rejects
Run TestLocalCycles($<_\sigma, \mathcal{D}, \varepsilon, \mathcal{S}$) and **reject** if it rejects
accept

Lemma 3.11. TestLocalCycles *makes* $\tilde{O}(\sqrt{n}/\varepsilon)$ *queries.*

Suppose that $<_\sigma$ *is a total ordering and* $\mathcal{S}$ *is a sketch that is consistent with* $<_\sigma$ *and satisfies (3.1). Then* TestLocalCycles *accepts with probability at least* $1 - o_n(1)$.

Suppose that any feedback edge set R *of local cycles in* G_σ *has* $\mathcal{D}(R) \geq \varepsilon/2$. *Then it rejects with probability at least* $1 - o_n(1)$.

Proof. The query complexity follows from the fact that for any edge $(u, v) \in S$, there are at most $O(\log n)$ many $w \in T$ that lie in the same block; otherwise the procedure rejects on line 3. So the number of potential triangles that we need to check is no more than $O(|S| \log n) = \tilde{O}(\sqrt{n}/\varepsilon)$.

The no case follows directly from Lemma 3.10 and Lemma 2.1.

For the yes case, we assume that the sketch $\mathcal{S}$ satisfies

$$\Pr_{u \sim \mathcal{D}^*}\left[s^{(\ell)} <_\sigma u <_\sigma s^{(\ell+1)}\right] \leq \frac{100\varepsilon \log n}{\sqrt{n}} \tag{3.2}$$

for all ℓ. Note that we only reject when T contains more than $1000 \log n$ points from some block. For the ℓ-th block, by (3.2) and a Chernoff bound, the probability of having more than $900 \log n$ points $u \in T$ with $s^{(\ell)} <_\sigma u <_\sigma s^{(\ell+1)}$ is at most n^{-9}. So by a union bound, this does not happen with probability $1 - o_n(1)$ for all blocks, in which case the number of points sampled in each block is no more than $900 \log n + 1 < 1000 \log n$ even after counting the left end point of the block. $\square$

3.3 Putting it all together: An $\tilde{O}(\sqrt{n}/\varepsilon)$ tester for total orderings

We now have everything we need to analyze our testing algorithm TestTotalOrdering.

Proof of Theorem 3.1. The query complexity is trivial.

When $<_\sigma$ is a total ordering, Sketch always returns a sketch $\mathcal{S}$ consistent with $<_\sigma$ and $\mathcal{S}$ in addition satisfies (3.1) with probability at least $1 - o_n(1)$. TestLongEdges never rejects as it is one-sided. On the other hand, when $\mathcal{S}$ satisfies (3.1), by Lemma 3.11, TestLocalCycles accepts with probability at least $1 - o_n(1)$. So the algorithm accepts with probability $1 - o_n(1)$ overall.

Suppose now that $<_\sigma$ is ε-far from total orderings with respect to $\mathcal{D}$. Assume without loss of generality that Sketch returns a sketch $\mathcal{S}$ consistent with $<_\sigma$; otherwise we are trivially done. By

Algorithm 6 PREPROCESS($f, \mathcal{D}, \varepsilon$)

Input: Oracle access to $f : \{0,1\}^n \to \{0,1\}$, sampling access to $\mathcal{D}$ and $\varepsilon > 0$

1: Draw a set T^* of $n^{1-\delta/2}/\varepsilon$ points from $\mathcal{D}$ and let $T \leftarrow T^* \setminus \{0^n\}$
2: **accept** if T is either empty, contains 0-strings only, or contains 1-strings of f only
3: **if** SKETCH(f, T) = nil **then**
4: **reject**
5: **else** (letting $\mathcal{S} = (s^{(1)}, \ldots, s^{(k)})$ be the sketch returned)
6: Run FINDBIGBLOCKS($f, \mathcal{D}, \varepsilon, \mathcal{S}$) to obtain $\mathcal{L} \subseteq [0 : k+1]$
7: **return** $(\mathcal{S}, \mathcal{L})$
8: **end if**

Corollary 3.8, either any feedback edge set of long cycles in G_σ has mass at least $\varepsilon/2$, in which case TESTLONGCYCLES rejects with probability at least 0.99 by Lemma 3.9, or any feedback edge set of local cycles has mass at least $\varepsilon/2$, in which case TESTLOCALCYCLES rejects with probability at least $1 - o_n(1)$ by Lemma 3.11. So the algorithm rejects with probability at least 2/3 overall. $\square$

4 Testing Algorithm for Monotone Decision Lists

In this section, we present a distribution-free testing algorithm for testing monotone decision lists with $\tilde{O}(n^{11/12}/\varepsilon^2)$ queries and running time.

Theorem 4.1. *There is a two-sided, adaptive distribution-free testing algorithm for monotone decision lists that makes $\tilde{O}(n^{11/12}/\varepsilon^2)$ queries and has the same running time.*

It will be used in the next section to obtain a testing algorithm for general decision lists via a direct reduction, while losing an extra factor of $1/\varepsilon$. We will focus on the query complexity of the algorithm in this section; its time complexity upper bound follows from a standard implementation.

Similar to some of the procedures from the last section, many of the procedures in this section (especially those in Sections 4.1 and 4.2) are developed to extract structural information about an unknown input in the yes case, here a monotone decision list $f : \{0,1\}^n \to \{0,1\}$. So we encourage the reader to think about this case when going through them. Of course, these procedures will be executed on functions that are not monotone decision lists. This is why many of the lemmas about performance guarantees of these procedures consist of three parts: 1) the query complexity; 2) the performance guarantees when the function f is a monotone decision list; and 3) the performance guarantees when f is just an arbitrary function.

4.1 Preprocessing

Fix $\delta > 0$ to be a positive constant, which will be set to be 1/6 at the end to optimize the query complexity of the overall algorithm.

The preprocessing stage, PREPROCESS($f, \mathcal{D}, \varepsilon$), is described in Algorithm 6. At a high level, it either outputs a pair $(\mathcal{S}, \mathcal{L})$, or tells the main algorithm that there is already enough evidence to either accept or reject the input. Here $\mathcal{S}$ is a *sketch* consistent with f to be defined next, which can be used to partition the set of variables $[n]$ into blocks (using a procedure with the same name FINDBLOCK), and $\mathcal{L}$ contains some useful information about sizes of these blocks.

PREPROCESS starts by drawing a set T^* of $n^{1-\delta/2}/\varepsilon$ many independent samples from $\mathcal{D}$, and uses $T := T^* \setminus \{0^n\}$ to build a *sketch* $\mathcal{S}$ of the underlying function f (unless when T is either empty

Algorithm 7 $\textsc{FindRep}(f, X, Y)$

Input: Oracle access to f, two sets $X, Y \subseteq \{0,1\}^n$ and X is nonempty

Let $b \leftarrow f(\bigvee_{z \in X \cup Y} z)$ and $R \leftarrow X$
while $|R| > 1$ **do**
  Partition R into $R_1 \sqcup R_2$ such that $|R_1| = \lfloor |R|/2 \rfloor$ and $|R_2| = \lceil |R|/2 \rceil$
  if $f(\bigvee_{z \in R_1 \cup Y} z) = b$ **then**
    Set $R \leftarrow R_1$
  else
    Set $R \leftarrow R_2$
  end if
end while
return the string in the singleton set R

or consists of 0-strings of f or 1-strings of f only, in which case the main algorithm accepts since either $\mathcal{D}$ has most of its mass on 0^n, or f is very close to the all-0 or all-1 function).

We define sketches as follows:

Definition 4.2. A *sketch* $\mathcal{S}$ is a tuple $\mathcal{S} = (s^{(1)}, \ldots, s^{(k)})$ of strings in $\{0,1\}^n$ for some $k \geq 2$, such that $s^{(\ell)} \neq 0^n$ for all $\ell \in [k]$. We say a sketch $\mathcal{S}$ is *consistent* with a function $f : \{0,1\}^n \to \{0,1\}$ if

$$f(s^{(\ell)}) \neq f(s^{(\ell+1)}) \quad \text{and} \quad s^{(\ell)} \succ_f s^{(\ell+1)}, \qquad \text{for all } \ell \in [k-1].$$

To describe $\textsc{Sketch}(f, T)$ we start with the following simple deterministic procedure based on binary search, called $\textsc{FindRep}(f, X, Y)$ (Algorithm 7), where $X, Y \subseteq \{0,1\}^n$ are two sets and X is nonempty. The goal of $\textsc{FindRep}$ is to find a string $x^* \in X$ that satisfies

$$f\left(x^* \vee \left(\bigvee_{y \in Y} y\right)\right) = f\left(\bigvee_{z \in X \cup Y} z\right). \tag{4.1}$$

Note that such a string always exists when f is a monotone decision list.

We summarize properties of $\textsc{FindRep}$ in the following lemma:

Lemma 4.3. $\textsc{FindRep}(f, X, Y)$ *is deterministic and makes* $O(\log |X|)$ *queries on* f.

When f is a monotone decision list, $\textsc{FindRep}$ *always returns an $x^* \in X$ that satisfies (4.1).*

On the other hand, when f is an arbitrary function, $\textsc{FindRep}$ *always returns an $x^* \in X$ but x^* does not necessarily satisfy (4.1).*

We now describe the procedure $\textsc{Sketch}(f, T)$ (Algorithm 8), where $T \subseteq \{0,1\}^n \setminus \{0^n\}$ contains at least one 0-string and at least one 1-string of f. We summarize its properties below:

Lemma 4.4. $\textsc{Sketch}(f, T)$ *is deterministic and makes* $O(|T| \log |T|)$ *queries on* f.

When f is a monotone decision list, it always returns a sketch $\mathcal{S}$ that is consistent with f.

When f is an arbitrary function, it returns either nil *or a sketch $\mathcal{S}$ and in the latter case, $\mathcal{S}$ is always a sketch consistent with f.*

Proof. The query complexity is trivial as each run of $\textsc{FindRep}$ has query complexity $O(\log |T|)$.

When f is a decision list represented by (π, ν), it is easy to verify that for all $i < j$ in $[m]$ with $f(x^{(i)}) \neq f(x^{(j)})$, we have $\min_\pi(x^{(i)}) < \min_\pi(x^{(j)})$. We also have, $k \geq 2$ given that T contains at

Algorithm 8 SKETCH(f, T)

Input: Oracle access to f and $T \subseteq \{0,1\}^n \setminus \{0^n\}$ has at least one 0-string and one 1-string of f

1: Let $m = |T|$, $T_0 \leftarrow \{x \in T : f(x) = 0\}$ and $T_1 \leftarrow \{x \in T : f(x) = 1\}$
2: **for** i from 1 to m **do**
3:     **if** both T_0 and T_1 are nonempty **then**
4:         Let $b = f(\vee_{x \in T_0 \cup T_1} x)$; Set $x^{(i)} \leftarrow$ FINDREP$(f, T_b, T_{\bar{b}})$ and $T_b \leftarrow T_b \setminus \{x^{(i)}\}$
5:     **else**
6:         Let b be such that $T_b \neq \emptyset$; Set $x^{(i)} \leftarrow$ an arbitrary string in T_b and $T_b \leftarrow T_b \setminus \{x^{(i)}\}$
7:     **end if**
8: **end for**
9: Divide $[m]$ into a disjoint union of nonempty intervals $[m] = I_1 \sqcup \cdots \sqcup I_k$ such that

   i) $f(x^{(i)}) = f(x^{(j)})$ for all $\ell \in [k]$ and all $i, j \in I_\ell$
   ii) $f(x^{(i)}) \neq f(x^{(j)})$ for all $\ell \in [k-1]$, $i \in I_\ell$ and $j \in I_{\ell+1}$

10: Check if $k \geq 2$ and $\mathcal{S} = (s^{(1)}, \ldots, s^{(k)})$ is consistent with f, where $s^{(\ell)} \leftarrow \vee_{i \in I_\ell} x^{(i)}$
11: **return** $\mathcal{S}$ if so and **return** nil otherwise

least one 0-string and at least one 1-string of f. It follows that SKETCH always returns a sketch $\mathcal{S}$, and $\mathcal{S}$ must be a sketch consistent with f.

For the general case, note that SKETCH always verifies whether $\mathcal{S}$ is a sketch consistent with f or not, and returns nil when $\mathcal{S}$ is not. The lemma follows. □

It is clear from Lemma 4.4 that if SKETCH returns nil in PREPROCESS, we know for sure that f is not a monotone decision list and thus, should be rejected. When SKETCH returns a sketch $\mathcal{S}$ in PREPROCESS$(f, \mathcal{D})$, we know it must be consistent with f and PREPROCESS continues by running a procedure called FINDBIGBLOCKS$(f, \mathcal{D}, \varepsilon, \mathcal{S})$, which uses a procedure called FINDBLOCK$(f, \mathcal{S}, x)$ that plays a similar role as the FINDBLOCK in the last section.

To motivate FINDBLOCK, we make the following observation. Let f be any function and $\mathcal{S}$ be a sketch that is consistent with f. Given $x \in \{0,1\}^n$, there must exist an index $\ell \in [0 : k+1]$ such that one of the following three conditions holds:

1. either $\ell \in [2 : k-1]$ such that $f(x) \neq f(s^{(\ell-1)}) = f(s^{(\ell+1)})$ and $s^{(\ell-1)} \succ_f x \succ_f s^{(\ell+1)}$;
2. or $\ell \in \{0, 1\}$ such that $f(x) \neq f(s^{(\ell+1)})$ and $x \succ_f s^{(\ell+1)}$;
3. or $\ell \in \{k, k+1\}$ such that $f(x) \neq f(s^{(\ell-1)})$ and $s^{(\ell-1)} \succ_f x$.

Furthermore, ℓ is *unique* when f is a monotone decision list.

The deterministic procedure FINDBLOCK$(f, \mathcal{S}, x)$ (Algorithm 9) finds such an ℓ efficiently for any given string $x \in \{0,1\}^n$:

Lemma 4.5. FINDBLOCK$(f, \mathcal{S}, x)$ *is deterministic and makes* $O(\log k)$ *queries. It always returns an* $\ell \in [0 : k+1]$ *for* x *as described above, which is unique when* f *is a monotone decision list.*

Using FINDBLOCK we partition variables $[n]$ into *blocks* (note that we cannot afford to compute these blocks but they are well defined given that FINDBLOCK is deterministic):

Definition 4.6 (Blocks)**.** For fixed f and $\mathcal{S}$, we define the ℓ-th *block* $B_{f,\mathcal{S},\ell}$ with respect to $\mathcal{S}$ as

$$B_{f,\mathcal{S},\ell} = \big\{ i \in [n] : \textsc{FindBlock}(f, \mathcal{S}, e_i) = \ell \big\}, \quad \text{for each } \ell \in [0 : k+1].$$

We usually write B_ℓ to denote $B_{f,\mathcal{S},\ell}$ when f and $\mathcal{S}$ are clear from the context.

Before moving to FindBigBlocks, we record a lemma about $\mathcal{S}$ when f is a monotone decision list. The definition below and Lemma 4.8 will only be used in the analysis of the yes case.

Definition 4.7. Let f be a monotone decision list and $\mathcal{S}$ be a sketch consistent with f. We say $\mathcal{S}$ is *scattered* if we have

$$\Pr_{x\sim\mathcal{D}} \Big[\textsc{FindBlock}(f, \mathcal{S}, x) = k+1 \Big] \le \frac{10\varepsilon \log n}{n^{1-\delta/2}}$$

and for every $\ell \in [k]$, we have (noting that $f(x) = f(s^{(\ell)})$ if $\textsc{FindBlock}(f, \mathcal{S}, x) = \ell$)

$$\Pr_{x\sim\mathcal{D}} \Big[\textsc{FindBlock}(f, \mathcal{S}, x) = \ell \text{ and } \exists i \in [n] : f(e_i) \neq f(x) \text{ and } x \succ_f e_i \succ_f s^{(\ell)} \Big] \le \frac{10\varepsilon \log n}{n^{1-\delta/2}}.$$

Lemma 4.8. *Let f be a monotone decision list, and T^* be a set of $n^{1-\delta/2}/\varepsilon$ strings drawn from $\mathcal{D}$ (as on line 1 of* $\textsc{Preprocess}(f, \mathcal{D}, \varepsilon)$*). The probability that* $\textsc{Preprocess}(f, \mathcal{D}, \varepsilon)$ *returns a sketch $\mathcal{S}$ that is not scattered is $o_n(1)$ over the randomness of T^*.*

Proof. Let (π, ν) be a representation of f. It follows from Definition 4.7 that a necessary condition for $\mathcal{S}$ to be not scattered is that there exists an interval I in $[n]$ and $b \in \{0, 1\}$ such that

$$\Pr_{x\sim\mathcal{D}} \Big[f(x) = b \text{ and } \min_\pi(x) \in I \Big] > \frac{10\varepsilon \log n}{n^{1-\delta/2}}$$

and no sample $x \in T^*$ satisfies $f(x) = b$ and $\min_\pi(x) \in I$. The lemma follows Chernoff bound and a union bound using the fact that there are only $O(n^2)$ many intervals I in $[n]$. $\square$

The preprocessing stage ends by running $\textsc{FindBigBlocks}(f, \mathcal{D}, \varepsilon, \mathcal{S})$ (Algorithm 10):

Lemma 4.9. $\textsc{FindBigBlocks}(f, \mathcal{D}, \varepsilon, \mathcal{S})$ *makes $\tilde{O}(n^{1-\delta}/\varepsilon^2)$ queries and it always returns a subset $\mathcal{L} \subseteq [0 : k+1]$. With probability at least $1 - o_n(1)$, $\mathcal{L}$ satisfies the following properties*[10]*:*

1. $|\mathcal{L}| \le n^{1-\delta}$;
2. *Let $N(\mathcal{L})$ denote the set of neighboring blocks of $\mathcal{L}$:*

   $$N(\mathcal{L}) := \big\{ \ell' \in [0 : k+1] \setminus \mathcal{L} : |\ell' - \ell| = 1 \text{ for some } \ell \in \mathcal{L} \big\}.$$

   Then we have

   $$\Pr_{x\sim\mathcal{D}} \Big[\textsc{FindBlock}(f, \mathcal{S}, x) \in N(\mathcal{L}) \Big] \le 0.1\varepsilon.$$

3. *For every $\ell \in [0 : k+1] \setminus \mathcal{L}$, we have*

   $$|B_\ell| \le \frac{16 n^\delta \log n}{\varepsilon}.$$

[10]Note that this holds no matter whether f is a monotone decision list or not.

Proof. We start by considering the $\mathcal{L}$ computed on line 6. By Chernoff bound and a union bound, we have that with probability at least $1 - o_n(1)$, $\mathcal{L}$ on line 6 satisfies the following conditions:

1. Every $\ell \in \mathcal{L}$ satisfies $|B_\ell| \geq (n^\delta \log n)/\varepsilon$, from which we also have (for the $\mathcal{L}$ on line 6)
$$|\mathcal{L}| \leq n \cdot \frac{\varepsilon}{n^\delta \log n} = \frac{\varepsilon n^{1-\delta}}{\log n}.$$
2. Every $\ell \notin \mathcal{L}$ satisfies $|B_\ell| \leq (16 n^\delta \log n)/\varepsilon$.

Condition 3 of the Lemma follow directly. For condition 1, note that the for loop beginning on line 7 only repeats for $O(1/\varepsilon)$ rounds and in each round, $\mathcal{L}$ can grow by no more than twice of the size of $\mathcal{L}$ on line 6. As a result, the final size of $\mathcal{L}$ is no more than
$$O\left(\frac{1}{\varepsilon}\right) \cdot \frac{\varepsilon n^{1-\delta}}{\log n} \leq n^{1-\delta}.$$

Finally, for condition 2, again by Chernoff bound and a union bound, with probability at least $1 - o_n(1)$, we have for every iteration of the for loop on line 7:

1. When $c < 5\log(n/\varepsilon)$, we have
$$\Pr_{x \sim \mathcal{D}}\Big[\textsc{FindBlock}(f, \mathcal{S}, x) \in N(\mathcal{L})\Big] \leq 0.1\varepsilon;$$
2. When $c \geq 5\log(n/\varepsilon)$, the probability is at least 0.01ε.

Since we repeat the for loop $200/\varepsilon$ times, it must end with an iteration with $c < 5\log(n/\varepsilon)$ (instead of reaching the last line) and this finishes the proof of the lemma. $\square$

We say $\mathcal{L}$ returned by FindBigBlocks is *good* with respect to $\mathcal{S}$ if it satisfies all conditions stated in Lemma 4.9. We will refer to $\ell \in \mathcal{L}$ as *big* blocks and $\ell \notin \mathcal{L}$ as *small* blocks.

4.2 MaxIndex

Let $f : \{0,1\}^n \to \{0,1\}$, $\mathcal{S} = (s^{(1)}, \ldots, s^{(k)})$ be a sketch that is consistent with f, and $\mathcal{L} \subseteq [0 : k+1]$ be a good set of big blocks with respect to $\mathcal{S}$. We describe a deterministic procedure MaxIndex that will play an important role in the main testing algorithm.

To motivate MaxIndex, consider the case when f is a monotone decision list (even though it will be ran on general functions). Given $x \in \{0,1\}^n \setminus \{0^n\}$, we would like to find an i such that $f(e_i) = f(x)$ and $f(e_i \vee e_j) = f(e_i)$ for all $j \in \text{supp}(x)$ such that $f(e_j) \neq f(x)$, i.e., i is one of the $f(x)$-rule variables that has priority higher than any of the $\overline{f(x)}$-rule variables in the support of x. MaxIndex$(f, \mathcal{S}, x)$ (Algorithm 11) achieves this with $\tilde{O}(n^\delta/\varepsilon)$ queries, with a caveat though that it only promises to work when x lies in a block not in $\mathcal{L}$: FindBlock$(f, \mathcal{S}, x) \notin \mathcal{L}$.

Lemma 4.10. *MaxIndex$(f, \mathcal{S}, \mathcal{L}, x)$ is a deterministic procedure. It makes $O(\log n)$ many queries on f when FindBlock$(f, \mathcal{S}, x) \in \mathcal{L}$ and $\tilde{O}(n^\delta/\varepsilon)$ many queries when FindBlock$(f, \mathcal{S}, x) \notin \mathcal{L}$. It returns either an $i \in \text{supp}(x)$ or nil; whenever it returns an $i \in \text{supp}(x)$, we always have*
$$\textsc{FindBlock}(f, \mathcal{S}, e_i) = \textsc{FindBlock}(f, \mathcal{S}, x) \quad \textit{and} \quad f(x) = f(e_i). \tag{4.2}$$
Suppose that f is a monotone decision list. Then MaxIndex always returns an i. Moreover, when FindBlock$(f, \mathcal{S}, x) \notin \mathcal{L}$, the $i \in \text{supp}(x)$ returned additionally satisfies
$$f(e_i \vee e_j) = f(e_i), \quad \textit{for all } j \in \text{supp}(x).$$

Algorithm 9 FindBlock$(f, \mathcal{S}, x)$

Input: Oracle access to f, a sketch $\mathcal{S} = (s^{(1)}, \ldots, s^{(k)})$ that is consistent with f and $x \in \{0,1\}^n$

if $f(s^{(1)}) = f(x)$ **then**
    if $f(s^{(2)} \vee x) = f(x)$ **then**
        return 1
    else if $f(s^{(2\lfloor k/2 \rfloor)} \vee x) \neq f(x)$ **then**
        return $2\lfloor k/2 \rfloor + 1$
    else
        Binary search to **return** an odd ℓ with $f(s^{(\ell-1)} \vee x) \neq f(x) = f(s^{(\ell+1)} \vee x)$
    end if
end if
The case when $f(s^{(1)}) \neq f(x)$ (or equivalently, $f(s^{(2)}) = f(x)$) is symmetric

Proof. The query complexity part is trivial. For the case with a general Boolean function f, note that MaxIndex always checks (4.2) before returning i.

When f is a monotone decision list, every e_i added to U must satisfy $s^{(\ell-1)} \succ_f e_i \succ_f s^{(\ell+1)}$. But given that $\mathcal{L}$ is good and $\ell \notin \mathcal{L}$ and the while loop on line 8 is repeated enough number of times, U by the end must contain all $i \in \mathrm{supp}(x)$ with $f(e_i) = f(x)$ and $s^{(\ell-1)} \succ_f e_i \succ_f s^{(\ell+1)}$. At least one of the e_i's in U at the end will satisfy all conditions checked on line 13. $\square$

4.3 The auxiliary graph and classification of its cycles

After the preprocessing stage, let $\mathcal{S} = (s^{(1)}, \ldots, s^{(k)})$ be a sketch that is consistent with f, and $\mathcal{L} \subseteq [0 : k+1]$ be a good set of big blocks with respect to $\mathcal{S}$. (When analyzing the yes case later, we will add the condition that $\mathcal{S}$ is scattered as well.) Given $\mathcal{S}$ and $\mathcal{L}$,

$$\textsc{FindBlock}(f, \mathcal{S}, \cdot) : \{0,1\}^n \to [0 : k+1] \quad \text{and}$$
$$\textsc{MaxIndex}(f, \mathcal{S}, \mathcal{L}, \cdot) : \{0,1\}^n \setminus \{0^n\} \to [n] \cup \{\mathsf{nil}\}$$

are two well-defined deterministic maps. We use these two maps to classify cycles in the following auxiliary directed graph H:

Definition 4.11. We write H to denote the directed (bipartite) graph with vertex set

$$V(H) = \big\{(u, W) : u \in [n] \text{ and } W \subseteq [n]\big\},$$

and there is a directed edge from (u, W_1) to (v, W_2) in H if and only if $v \in W_1$ and $f(e_u) \neq f(e_v)$.

We will refer to H as the *auxiliary* graph. The following definition shows how $\mathcal{S}$ and $\mathcal{L}$ together can be used to define a map φ from $\{0,1\}^n$ to $V(H) \cup \{\star, \mathsf{nil}\}$, which in turn induces a probability distribution over $V(H) \cap \{\star, \mathsf{nil}\}$ from $\mathcal{D}$:

Definition 4.12. The map $\varphi_{f,\mathcal{S},\mathcal{L}} : \{0,1\}^n \to V(H) \cup \{\star, \mathsf{nil}\}$ is defined as follows: $\varphi_{f,\mathcal{S},\mathcal{L}}(0^n) = \star$;

$$\varphi_{f,\mathcal{S},\mathcal{L}}(x) := \Big(\textsc{MaxIndex}(f, \mathcal{S}, \mathcal{L}, x),\ \big\{i \in \mathrm{supp}(x) : f(e_i) \neq f(x)\big\}\Big)$$

if $x \neq 0^n$ and $\textsc{MaxIndex}(f, \mathcal{S}, \mathcal{L}, x) \neq \mathsf{nil}$; and $\varphi_{f,\mathcal{S},\mathcal{L}}(x) = \mathsf{nil}$ otherwise.

Algorithm 10 FINDBIGBLOCKS$(f, \mathcal{D}, \varepsilon, \mathcal{S})$

Input: Oracle access to f, sampling access to $\mathcal{D}$, $\varepsilon > 0$ and a sketch $\mathcal{S}$ consistent with f

Create and initialize a counter $c_\ell \leftarrow 0$ for each $\ell \in [0 : k+1]$.
for $n^{1-\delta}$ times **do**
  Sample an $i \sim [n]$ uniformly at random
  Let $\ell \leftarrow$ FINDBLOCK$(f, \mathcal{S}, e_i)$ and update counter $c_\ell \leftarrow c_\ell + 1$
end for
Set $\mathcal{L}$ to be

$$\mathcal{L} \leftarrow \left\{ \ell \in [0 : k+1] : c_\ell \geq \frac{4 \log n}{\varepsilon} \right\}$$

for $200/\varepsilon$ times **do**
  Set counter $c \leftarrow 0$ and let

$$N(\mathcal{L}) \leftarrow \left\{ \ell' \in [0 : k+1] : \ell' \notin \mathcal{L} \text{ and } |\ell' - \ell| = 1 \text{ for some } \ell \in \mathcal{L} \right\}$$

  for $(100/\varepsilon)\log(n/\varepsilon)$ times **do**
    Sample a string $x \sim \mathcal{D}$
    Let $\ell \leftarrow$ FINDBLOCK$(f, \mathcal{S}, x)$ and update counter $c \leftarrow c + 1$ if $\ell \in N(\mathcal{L})$
  end for
  if $c < 5 \log(n/\varepsilon)$ **then**
    return $\mathcal{L}$
  else
    Set $\mathcal{L} \leftarrow \mathcal{L} \cup N(\mathcal{L})$
  end if
end for
return $\mathcal{L}$ ▷ This line is reached with low probability

For convenience, we will just write φ for $\varphi_{f,\mathcal{S},\mathcal{L}}$ when its subscripts are clear from the context. Given φ and $\mathcal{D}$, we write $\mathcal{D} \circ \varphi^{-1}$ to denote the push-forward of the probability distribution $\mathcal{D}$ by φ, i.e., for each $(u, W) \in V(H)$,

$$\mathcal{D} \circ \varphi^{-1}(u, W) = \sum_{x \in \{0,1\}^n \setminus \{0^n\}} \mathcal{D}(x) \cdot \mathbf{1}\big[\varphi(x) = (u, W)\big],$$

$$\mathcal{D} \circ \varphi^{-1}(\mathsf{nil}) = \sum_{x \in \{0,1\}^n \setminus \{0^n\}} \mathcal{D}(x) \cdot \mathbf{1}\big[\varphi(x) = \mathsf{nil}\big] \quad \text{and} \quad \mathcal{D} \circ \varphi^{-1}(\star) = \mathcal{D}(0^n).$$

The following lemma shows that, when f is ϵ-far from monotone decision lists with respect to $\mathcal{D}$, any feedback vertex set of H must have mass at least $\Omega(\epsilon)$ in $\mathcal{D} \circ \varphi^{-1}$.

Lemma 4.13. *Suppose that f is ε-far from monotone decision lists with respect to $\mathcal{D}$, $\mathcal{S}$ is a sketch consistent with f, and $\mathcal{L}$ is a good set of big blocks with respect to $\mathcal{S}$. Then either $\mathcal{D} \circ \varphi^{-1}(\mathsf{nil}) \geq \epsilon/2$, or we have $\mathcal{D} \circ \varphi^{-1}(U) \geq \varepsilon/2$ for any feedback vertex set $U \subseteq V(H)$ of H.*

Proof. Assume that $\mathcal{D} \circ \varphi^{-1}(\mathsf{nil}) < \epsilon/2$. Let I denote the set of $i \in [n]$ such that $(i, W) \in V(H) \setminus U$ for some W. As H after removing U is cycle-free, it induces a partial order over I. Consider any total order over I consistent with that partial order, which together with values of $f(e_i)$ (for $i \in I$) and $f(0^n)$ (for the case when $x_i = 0$ for all $i \in I$) defines a monotone decision list g on $\{0,1\}^n$.

Algorithm 11 MaxIndex$(f, \mathcal{S}, \mathcal{L}, x)$

Input: Oracle access to f, a sketch $\mathcal{S} = (s^{(1)}, \ldots, s^{(k)})$ consistent with f, $\mathcal{L} \subseteq [0 : k+1]$ that is good with respect to $\mathcal{S}$, and a string $x \in \{0,1\}^n \setminus \{0^n\}$

Let $\ell \leftarrow$ FindBlock$(f, \mathcal{S}, x)$ and $s^{(\ell+1)} \leftarrow 0^n$ if $\ell \in \{k, k+1\}$
if $\ell \in \mathcal{L}$ **then** ▷ Case when $\ell \in \mathcal{L}$
    Let $E \leftarrow \{e_j : j \in \mathrm{supp}(x)\}$ and $e_i \leftarrow$ FindRep$(f, E, \{s^{(\ell+1)}\})$
    return i if FindBlock$(f, \mathcal{S}, e_i) = \ell$ and $f(e_i) = f(x)$; **return** nil otherwise
end if
if $\ell \notin \mathcal{L}$ **then**
    Let $E \leftarrow \{e_j : j \in \mathrm{supp}(x)\}$ and $U \leftarrow \emptyset$ ▷ Case when $\ell \notin \mathcal{L}$
    while $|U| < (16n^\delta \log n)/\varepsilon$ and $f(s^{(\ell+1)} \vee (\vee_{e \in E}\, e)) = f(x)$ **do**
        Set $z \leftarrow$ FindRep$(f, E, \{s^{(\ell+1)}\})$
        Update $U \leftarrow U \cup \{z\}$ and $E \leftarrow E \setminus \{z\}$
    end while
    for each $e_i \in U$ **do**
        return i if FindBlock$(f, \mathcal{S}, e_i) = \ell$, $f(e_i) = f(x)$ and $f(e_i \vee (\vee_{e \in E}\, e)) = f(x)$
    end for
    return nil
end if

For any $x \in \{0,1\}^n \setminus \{0^n\}$ such that $\varphi(x) \in V(H) \setminus U$, we show that $g(x) = f(x)$. To see this is the case, taking any such x, we note that by Lemma 4.10 about MaxIndex,

$$\varphi(x)_1 \in \mathrm{supp}(x) \quad \text{and} \quad f(x) = f\big(e_{\varphi(x)_1}\big).$$

On the other hand, assume for a contradiction that in the monotone decision list g, $g(x)$ is set to be $\overline{f(x)} = f(e_i)$ because of some variable $i \in I$. (Note that it cannot be the case that $x_i = 0$ for all $i \in I$, since we already know that $\varphi(x)_1 \in \mathrm{supp}(x)$ and $\varphi(x)_1 \in I$ using $\varphi(x) \in V(H) \setminus U$.) Then we have that H after removing U has an edge from $\varphi(x)$ to (i, W) for some W, as $i \in I$. This implies that i is dominated by $\varphi(x)_1$ in the total order over I and thus, $g(x)$ cannot be set according to i, a contradiction.

Finally, given that $f(x) = g(x)$ for all $x \in \{0,1\}^n \setminus \{0^n\}$ such that $\varphi(x) \in V(H) \setminus U$, we have

$$\epsilon \le \mathrm{dist}_{\mathcal{D}}(f, \mathrm{monDL}) \le \mathcal{D}\big(\{x : g(x) \neq f(x)\}\big) \le \varepsilon/2 + \mathcal{D} \circ \varphi^{-1}(U).$$

This finishes the proof of the lemma. □

We further classify cycles of H into six types using $\mathcal{S}, \mathcal{L}$ and the map FindBlock$(f, \mathcal{S}, \cdot)$. It follows from Lemma 4.13 that, for some $c \in [5]$, any vertex feedback set of type-c cycles in H must have mass $\Omega(\epsilon)$ in $\mathcal{D} \circ \varphi^{-1}$ (it will become clear that we don't need to deal with type-0 cycles). Our main testing algorithm then consists of five procedures, each handling one type of cycles.

Definition 4.14 (Types of cycles in H)**.** Let C be a directed cycle in H. We say

0. C is of type 0 if it contains a vertex (u, W) with FindBlock$(f, \mathcal{S}, e_u) \in N(\mathcal{L})$;

1. C is of type 1 if it contains a directed edge $(u, W_1) \to (v, W_2)$ that satisfies

$$\textsc{FindBlock}(f, \mathcal{S}, e_v) \le \textsc{FindBlock}(f, \mathcal{S}, e_u) - 2;$$

2. C is of type 2 if it contains a directed edge $(u, W_1) \to (v, W_2)$ that satisfies

$$\textsc{FindBlock}(f, \mathcal{S}, e_v) = \textsc{FindBlock}(f, \mathcal{S}, e_u) - 1$$

   and both $\textsc{FindBlock}(f, \mathcal{S}, e_u), \textsc{FindBlock}(f, \mathcal{S}, e_v) \in \mathcal{L}$;

3. C is of type 3 if it contains a directed edge $(u, W_1) \to (v, W_2)$ that satisfies

$$\left|\textsc{FindBlock}(f, \mathcal{S}, e_u) - \textsc{FindBlock}(f, \mathcal{S}, e_v)\right| = 1$$

   and $\textsc{FindBlock}(f, \mathcal{S}, e_u), \textsc{FindBlock}(f, \mathcal{S}, e_v) \notin \mathcal{L} \cup N(\mathcal{L})$ and $f(e_u \vee e_v) \neq f(e_u)$;

4. C is of type 4 if it contains two consecutive edges $(u, W_1) \to (v, W_2) \to (w, W_3)$ such that

$$\textsc{FindBlock}(f, \mathcal{S}, e_w) + 2 = \textsc{FindBlock}(f, \mathcal{S}, e_v) + 1 = \textsc{FindBlock}(f, \mathcal{S}, e_u) \tag{4.3}$$

   and $\textsc{FindBlock}(f, \mathcal{S}, e_u), \textsc{FindBlock}(f, \mathcal{S}, e_v), \textsc{FindBlock}(f, \mathcal{S}, e_w) \notin \mathcal{L} \cup N(\mathcal{L})$ and

$$f(e_u \vee e_v) = f(e_u) \quad \text{and} \quad f(e_v \vee e_w) = f(e_v);$$

5. C is of type 5 if it (1) has length at least 4, (2) satisfies $\textsc{FindBlock}(f, \mathcal{S}, e_u) \notin \mathcal{L} \cup N(\mathcal{L})$ for all (u, W) in C, (3) satisfies $f(e_u \vee e_v) = f(e_u)$ for all edges $(u, W_1) \to (v, W_2)$ in C, and (4)

$$\max_{(u,W) \in C} \textsc{FindBlock}(f, \mathcal{S}, e_u) = \min_{(u,W) \in C} \textsc{FindBlock}(f, \mathcal{S}, e_u) + 1. \tag{4.4}$$

The following lemma shows that every cycle in H falls into at least one of the five types:

Lemma 4.15. *Any cycle in H must be of type c for at least one $c \in \{0, 1, \ldots, 5\}$.*

Proof. Let C be a cycle in H with the following vertex sequence:

$$(u_1, W_1), (u_2, W_2), \ldots, (u_\ell, W_\ell), (u_{\ell+1}, W_{\ell+1}) = (u_1, W_1),$$

i.e. $u_{r+1} \in W_r$ for all $r \in [\ell]$. Given that H is bipartite, $\ell \geq 2$ and must be an even number.

We start by showing that if C is neither type 1 nor type 2, then it must be the case that

- either $\textsc{FindBlock}(f, \mathcal{S}, e_u) \in \mathcal{L}$ for all (u, W) in C;
- or $\textsc{FindBlock}(f, \mathcal{S}, e_u) \notin \mathcal{L} \cup N(\mathcal{L})$ for all (u, W) in C.

Otherwise, given the definition of $N(\mathcal{L})$, there must be $(u, W_1), (v, W_2) \in \mathcal{L}$ such that

$$\textsc{FindBlock}(f, \mathcal{S}, e_u) \leq \textsc{FindBlock}(f, \mathcal{S}, e_v) - 2.$$

However, the part of the cycle C from (v, W_2) to (u, W_1) can never skip a block whenever it goes down in blocks (since C is not type 1). Therefore, the cycle must visit every block between that of u and that of v. But given that one of them is in $N(\mathcal{L})$, C must be of type-0, a contradiction. It is easy to show that, when $\textsc{FindBlock}(f, \mathcal{S}, e_u) \in \mathcal{L}$ for all (u, W) in C, C must be of type 2. So we are left with the case when $\textsc{FindBlock}(f, \mathcal{S}, e_u) \notin \mathcal{L} \cup N(\mathcal{L})$ for all (u, W) in C.

Assume that the cycle C is not type 3 or type 4. We finish the proof by showing that C must be of type 5. First, given that C is not of type 3 or type 4, there cannot be two consecutive edges $(u, W_1) \to (v, W_2) \to (w, W_3)$ that satisfy (4.3). Combining this with the assumption that the cycle is not type 1, C must satisfy (4.4). We also note that C cannot be of length 2 (otherwise, it is of type 3), and it must satisfy $f(e_u \vee e_v) = f(e_u)$ for all edges $(u, W_1) \to (v, W_2)$ in C (otherwise, it is of type 3). This finishes the proof of the lemma. ∎

Algorithm 12 MonotoneDL$(f, \mathcal{D}, \epsilon)$

Input: Oracle access to $f : \{0,1\}^n \to \{0,1\}$, sampling access to $\mathcal{D}$ and $\epsilon > 0$

```
1: Run Preprocess(f, D, ε)
2: if it accepts or rejects then
3:     return the same answer
4: else
5:     Let (S, L) be the pair it returns
6: end if
7: for O(1/ε) times do
8:     Draw x ~ D and run MaxIndex(f, S, x); reject if it returns nil
9: end for
10: TestType-1(f, D, ε, S, L) and reject if it rejects
11: TestType-2(f, D, ε, S, L) and reject if it rejects
12: TestType-3(f, D, ε, S, L) and reject if it rejects
13: TestType-4(f, D, ε, S, L) and reject if it rejects
14: TestType-5(f, D, ε, S, L) and reject if it rejects
15: accept
```

Corollary 4.16. *Suppose f is ϵ-far from monotone decision lists with respect to $\mathcal{D}$, $\mathcal{S}$ is a sketch consistent with f, and $\mathcal{L}$ is a good set of blocks with respect to $\mathcal{S}$. Then either $\mathcal{D} \circ \varphi^{-1}(\mathsf{nil}) \geq \epsilon/2$, or for some $c \in [5]$, any feedback vertex set U of type-c cycles in H has $\mathcal{D} \circ \varphi^{-1}(U) \geq \Omega(\varepsilon)$.*

Proof. This follows from Lemma 4.13, Lemma 4.15, as well as the fact that the mass of $\mathcal{D} \circ \varphi^{-1}$ on (u, W) with FindBlock$(f, \mathcal{S}, e_u) \in N(\mathcal{L})$ is at most 0.1ϵ given that $\mathcal{L}$ is good. $\square$

4.4 The main testing algorithm and proof of Theorem 4.1

The testing algorithm MonotoneDL$(f, \mathcal{D}, \epsilon)$ for monotone decision lists is given in Algorithm 12. After running the preprocessing stage to obtain a sketch $\mathcal{S}$ and a set $\mathcal{L}$ of big blocks, the algorithm quickly checks whether $\mathcal{D} \circ \varphi^{-1}(\mathsf{nil}) \geq \epsilon/2$ or not in line 8. The rest of the algorithm then consists of one procedure for each of the five types of cycles.

We list performance guarantees of these procedures in lemmas below. In all lemmas we assume

1. $f : \{0,1\}^n \to \{0,1\}$ and $\mathcal{D}$ is a probability distribution over $\{0,1\}^n$;
2. $\mathcal{S} = (s^{(1)}, \ldots, s^{(k)})$ is a sketch that is consistent with f; and
3. $\mathcal{L}$ is a good set of big blocks with respect to $\mathcal{S}$.

In each lemma we describe performance guarantees of the procedure when f is a monotone decision list and when any feedback vertex set for type-c cycles in H, for some $c \in [5]$, is at least $\Omega(\epsilon)$.

Lemma 4.17. TestType-1 *makes $\tilde{O}(n^{0.5+\delta}/\epsilon^2)$ queries and always accepts when f is a monotone decision list. If any feedback vertex set of type-1 cycles in H has probability mass $\Omega(\epsilon)$ in $\mathcal{D} \circ \varphi^{-1}$, then* TestType-1 *rejects with probability at least* 0.9.

Lemma 4.18. TestType-2 *makes $\tilde{O}(n^{1-\delta/2}/\epsilon)$ queries.*

When f is a monotone decision list and $\mathcal{S}$ is scattered, it rejects with probability at most $o_n(1)$.

If any feedback vertex set of type-2 cycles in H has probability mass at least $\Omega(\epsilon)$ in $\mathcal{D} \circ \varphi^{-1}$, then TestType-2 *rejects with probability at least* 0.9.

Lemma 4.19. $\textsc{TestType-3}$ *makes* $\tilde{O}(n^{0.5+\delta}/\epsilon^2)$ *queries and always accepts when* f *is a monotone decision list. If any feedback vertex set of type-*3 *cycles in* H *has probability mass* $\Omega(\epsilon)$ *in* $\mathcal{D}\circ\varphi^{-1}$*, then* $\textsc{TestType-3}$ *rejects with probability at least* 0.9*.*

Lemma 4.20. $\textsc{TestType-4}$ *makes* $\tilde{O}(n^{2/3+\delta}/\epsilon^2)$ *queries and always accepts when* f *is a monotone decision list. If any feedback vertex set of type-*4 *cycles in* H *has probability mass* $\Omega(\epsilon)$ *in* $\mathcal{D}\circ\varphi^{-1}$*, then* $\textsc{TestType-4}$ *rejects with probability at least* 0.9*.*

Lemma 4.21. $\textsc{TestType-5}$ *makes* $\tilde{O}(n^{3/4+\delta}/\epsilon^2)$ *queries and always accepts when* f *is a monotone decision list. If any feedback vertex set of type-*5 *cycles in* H *has probability mass* $\Omega(\epsilon)$ *in* $\mathcal{D}\circ\varphi^{-1}$*, then* $\textsc{TestType-5}$ *rejects with probability at least* 0.9*.*

We prove these five lemmas in Section 4.5. Theorem 4.1 follows directly:

Proof of Theorem 4.1. The overall query complexity of $\textsc{MonotoneDL}$ is

$$\tilde{O}\left(\frac{n^{1-\delta/2}}{\epsilon}\right)+\tilde{O}\left(\frac{n^{1-\delta}}{\epsilon^2}\right)+\tilde{O}\left(\frac{n^{3/4+\delta}}{\epsilon^2}\right)=\tilde{O}\left(\frac{n^{11/12}}{\epsilon^2}\right),$$

when δ is set to be $1/6$.

When f is a monotone decision list, the only possibility for it to be rejected is by $\textsc{TestType-2}$. But by Lemma 4.8, $\mathcal{S}$ is not scattered with probability $o_n(1)$ and when it is scattered, by Lemma 4.18, $\textsc{TestType-2}$ rejects with probability $o_n(1)$.

When f is ϵ-far from monotone decision lists with respect to $\mathcal{D}$, it is accepted by $\textsc{Preprocess}$ with probability $o_n(1)$, given that $\mathcal{D}$ cannot have more than $1-\epsilon$ mass on 0^n and that f cannot be ϵ-close to the all-0 or all-1 function with respect to $\mathcal{D}$. Therefore, with probability at least $1-o_n(1)$ either $\textsc{MonotoneDL}$ already rejected $(f,\mathcal{D})$ or it reaches line 7 with a sketch $\mathcal{S}$ consistent with f and an $\mathcal{L}$ that is good with respect to $\mathcal{S}$ by Lemma 4.9. When this happens, either the probability of $\textsc{MaxIndex}(f,\mathcal{S},x)=\mathsf{nil}$ as $x\sim\mathcal{D}$ is at least 0.5ϵ, in which case line 8 rejects with probability at least 0.9, or the rest of $\textsc{MonotoneDL}$ rejects with probability at least 0.9 by Lemma 4.17, 4.18, 4.19, 4.20 and 4.21. This finishes the proof of the theorem. ☐

4.5 Main procedures and their analyses

Proof of Lemma 4.17. The procedure $\textsc{TestType-1}(f,\mathcal{D},\epsilon,\mathcal{S},\mathcal{L})$ is described in Algorithm 13.

First we show that $\textsc{TestType-1}$ never rejects when f is a monotone decision list. To see this is the case, assume for a contradiction that the event occurs on $x\in P$ and $y\in Q$. Let

$$i=\textsc{MaxIndex}(f,\mathcal{S},\mathcal{L},y),\quad \ell=\textsc{FindBlock}(f,\mathcal{S},x),\quad\text{and}\quad \ell'=\textsc{FindBlock}(f,\mathcal{S},y).$$

We have by Lemma 4.10 that

$$\textsc{FindBlock}(f,\mathcal{S},e_i)=\textsc{FindBlock}(f,\mathcal{S},y)=\ell'\le\ell-2.$$

Using $i\in\mathrm{supp}(x)$, we have $x\succ_f e_i\succ_f s^{(\ell'+1)}$ which contradicts with $s^{(\ell-1)}\succ_f x$ and $\ell\ge\ell'+2$.

Now we consider the case when any feedback vertex set of type-1 cycles in H has mass at least $\Omega(\epsilon)$ in $\mathcal{D}\circ\varphi^{-1}$. To this end, we apply the birthday paradox lemma (Lemma 2.1) on the following bipartite graph G. The left side of G has vertices $U:=[n]$; the right side has vertices $V:=V(H)$; and $(v,(u,W))$ is an edge in G if and only if $v\in W$, $f(e_u)\ne f(e_v)$, and

$$\textsc{FindBlock}(f,\mathcal{S},e_v)\le\textsc{FindBlock}(f,\mathcal{S},e_u)-2.$$

Algorithm 13 TESTTYPE-1$(f, \mathcal{D}, \epsilon, \mathcal{S}, \mathcal{L})$

1: Draw two sets P^*, Q^* of $O(\sqrt{n}/\varepsilon)$ samples from $\mathcal{D}$; let $P \leftarrow P^* \setminus \{0^n\}$ and $Q \leftarrow Q^* \setminus \{0^n\}$
2: For each $x \in P \cup Q$, compute FINDBLOCK$(f, \mathcal{S}, x)$ and MAXINDEX$(f, \mathcal{S}, \mathcal{L}, x)$
3: **reject** if there exist $x \in P$ and $y \in Q$ such that $f(x) \neq f(y)$,

$$\textsc{FindBlock}(f, \mathcal{S}, y) \leq \textsc{FindBlock}(f, \mathcal{S}, x) - 2 \quad \text{and} \quad \textsc{MaxIndex}(f, \mathcal{S}, \mathcal{L}, y) \in \mathrm{supp}(x)$$

4: **accept** otherwise

Algorithm 14 TESTTYPE-2$(f, \mathcal{D}, \epsilon, \mathcal{S}, \mathcal{L})$

1: Draw a set P^* of samples and a set Q^* of samples from $\mathcal{D}$ of size given as follows:

$$\frac{n^{\delta/2}}{\varepsilon \log^2 n} \quad \text{and} \quad \frac{n^{1-\delta/2} \log^3 n}{\epsilon}, \quad \text{respectively}$$

2: Let $P \leftarrow P^* \setminus \{0^n\}$ and $Q \leftarrow Q^* \setminus \{0^n\}$
3: For each $x \in P \cup Q$, compute FINDBLOCK$(f, \mathcal{S}, x)$
4: For each $x \in P \cup Q$ with FINDBLOCK$(f, \mathcal{S}, x) \in \mathcal{L}$, compute MAXINDEX$(f, \mathcal{S}, \mathcal{L}, x)$
5: **reject** if there exist $x \in P$ and $y \in Q$ such that

i) FINDBLOCK$(f, \mathcal{S}, y) =$ FINDBLOCK$(f, \mathcal{S}, x) - 1$;
ii) FINDBLOCK$(f, \mathcal{S}, x)$, FINDBLOCK$(f, \mathcal{S}, y) \in \mathcal{L}$; and
iii) MAXINDEX$(f, \mathcal{S}, \mathcal{L}, y) \in \mathrm{supp}(x)$

6: **accept** otherwise

Algorithm 15 TESTTYPE-3$(f, \mathcal{D}, \epsilon, \mathcal{S}, \mathcal{L})$

1: Draw two sets P^*, Q^* of $O(\sqrt{n}/\epsilon)$ samples from $\mathcal{D}$; let $P \leftarrow P^* \setminus \{0^n\}$ and $Q \leftarrow Q^* \setminus \{0^n\}$
2: Compute FINDBLOCK$(f, \mathcal{S}, x)$ and MAXINDEX$(f, \mathcal{S}, \mathcal{L}, x)$ for each $x \in P \cup Q$
3: **reject** if there exist $x \in P$ and $y \in Q$ such that

i) $\left|\textsc{FindBlock}(f, \mathcal{S}, x) - \textsc{FindBlock}(f, \mathcal{S}, y)\right| = 1$;
ii) FINDBLOCK$(f, \mathcal{S}, x)$, FINDBLOCK$(f, \mathcal{S}, y) \notin \mathcal{L} \cup N(\mathcal{L})$;
iii) MAXINDEX$(f, \mathcal{S}, \mathcal{L}, y) \in \mathrm{supp}(x)$; and
iv) $f(e_u \vee e_v) \neq f(e_u)$, where $u =$ MAXINDEX$(f, \mathcal{S}, \mathcal{L}, x)$ and $v =$ MAXINDEX$(f, \mathcal{S}, \mathcal{L}, y)$

4: **accept** otherwise

To apply Lemma 2.1, the distribution μ over $U \cup \{\#\}$ is defined as

$$\mu(u) = \sum_{(u,W) \in V(H)} \mathcal{D} \circ \varphi^{-1}(u, W) \quad \text{and} \quad \mu(\#) = \mathcal{D} \circ \varphi^{-1}(\star) + \mathcal{D} \circ \varphi^{-1}(\mathsf{nil}).$$

The distribution ν over $V \cup \{\#\}$ is exactly $\mathcal{D} \circ \varphi^{-1}$ except that $\nu(\#) = \mathcal{D} \circ \varphi^{-1}(\star) + \mathcal{D} \circ \varphi^{-1}(\mathsf{nil})$. It follows that any vertex cover of G must have total weight at least $\Omega(\epsilon)$. The no-part of the lemma follows directly from Lemma 2.1. $\square$

Algorithm 16 TESTTYPE-4($f, \mathcal{D}, \epsilon, \mathcal{S}, \mathcal{L}$)

1: Draw a set P^* of $O(n^{2/3}/\epsilon)$ samples from $\mathcal{D}$; let $P \leftarrow P^* \setminus \{0^n\}$
2: Compute FINDBLOCK($f, \mathcal{S}, x$) and MAXINDEX($f, \mathcal{S}, \mathcal{L}, x$) for each $x \in P$
3: **reject** if there exist $x, y, z \in P$ such that

  i) FINDBLOCK($f, \mathcal{S}, z$) $+ 2 =$ FINDBLOCK($f, \mathcal{S}, y$) $+ 1 =$ FINDBLOCK($f, \mathcal{S}, x$);

  ii) FINDBLOCK($f, \mathcal{S}, x$), FINDBLOCK($f, \mathcal{S}, y$), FINDBLOCK($f, \mathcal{S}, z$) $\notin \mathcal{L} \cup N(\mathcal{L})$; and

  iii) $f(e_u \vee e_v) = f(e_u)$ and $f(e_v \vee e_w) = f(e_v)$: u, v and w are MAXINDEX of x, y and z

4: **accept** otherwise

Algorithm 17 TESTTYPE-5($f, \mathcal{D}, \epsilon, \mathcal{S}, \mathcal{L}$)

1: Draw a set P^* of $O(n^{3/4}/\epsilon)$ samples from $\mathcal{D}$; let $P \leftarrow P^* \setminus \{0^n\}$
2: Compute FINDBLOCK$_{f,\mathcal{S}}(x)$ and MAXINDEX$_{f,\mathcal{S}}(x)$ for each $x \in P$
3: **reject** if there exist $x^1, x^2, x^3, x^4 \in P$ such that

  i) Let ℓ_1, ℓ_2, ℓ_3 and ℓ_4 be the FINDBLOCK of x^1, x^2, x^3 and x^4, respectively

  ii) $\ell_1 = \ell_3 = \ell_2 + 1 = \ell_4 + 1$ and $\ell_1, \ell_2, \ell_3, \ell_4 \notin \mathcal{L} \cup N(\mathcal{L})$

  iii) Let u_1, u_2, u_3 and u_4 be the MAXINDEX of x^1, x^2, x^3 and x^4, respectively

  iv) $f(e_{u_1} \vee e_{u_2}) = f(e_{u_3} \vee e_{u_4}) = 0$ and $f(e_{u_2} \vee e_{u_3}) = f(e_{u_4} \vee e_{u_1}) = 1$.

4: **accept** otherwise

Proof of Lemma 4.18. TESTTYPE-2 is described in Algorithm 14. It uses $\tilde{O}(n^{1-\delta/2}/\epsilon)$ queries as each call to MAXINDEX uses $O(\log n)$ queries when FINDBLOCK($f, \mathcal{S}, x$) $\in \mathcal{L}$ given that $\mathcal{L}$ is good.

For the yes case, if TESTTYPE-2 rejects because of $x \in P$ and $y \in Q$, it must be either

1. FINDBLOCK($f, \mathcal{S}, x$) $= k + 1$; or
2. Letting $\ell =$ FINDBLOCK($f, \mathcal{S}, x$) $\in [k]$, $\exists\, i \in [n]$ such that $f(e_i) \neq f(x)$ and $x \succ_f e_i \succ_f s^{(\ell)}$.

As $\mathcal{S}$ is scattered, it follows from $|\mathcal{L}| \leq n^{1-\delta}$ that the probability of P having such an x is at most

$$\frac{10\epsilon \log n}{n^{1-\delta/2}} \cdot n^{1-\delta} \cdot \frac{n^{\delta/2}}{\epsilon \log^2 n} = o_n(1).$$

The proof of the no case is similar to that of Lemma 4.17. Assume that any feedback vertex set of type-2 cycles in H has mass at least $\Omega(\epsilon)$ in $\mathcal{D} \circ \varphi^{-1}$. The left side of G has vertices $U := [n]$ and the right side has vertices $V := V(H)$, and $(v, (u, W))$ is in G iff $v \in W$, $f(e_u) \neq f(e_v)$ and

$$\text{FINDBLOCK}(f, \mathcal{S}, e_v) = \text{FINDBLOCK}(f, \mathcal{S}, e_u) - 1$$

and FINDBLOCK($f, \mathcal{S}, e_u$), FINDBLOCK($f, \mathcal{S}, e_v$) $\in \mathcal{L}$.

The distribution μ over $U \cup \{\#\}$ and ν over $V \cup \{\#\}$ is defined in the same way as those in the proof of Lemma 4.17. It follows that any vertex cover of G must have total weight at least $\Omega(\epsilon)$. The no-part of the lemma follows directly from Lemma 2.1. $\square$

Proof of Lemma 4.19. The procedure TESTTYPE-3 is described in Algorithm 15. For the analysis of its query complexity, the only nontrivial part is to observe that we don't need $|P| \times |Q|$ many queries to evaluate $f(e_u \vee e_v)$ on line iv) but

$$|P| \cdot \frac{32n^\delta \log n}{\epsilon}$$

queries suffice. This is because

$$\textsc{FindBlock}(f, \mathcal{S}, x) = \textsc{FindBlock}(f, \mathcal{S}, e_u) \quad \text{and} \quad \textsc{FindBlock}(f, \mathcal{S}, y) = \textsc{FindBlock}(f, \mathcal{S}, e_v)$$

and thus, for any u, the number of v that can satisfy i) and ii) is no more than by $2 \cdot (16n^\delta \log n)/\epsilon$ given that all these are small blocks outside of $\mathcal{L} \cup N(\mathcal{L})$.

It is easy to verify that TestType-3 never rejects when f is a monotone decision list. Assume for a contradiction that the event occurs on $x \in P$ and $y \in Q$ with u, v given in iv). Given that $v \in \mathrm{supp}(x)$, we should have $f(e_u \vee e_v) = f(e_u)$ by Lemma 4.10, a contradiction.

The proof of the no case is similar to that of the previous two lemmas. We apply the birthday paradox lemma (Lemma 2.1) on the following bipartite graph G. The left side has vertices $U = [n]$ and the right side of G has vertices $V = V(H)$, and $(v, (u, W))$ is an edge in G if and only if $v \in W$, $f(e_u) \neq f(e_v)$, $|\textsc{FindBlock}(f, \mathcal{S}, e_u) - \textsc{FindBlock}(f, \mathcal{S}, e_v)| = 1$,

$$\textsc{FindBlock}(f, \mathcal{S}, e_u), \textsc{FindBlock}(f, \mathcal{S}, e_v) \notin \mathcal{L} \cup N(\mathcal{L}),$$

and $f(e_u \vee e_v) \neq f(e_u)$. The rest of the proof is similar so we skip the details. $\square$

Proof of Lemma 4.20. The procedure TestType-4$(f, \mathcal{D}, \mathcal{S}, \mathcal{L})$ is described in Algorithm 16. For the analysis of its query complexity, we make the same observation as in the previous lemma that the number of queries needed for $f(e_u \vee e_v)$ and $f(e_v \vee e_w)$ in iii) is at most $|P| \cdot O((n^\delta \log n)/\epsilon)$.

For the yes case, we assume for a contradiction that f is a monotone decision list but there are $x, y, z \in P$ and u, v, w that satisfy i), ii) and iii). Then we have $e_u \succ_f e_v \succ_f e_w$ but

$$\begin{aligned} \textsc{FindBlock}(f, \mathcal{S}, e_u) &= \textsc{FindBlock}(f, \mathcal{S}, x) \\ &= \textsc{FindBlock}(f, \mathcal{S}, z) + 2 = \textsc{FindBlock}(f, \mathcal{S}, e_w) + 2, \end{aligned}$$

a contradiction.

For the no case, we apply the hypergraph birthday paradox lemma (Lemma 2.2). We consider the following 3-uniform hypergraph G over $[n]$: $\{u, v, w\}$ is an edge of G if and only if

$$\textsc{FindBlock}(f, \mathcal{S}, e_w) + 2 = \textsc{FindBlock}(f, \mathcal{S}, e_v) + 1 = \textsc{FindBlock}(f, \mathcal{S}, e_u),$$

$\textsc{FindBlock}(f, \mathcal{S}, e_u), \textsc{FindBlock}(f, \mathcal{S}, e_v), \textsc{FindBlock}(f, \mathcal{S}, e_w) \notin \mathcal{L} \cup N(\mathcal{L})$, $f(e_u \vee e_v) = f(e_u)$ and $f(e_v \vee e_w) = f(e_v)$. The distribution μ over $[n] \cup \{\#\}$ is defined naturally as

$$\mu(u) = \sum_{(u,W) \in V(H)} \mathcal{D} \circ \varphi^{-1}(u, W) \quad \text{and} \quad \mu(\#) = \mathcal{D} \circ \varphi^{-1}(\star) + \mathcal{D} \circ \varphi^{-1}(\mathsf{nil}).$$

It follows from the assumption of the lemma that any vertex cover of G must have mass at least $\Omega(\epsilon)$. The no part of the lemma follows directly from Lemma 2.2. $\square$

Proof of Lemma 4.21. The procedure TestType-5 is described in Algorithm 17. The analysis of its query complexity is similar to that of the previous two lemmas.

To see that TestType-5 always rejects when f is a monotone decision list, it is easy to verify that iv) cannot be consistent with any monotone decision list since $f(e_{u_1} \vee e_{u_2}) = f(e_{u_3} \vee e_{u_4}) = 0$ implies that $f(e_{u_1} \vee e_{u_2} \vee e_{u_3} \vee e_{u_4}) = 0$ but the second part implies it to be 1.

For the no case, we consider the following 4-uniform hypergraph G over $[n]$: $\{u_1, u_2, u_3, u_4\}$ is an edge of G if and only if

$$\begin{aligned}\textsc{FindBlock}(f, \mathcal{S}, e_{u_1}) &= \textsc{FindBlock}(f, \mathcal{S}, e_{u_3}) \\ &= \textsc{FindBlock}(f, \mathcal{S}, e_{u_2}) + 1 = \textsc{FindBlock}(f, \mathcal{S}, e_{u_4}) + 1\end{aligned}$$

and $f(e_{u_1} \vee e_{u_2}) = f(e_{u_3} \vee e_{u_4}) = 0$ and $f(e_{u_2} \vee e_{u_3}) = f(e_{u_4} \vee e_{u_1}) = 1$. The distribution μ over $[n] \cup \{\#\}$ is defined naturally as

$$\mu(u) = \sum_{(u,W) \in V(H)} \mathcal{D} \circ \varphi^{-1}(u, W) \quad \text{and} \quad \mu(\#) = \mathcal{D} \circ \varphi^{-1}(\star) + \mathcal{D} \circ \varphi^{-1}(\mathsf{nil}).$$

It follows from the assumption of the lemma and Lemma 4.22 that any vertex cover of G must have mass at least $\Omega(\epsilon)$. The no part of the lemma follows directly from Lemma 2.2. $\square$

Lemma 4.22. *Let $G = (U, V, E)$ be a complete bipartite graph with an edge labeling $\phi : E \to \{0, 1\}$. For any integer $k \geq 2$, we call a sequence of vertices $(u_1, u_2, \ldots, u_{2k})$ an alternating $2k$-cycle in (G, ϕ) if the following holds:*

- *$u_1, \ldots, u_{2k-1} \in U$ and $u_2, \ldots, v_{2k} \in V$.*
- *$\phi(\{u_{2i-1}, u_{2i}\}) = 0$ and $\phi(\{u_{2i}, u_{2i+1}\}) = 1$ for all $i \in [k]$, where $u_{2k+1} := u_1$.*

If (G, ϕ) has an alternating $2k$-cycle, then it also has an alternating 4-cycle.

Proof. Assume that $k \geq 3$ is the smallest integer such that (G, ϕ) contains an alternating $2k$-cycle, and let $(u_1, \ldots, u_{2k})$ be such a cycle. On one hand, if $\phi(\{u_2, u_{2k-1}\}) = 1$, then $(u_1, u_2, u_{2k-1}, u_{2k})$ forms an alternating 4-cycle. On the other hand, if $\phi(\{u_2, u_{2k-1}\}) = 0$, then $(u_{2k-1}, u_2, u_3, \ldots, u_{2k-2})$ forms an alternating $2(k-1)$-cycle. In both cases we find a contradiction to the assumed minimality of k, so there must exist an alternating 4-cycle in (G, ϕ). $\square$

5 Testing Algorithm for Decision Lists

We prove Theorem 1.1 by giving a reduction from the problem of testing decision lists to that of testing monotone decision lists. We start with a standard amplification on MonotoneDL to get MonotoneDL* with the following properties:

1. The number of queries made by MonotoneDL$^*(f, \mathcal{D}, \epsilon)$, denoted by $N_{n,\epsilon}$, is $\tilde{O}(n^{11/12}/\epsilon^2)$;
2. For any $(f, \mathcal{D})$ such that f is a monotone decision list, MonotoneDL$^*(f, \mathcal{D}, \epsilon)$ accepts with probability at least $1 - o_n(1)$; and
3. For any $(f, \mathcal{D})$ such that f is ϵ-far from monotone decision lists with respect to $\mathcal{D}$, we have that MonotoneDL$^*(f, \mathcal{D}, \epsilon)$ rejects with probability at least $1 - o_n(1)$.

The algorithm DecisionList$(f, \mathcal{D}, \epsilon)$ for testing general decision lists is described in Algorithm 18. It uses a procedure called CheckDL described in Algorithm 21.

The high-level idea behind the reduction is that when f is a decision list represented by (π, μ, ν), if we happen to know the minimum element in the decision list (i.e., $r \in \{0,1\}^n$ such that $r_{\pi(j)} \neq \mu_j$ for all $j \in [n]$ or equivalently, $\min_{\pi,\mu}(r) = n+1$, then g defined as $g(x) := f(x \oplus r)$ would become a monotone decision list on which we can run MonotoneDL*. Of course, it is not clear how to find

Algorithm 18 DecisionList$(f, \mathcal{D}, \epsilon)$

Input: Oracle access to f, sampling access to $\mathcal{D}$, and a distance parameter $\epsilon > 0$

1: **for** $100/\epsilon$ rounds **do**
2:     Draw $r \sim \mathcal{D}$ and set counter $c \leftarrow 0$
3:     **for** $100 \log(n/\epsilon)$ rounds **do**
4:         Run CheckDL$(f, \mathcal{D}, \epsilon, r)$ and set $c \leftarrow c + 1$ if it accepts
5:     **end for**
6:     **accept** if $c \geq \log(n/\epsilon)$
7: **end for**
8: **reject**

the minimum element efficiently, but we will show that it suffices to work with an element that is *close* to being the minimum.

In more details, let's consider the case when f is a decision list and is represented by (π, μ, ν). Since we repeat the main loop of DecisionList (which starts on line 1) for $100/\epsilon$ times, it is not hard to show that with probability at least 0.9, at least one of the r sampled on line 2 satisfies

$$\Pr_{x \sim \mathcal{D}}\left[\min_{\pi,\mu}(r) < \min_{\pi,\mu}(x)\right] \leq 0.1\epsilon. \tag{5.1}$$

To see this is the case, let $j^* \in [n+1]$ be the largest integer such that

$$\Pr_{x \sim \mathcal{D}}\left[\min_{\pi,\mu}(x) \geq j\right] \geq 0.1\epsilon.$$

Then at least one of the r's sampled on line 2 satisfies $\min_{\pi,\mu}(x) \geq j^*$ with probability at least 0.9 and any such r satisfies (5.1).

Assuming that r satisfies (5.1). The simple subroutine TestDL described in Algorithm 19 can help us test whether f is a decision list with almost one-sided error. Its performance guarantees are stated in the following lemma:

Lemma 5.1. TestDL *makes* $\tilde{O}(n^{11/12}/\epsilon^2)$ *many queries.*

If f is ϵ-far from decision lists with respect to $\mathcal{D}$, then for any strings $r, z \in \{0,1\}^n$, TestDL *rejects with probability at least $1 - o_n(1)$.*

Suppose f is a decision list represented by (π, μ, ν), r satisfies (5.1) and z satisfies $\min_{\pi,\mu}(z) > \min_{\pi,\mu}(r)$*. Then* TestDL *accepts with probability at least $1 - o_n(1)$.*

Proof. The query complexity part is trivial. When f is ϵ-far from decision lists with respect to $\mathcal{D}$, note that either $\mathrm{dist}_{\mathcal{D}\oplus z}(g, h)$ is at least 0.4ϵ, in which case line 6 rejects with probability $1 - o_n(1)$, or $\mathrm{dist}_{\mathcal{D}\oplus z}(g, h) \leq 0.4\epsilon$ and thus, by triangular inequality, h is at least 0.6ϵ-far from decision lists with respect to $\mathcal{D} \oplus z$. Therefore, MonotoneDL* on line 7 rejects with probability $1 - o_n(1)$.

On the other hand, suppose that f is a decision list represented by (π, μ, ν) and $r, z \in \{0,1\}^n$ satisfy (5.1) and $\min_{\pi,\mu}(z) > \min_{\pi,\mu}(z)$. It is easy to verify that h is a monotone decision list. Also by (5.1), $\mathrm{dist}_{\mathcal{D}\oplus z}(g, h) \leq 0.1\epsilon$ so line 6 continues with probability at least $1 - o_n(1)$. Finally, line 7 accepts with probability at least $1 - o_n(1)$ given that h is a monotone decision list. □

Given TestDL, the challenge for the case when f is a decision list is to find a $z \in \{0,1\}^n$ with $\min_{\pi,\mu}(z) > \min_{\pi,\mu}(r)$. This is done in CheckDL (Algorithm 21), where the deterministic binary search subroutine called IndexSearch (Algorithm 20) will play an important role:

Algorithm 19 **TestDL**$(f, \mathcal{D}, \epsilon, r, z)$

Input: Oracle access to f, sampling access to $\mathcal{D}$, $\epsilon > 0$ and $r, z \in \{0,1\}^n$

Let $b = f(r)$, g be the function $g(x) = g(x \oplus z)$ and h be the function defined as follows:

$$h(x) = \begin{cases} b & \text{if } g(x) = b \\ \overline{b} & \text{if } g(x) = \overline{b} \text{ and } x \succ_g r \oplus z \\ b & \text{if } g(x) = \overline{b} \text{ and } r \oplus z \succ_g x \end{cases}$$

Set a counter $c \leftarrow 0$

for $(10 \log n)/\epsilon$ rounds **do**

    Draw $x \sim \mathcal{D} \oplus z$ and increment c if $g(x) \neq h(x)$

end for

reject if $c \geq (2 \log n)/\epsilon$ and continue otherwise

Run MonotoneDL$^*(h, \mathcal{D} \oplus z, \epsilon/2)$; **accept** if it accepts; **reject** if it rejects

Algorithm 20 **IndexSearch**(f, r, y)

Input: Oracle access to f and $r, y \in \{0,1\}^n$ with $f(r) \neq f(y)$

1: Let $b = f(r)$ and $g(x) := f(x \oplus r)$; below we write $g(T)$ to denote $g(x)$ where $x_i = 1$ iff $i \in T$

2: Run deterministic binary search to look for an $i \in \text{supp}(y \oplus r)$ such that $g(e_i) = \overline{b}$

3: **return** this i if found and continue if the binary search fails

4: Note that the binary search can fail if in one round, both branches evaluate to b in g. In more details, let $T_0, T_1, \ldots, T_p$ be the sequence of subsets of $\text{supp}(y \oplus r)$ followed by the binary search such that $T_0 = \text{supp}(y \oplus r)$, $T_{i+1} \subset T_i$ with $|T_{i+1}| \leq \lceil |T_i|/2 \rceil$ and $g(T_0) = \cdots = g(T_p) = \overline{b}$ but both subsets that T_p splits into evaluate to b in g so the binary search fails

5: **if** at least one of $g(T_0 \setminus T_1), \ldots, g(T_{p-1} \setminus T_p)$ is $\overline{b}$, say $g(T^*) = \overline{b}$ **then**

6:     Run binary search on T^* to look for an $i \in T^*$ such that $g(e_i) = \overline{b}$

7:     **return** this i if found; **return** nil if binary search fails again

8: **else**

9:     **return** nil

10: **end if**

Lemma 5.2. IndexSearch(f, r, y) *is deterministic and uses* $O(\log n)$ *queries on* f.

When f *is a decision list represented by* (π, μ, ν) *and* r, y *satisfy* $f(r) \neq f(y)$ *and* $\min_{\pi,\mu}(r) > \min_{\pi,\mu}(y)$, *it always returns an* i *with* $\pi^{-1}(i) \leq \min_{\pi,\mu}(r)$ *such that* $f(r^{(i)}) \neq f(r)$.

When f *is a decision list represented by* (π, μ, ν) *and* r, y *satisfy* $f(r) \neq f(y)$ *and* $\min_{\pi,\mu}(r) < \min_{\pi,\mu}(y)$, *it returns either* $\pi(\min_{\pi,\mu}(r))$ *or* nil.

Proof. The query complexity is trivial. The case when $\min_{\pi,\mu}(r) < \min_{\pi,\mu}(y)$ is also trivial because the algorithm always returns either nil or some $i \in \text{supp}(y \oplus r)$ such that $\overline{b} = g(e_i) = f(r^{(i)})$. But given the assumption that $\min_{\pi,\mu}(r) < \min_{\pi,\mu}(y)$, it must be the case that $i = \min_{\pi,\mu}(r)$.

Next consider the case when $\min_{\pi,\mu}(r) > \min_{\pi,\mu}(y)$, and let $i^* = \pi(\min_{\pi,\mu}(r))$. If an i is found by the first binary search on line 3, we are done as any i with $f(r^{(i)}) \neq f(r)$ has $\pi^{-1}(i) \leq \min_{\pi,\mu}(r)$. Otherwise, the first binary search fails and this can happen only when $i^* \in T_p$ and $\pi(\min_{\pi,\mu}(y)) \notin T_p$. As a result, the T^* we look for on line 5 must exist and it satisfies that $i^* \notin T^*$. Therefore, the second binary search always finds an i such that $f(r^{(i)}) = \overline{b} \neq f(r)$. ☐

We are now ready to prove Theorem 1.1:

Algorithm 21 CheckDL$(f, \mathcal{D}, \epsilon, r)$

Input: Oracle access to f, sampling access to $\mathcal{D}$, $\epsilon > 0$ and $r \in \{0,1\}^n$

1: Run MonotoneDL$^*(g, \mathcal{D}, \epsilon)$ with $g(x) := f(x \oplus r)$ and **accept** if it accepts
2: Otherwise, let $b = f(r)$ and S be the set of strings queried by MonotoneDL* on line 1
3: Run Sketch(g, S) but stop on line 9 to obtain sequence $X = (x^{(1)}, \ldots, x^{(m)})$ and $I_1, \ldots, I_k$
4: Let x^* denote the last $\overline{b}$-string of g in X
5: Run TestDL$(f, \mathcal{D}, \epsilon, r, x^* \oplus r)$; **accept** if it accepts and continue otherwise
6: Let A (or B) denote the last interval of $\overline{b}$-strings (or b-strings) of g in X
7: Run IndexSearch$(f, r, x \oplus r)$ for every $x \in A$
8: **if** IndexSearch$(f, r, z \oplus r) = \mathsf{nil}$ for some $z \in A$ **then**
9:     Pick any such $z \in A$
10:     Run TestDL$(f, \mathcal{D}, \epsilon, r, z \oplus r)$; **accept** if it accepts and **reject** if it rejects
11: **else if** IndexSearch$(f, r, x \oplus r) = i$ for some $x \in A$ such that $y_i = 1$ for some $y \in B$ **then**
12:     Pick any such i and run TestDL$(f, \mathcal{D}, \epsilon, r, r^{(i)})$; **accept** if it accepts; **reject** if it rejects
13: **else**
14:     **reject**
15: **end if**

Proof of Theorem 1.1. The correctness of DecisionList for the case when f is ϵ-far from decision lists with respect to $\mathcal{D}$ follows from the following claim:

Claim 1. *Suppose f is ϵ-far from decision lists with respect to $\mathcal{D}$. Then for any string $r \in \{0,1\}^n$, CheckDL$(f, \mathcal{D}, \epsilon, r)$ rejects with probability at least $1 - o_n(1)$.*

By a Chernoff bound followed by a union bound over the $100/\epsilon$ rounds, DecisionList accepts in this case with probability at most

$$\frac{100}{\epsilon} \cdot \exp(-\Omega(\log(n/\epsilon))) < 1/3.$$

Proof of Claim 1. Given that f is ϵ-far from decision lists with respect to $\mathcal{D}$, we have that $g(x) := f(x \oplus r)$ is also ϵ-far from decision lists and in particular, ϵ-far from monotone decision lists. As a result, line 1 of CheckDL accepts with probability $o_n(1)$.

Other than line 1, CheckDL accepts when one of the three executions of TestDL accepts but by Lemma 5.1, this also happens with probability $o_n(1)$. This finishes the proof of the claim. $\square$

The correctness of the algorithm for the case when f is a decision list follows from the following claim about CheckDL:

Claim 2. *Suppose f is a decision list represented by (π, μ, ν), and $r \in \{0,1\}^n$ satisfies (5.1), Then CheckDL$(f, \mathcal{D}, \epsilon, r)$ accepts with probability at least* 0.1.

For this case, it follows by Claim 2 that DecisionList accepts with probability at least

$$0.9 \cdot \big(1 - \exp(-\Omega(\log(n/\epsilon)))\big) > 2/3.$$

Proof of Claim 2. Let (π, μ, ν) be a representation of the input decision list f and let $b = f(r)$ and $j^* = \min_{\pi,\mu}(r) \in [n+1]$. Let g be the function with $g(x) := f(x \oplus r)$. Then g is also a decision list and can be represented by (π, μ', ν) with $\mu' = \mu \oplus r$. Let g^* be the function defined as

$$g^*(x) := \begin{cases} b & \text{if } g(x) = b \\ \overline{b} & \text{if } g(x) = \overline{b} \text{ and } \min_{\pi,\mu'}(x) < j^* \\ b & \text{if } g(x) = \overline{b} \text{ and } \min_{\pi,\mu'}(x) > j^* \end{cases}$$

Note that these three cases cover all possible x since $\min_{\pi,\mu'}(x) = j^*$ implies that $g(x) = b$. Let $g^\dagger$ be the function defined as

$$g^\dagger(x) := \begin{cases} \overline{b} & \text{if } g(x) = \overline{b} \\ b & \text{if } g(x) = b \text{ and } \min_{\pi,\mu'}(x) \leq j^* \\ \overline{b} & \text{if } g(x) = b \text{ and } \min_{\pi,\mu'}(x) > j^* \end{cases}$$

We show that g^* is a monotone decision list. To see this is the case, it is easy to verify that g^* can be represented as (π, μ', ν^*) with $\nu_j^* = b$ for all $j \geq j^*$. This can be equivalently represented as $(\pi, 1^n, \nu^*)$, a monotone decision list (basically every x with $\min_{\pi,\mu'}(x) \geq j^*$ always gets $g^*(x) = b$). It is also easy to verify that $f^\dagger$ is a monotone decision list as well.

Consider $\textsc{MonotoneDL}^*(g, \mathcal{D}, \epsilon)$ on line 2. Let $\mathcal{S}$ denote the distribution over $N_{n,\epsilon}$-subsets of $\{0,1\}^n$ as the (random) set of queries made by $\textsc{MonotoneDL}^*(g, \mathcal{D}, \epsilon)$. We show that either

$$\Pr\Big[\textsc{MonotoneDL}^*(g, \mathcal{D}, \epsilon) \text{ accepts}\Big]$$

is at least 0.2, or

$$\Pr_{S \sim \mathcal{S}}\left[S \text{ has an } x \text{ with } g(x) = \overline{b} \text{ and } \min_{\pi,\mu'}(x) > j^*, \text{ and a } y \text{ with } g(y) = b \text{ and } \min_{\pi,\mu'}(y) > j^*\right]$$

is at least 0.2. This is because, if the second probability is at least 0.2 then we are done. Otherwise either with probability at least 0.4, $\textsc{MonotoneDL}^*(g, \mathcal{D}, \epsilon)$ behaves exactly the same as if it runs on $(g^*, \mathcal{D})$ and it follows that the first probability is at least $0.4 - o_n(1)$ given that g^* is a monotone decision list, or with probability at least 0.4, $\textsc{MonotoneDL}^*(g, \mathcal{D}, \epsilon)$ behaves exactly the same as if it runs on $(g^\dagger, \mathcal{D})$ and thus, the first probability is at least $0.4 - o_n(1)$ as well.

It suffices to show that whenever S contains an x with $g(x) = \overline{b}$ such that $\min_{\pi,\mu'}(x) > j^*$ and a y with $g(y) = b$ such that $\min_{\pi,\mu'}(y) > j^*$, the rest of the procedure (lines 3–15 of $\textsc{CheckDL}$) accepts with probability at least $1 - o_n(1)$.

First, if x^* on line 4 satisfies $\min_{\pi,\mu'}(x^*) > \min_{\pi,\mu'}(r)$ then we are done because the $\textsc{TestDL}$ on line 5 accepts with probability $1 - o_n(1)$. So we assume below that $\min_{\pi,\mu'}(x^*) < \min_{\pi,\mu'}(r)$ (they cannot be equal because x^* is a $\overline{b}$-string of g).

From this we can infer that A must contain a string $x^\dagger$ such that $g(x^\dagger) = \overline{b}$ and $\min_{\pi,\mu'}(x^\dagger) > j^*$ and B must contain a string $y^\dagger$ such that $g(y^\dagger) = b$ and $\min_{\pi,\mu'}(y^\dagger) > j^*$. To see this is the case, we just use the following property of the sorted sequence X together with the intervals $I_1, \ldots, I_k$: If x is any string in I_t for some t such that $\min_{\pi,\mu'}(x) < j^*$ (such as the x^* above), then all strings z in $I_1, \ldots, I_{t-1}$ (i.e., intervals before I_t) must satisfy $\min_{\pi,\mu'}(z) < j^*$ as well (though this does not apply to strings in the same interval I_t as x.)

Now if $\textsc{CheckDL}$ enters line 9, z must satisfy $\min_{\pi,\mu'}(z) > j^*$ by Lemma 5.2 and thus, we are done because $\textsc{TestDL}$ on line 9 accepts with probability $1 - o_n(1)$. Otherwise we must have that $\textsc{IndexSearch}(f, r, x^\dagger \oplus r)$ returns $i^* = \pi(j^*)$ by Lemma 5.2. And checking among all indices returned by $\textsc{IndexSearch}(f, r, x \oplus r)$, $x \in A$, i^* is the only index that can have $y_{i^*} = 1$ for some $y \in B$ (using that $y^\dagger \in B$ and that all strings $y \in B$ satisfy $\min_{\pi,\mu'}(y) \geq j^*$ because $x^\dagger \in A$). As a result, $\textsc{TestDL}$ must enter line 12 with $i = i^* = \pi(j^*)$ and thus, $\min_{\pi,\mu}(r^{(i)}) > \min_{\pi,\mu}(r)$. So we are done because $\textsc{TestDL}$ on line 12 accepts with probability $1 - o_n(1)$. □

It suffices to upperbound the number of queries made by $\textsc{DecisionList}$, which is at most

$$O\left(\frac{1}{\epsilon}\right) \cdot O\left(\log\left(\frac{n}{\epsilon}\right)\right) \cdot \tilde{O}\left(N_{n,\epsilon/2}\right) = \tilde{O}\left(\frac{n^{11/12}}{\epsilon^3}\right).$$

This finishes the proof of Theorem 1.1. □

6 An $\Omega(\sqrt{n})$ Lower Bound

We proceed by Yao's principle and construct two distributions $\mathcal{D}_{YES}$ and $\mathcal{D}_{NO}$ over function-distribution pairs $(f, \mathcal{D})$. For simplicity, we will assume that n is a multiple of 16.

- $\mathcal{D}_{NO}$: For each permutation π over $[n]$, we define the distribution $\mathcal{D}_\pi^{\mathsf{no}}$ to be the uniform distribution over

  $$\bigcup_{k=\frac{n}{8}}^{\frac{n}{4}-1} \left\{e_{\pi(4k+1)} \vee e_{\pi(4k+2)},\ e_{\pi(4k+2)} \vee e_{\pi(4k+3)},\ e_{\pi(4k+3)} \vee e_{\pi(4k+4)},\ e_{\pi(4k+4)} \vee e_{\pi(4k+1)}\right\}.$$

  We then define $\nu^{\mathsf{no}} \in \{0,1\}^{n+1}$ by setting

  $$\nu^{\mathsf{no}}_{4k+1} = \nu^{\mathsf{no}}_{4k+3} = 1,\ \nu^{\mathsf{no}}_{4k+2} = \nu^{\mathsf{no}}_{4k+4} = 0$$

  for each $k \in \{\frac{n}{8}, \frac{n}{8}+1, \ldots, \frac{n}{4}-1\}$, and $\nu^{\mathsf{no}}_i = 1$ for each $i \in \{1, 2, \ldots, \frac{n}{4}-1\} \cup \{n+1\}$. Now suppose $x \in \{0,1\}^n$. Let i be the smallest number in the set $\pi^{-1}(\mathrm{supp}(x))$. If $i = 4k+1$ for some $k \in \{\frac{n}{8}, \frac{n}{8}+1, \ldots, \frac{n}{4}-1\}$ and

  $$\pi^{-1}(\mathrm{supp}(x)) \cap \{4k+1, 4k+2, 4k+3, 4k+4\} = \{4k+1, 4k+4\},$$

  then set $f^{\mathsf{no}}_\pi(x) = 0$. Otherwise, set $f^{\mathsf{no}}_\pi(x)$ so that it agree with the monotone decision list represented by the pair (π, ν^{no}). Note that we have

  $$f^{\mathsf{no}}_\pi(e_{\pi(4k+4)} \vee e_{\pi(4k+1)}) = f^{\mathsf{no}}_\pi(e_{\pi(4k+2)} \vee e_{\pi(4k+3)}) = 1 \tag{6.1}$$

  and

  $$f^{\mathsf{no}}_\pi(e_{\pi(4k+1)} \vee e_{\pi(4k+2)}) = f^{\mathsf{no}}_\pi(e_{\pi(4k+3)} \vee e_{\pi(4k+4)}) = 0 \tag{6.2}$$

  for each $k \in \{\frac{n}{8}, \frac{n}{8}+1, \ldots, \frac{n}{4}-1\}$. The final distribution $\mathcal{D}_{NO}$ is taken to be the distribution of the random pair $(f^{\mathsf{no}}_\pi, \mathcal{D}^{\mathsf{no}}_\pi)$, where π is a uniformly random permutation over $[n]$.

- $\mathcal{D}_{YES}$: For each permutation π over $[n]$, we define the distribution $\mathcal{D}_\pi^{\mathsf{yes}}$ to be the uniform distribution over

  $$\bigcup_{k=\frac{n}{8}}^{\frac{n}{4}-1} \left\{e_{\pi(4k+1)} \vee e_{\pi(4k+2)},\ e_{\pi(4k+1)} \vee e_{\pi(4k+3)},\ e_{\pi(4k+2)} \vee e_{\pi(4k+4)},\ e_{\pi(4k+3)} \vee e_{\pi(4k+4)}\right\}.$$

  We then define $\nu^{\mathsf{yes}} \in \{0,1\}^{n+1}$ by setting

  $$\nu^{\mathsf{yes}}_{4k+1} = \nu^{\mathsf{yes}}_{4k+4} = 1,\ \nu^{\mathsf{yes}}_{4k+2} = \nu^{\mathsf{yes}}_{4k+3} = 0$$

  for each $k \in \{\frac{n}{8}, \frac{n}{8}+1, \ldots, \frac{n}{4}-1\}$, and $\nu^{\mathsf{no}}_i = 1$ for each $i \in \{1, 2, \ldots, \frac{n}{4}-1\} \cup \{n+1\}$. We then let $f^{\mathsf{yes}}_\pi(x)$ be the monotone decision list represented by the pair $(\pi, \nu^{\mathsf{yes}})$. The final distribution $\mathcal{D}_{YES}$ is taken to be the distribution of the random pair $(f^{\mathsf{yes}}_\pi, \mathcal{D}^{\mathsf{yes}}_\pi)$, where π is a uniformly random permutation over $[n]$.

By definition, we have that any function drawn from $\mathcal{D}_{YES}$ will be a decision list. Correspondingly, functions from $\mathcal{D}_{NO}$ will be far from any decision list.

Lemma 6.1. *For any permutation π over $[n]$ and for any linear threshold function $g : \{0,1\}^n \to \{0,1\}$, we have that*

$$\Pr_{x \sim \mathcal{D}^{\mathsf{no}}_{\pi}}[g(x) \neq f^{\mathsf{no}}_{\pi}(x)] \geq \frac{1}{4}.$$

Proof. By (6.1) and (6.2), it suffices to show that for each $k \in \{\frac{n}{8}, \frac{n}{8}+1, \ldots, \frac{n}{4}-1\}$, we cannot have

$$g(e_{\pi(4k+4)} \vee e_{\pi(4k+1)}) = g(e_{\pi(4k+2)} \vee e_{\pi(4k+3)}) = 1 \tag{6.3}$$

and

$$g(e_{\pi(4k+1)} \vee e_{\pi(4k+2)}) = g(e_{\pi(4k+3)} \vee e_{\pi(4k+4)}) = 0 \tag{6.4}$$

simultaneously. Assume on the contrary that both (6.3) and (6.4) hold for some $k \in \{\frac{n}{8}, \frac{n}{8} + 1, \ldots, \frac{n}{4} - 1\}$. Since g is an linear threshold function, g agrees with a halfspace function $h : \mathbb{R}^n \to \{0,1\}$. Let x_0 be the vector in $\mathbb{R}^n$ defined by

$$x_0 = \frac{1}{2}\left(e_{\pi(4k+1)} + e_{\pi(4k+2)} + e_{\pi(4k+3)} + e_{\pi(4k+4)}\right).$$

By convexity of halfspaces, (6.3) implies $h(x_0) = 1$ and (6.4) implies $h(x_0) = 0$, a contradiction. □

We now claim that in order to distinguish between these two distributions an algorithm must be able to query an edge between two vertices in a cycle or sample two edges from the same cycle. To make this more formal, we say that ALG sees an element i if it makes a query $y \in \{0,1\}^n$ with $y_i = 1$.

Lemma 6.2. *Suppose that ALG satisfies*

$$\Pr_{\mathcal{D}_{\boldsymbol{\pi}}, f_{\boldsymbol{\pi}} \sim \mathcal{D}_{NO}, \mathbf{s}_1, \ldots, \mathbf{s}_m \sim \mathcal{D}_{\boldsymbol{\pi}}}[\exists i,j : \mathbf{s}_i, \mathbf{s}_j \textit{ in same cycle or } \mathsf{ALG}(f_{\boldsymbol{\pi}}, \mathbf{s}_1, \ldots, \mathbf{s}_m) \textit{ sees a non-sampled pair in a cycle}] < 1/3$$

$$\Pr_{\mathcal{D}_{\boldsymbol{\pi}}, g_{\boldsymbol{\pi}} \sim \mathcal{D}_{YES}, \mathbf{s}_1, \ldots, \mathbf{s}_m \sim \mathcal{D}_{\boldsymbol{\pi}}}[\exists i,j : \mathbf{s}_i, \mathbf{s}_j \textit{ in same cycle or } \mathsf{ALG}(g_{\boldsymbol{\pi}}, \mathbf{s}_1, \ldots, \mathbf{s}_m) \textit{ sees a non-sampled pair in a cycle}] < 1/3$$

then ALG cannot distinguish between $\mathcal{D}_{YES}$ and $\mathcal{D}_{NO}$ with probability greater than $2/3$.

Proof. Now fix some set of samples $s_1, \ldots, s_m \in \{0,1\}^n$ and a set of queries $q_1, \ldots, q_M \in \{0,1\}^n$. Let $E_{s,q}(\pi)$ denote the event that (i) there exists two samples s_i, s_j in the same cycle, (ii) $\bigvee_{i=1}^M q_i \backslash \bigvee_{i=1}^m s_i$ contains two vertices from a cycle, or (iii) $\bigvee_{i=1}^M q_i \vee \bigvee_{i=1}^m s_i$ contains three vertices from some cycle. Note that this is precisely the complement of the event whose probability is bounded in the assumption of the lemma.

We will now show that for any boolean values v_i and w_j

$$\Pr_{\boldsymbol{\pi}}[f_{\boldsymbol{\pi}}(s_i) = v_i, f_{\boldsymbol{\pi}}(q_j) = w_j \quad \forall i,j | E_{s,q}(\boldsymbol{\pi})] = \Pr_{\boldsymbol{\pi}}[g_{\boldsymbol{\pi}}(s_i) = v_i, g(q_j) = w_j \quad \forall i,j | E_{s,q}(\boldsymbol{\pi})]$$

We prove the statement via a bijection between permutations. Let π_{YES} be a permutation such that $E_{s,q}(\pi_{YES})$ holds and $g_{\boldsymbol{\pi}_{YES}}(s_i) = v_i, g_{\boldsymbol{\pi}_{YES}}(q_j) = w_j$ for all i and j. We consider a permutation π_{NO} which is made by taking π_{YES} and permuting the order of elements $\pi_{YES}(4k), \pi_{YES}(4k+1), .., .\pi_{YES}(4k+2), \pi_{YES}(4k+3)$: If no samples are from the cycle or the sample s_i using a particular cycle is incident to $x_{\pi_{YES}(4k+2)}$ then we do nothing. Otherwise, we swap $\pi_{YES}(4k+1)$ and $\pi_{YES}(4k+3)$ in the ordering to get π_{NO}.

Now note that the cycles in $\mathcal{D}_{\pi_{NO}}$ are the same as those in $\mathcal{D}_{\pi_{YES}}$, so we have that $E_{s,q}(\pi_{NO})$ holds. Now note that under this coupling it follows that $f_{\pi_{NO}}(s_i) = g_{\pi_{YES}}(s_i) = v_i$. We also have that $f_{\pi_{NO}}(x_i) = g_{\pi_{YES}}(x_i)$ for all i. We now claim that

$$f_{\pi_{YES}}\left(\bigvee_{i\in S} e_i\right) = f_{\pi_{NO}}\left(\bigvee_{i\in S} e_i\right)$$

so long as S doesn't contain three vertices from a cycle or a non-sampled pair of vertices from a cycle. Suppose that the S contains sampled edges $s_1, ..., s_k$. If the rule that fires in $g_{\pi_{YES}}$ is not in $\vee_i s_i$ then this will also be the rule that fires in $f_{\pi_{NO}}$ since the relative ordering under π only changes for elements in the same cycle and no other vertices in S are in the same cycle as i. So now assume without loss of generality that $s_1 = e_a \vee e_b$ and the rule for x_a fires. It then follows that

$$g_{\pi_{YES}}\left(\bigvee_{i\in S} e_i\right) = g_{\pi_{YES}}(e_a) = g_{\pi_{YES}}(s_1) = f_{\pi_{NO}}(s_1) = f_{\pi_{NO}}\left(\bigvee_{i\in S} e_i\right).$$

So it now follows that $f_{\pi_{NO}}(q_i) = g_{\pi_{YES}}(q_i)$ for all i. Since our mapping is clearly a bijection, we have that

$$\Pr_{\boldsymbol{\pi}}[f_{\boldsymbol{\pi}}(s_i) = v_i, f_{\boldsymbol{\pi}}(q_j) = w_j \quad \forall i,j | E_{s,q}(\boldsymbol{\pi})] = \Pr_{\boldsymbol{\pi}}[g_{\boldsymbol{\pi}}(s_i) = v_i, g(q_j) = w_j \quad \forall i,j | E_{s,q}(\boldsymbol{\pi})]$$

as desired. Now fix a path p in the decision tree of ALG and let $Q(p)$ denote the queries made by p. Note that

$$\Pr_{\boldsymbol{\pi},\mathbf{s}_1,...,\mathbf{s}_m\sim\mathcal{D}_{\boldsymbol{\pi}}}[\mathsf{ALG}(\mathbf{f}, \mathbf{s}_1, ..., \mathbf{s}_m),\text{ follows } p \wedge E_{\mathbf{s},Q(p)}] = \Pr_{\boldsymbol{\pi},\mathbf{s}_1,...,\mathbf{s}_m\sim\mathcal{D}_{\boldsymbol{\pi}}}[\mathsf{ALG}(\mathbf{g}, \mathbf{s}_1, ..., \mathbf{s}_m)\text{ follows } p \wedge E_{\mathbf{s},Q(p)}]$$

Thus,

$$\Pr_{\boldsymbol{\pi},\mathbf{s}_1,...,\mathbf{s}_m\sim\mathcal{D}_{\boldsymbol{\pi}}}[\mathsf{ALG}(\mathbf{f}, \mathbf{s}_1, ..., \mathbf{s}_m),\text{ accepts } \wedge E_{\mathbf{s},Q(\mathsf{ALG})}] = \Pr_{\boldsymbol{\pi},\mathbf{s}_1,...,\mathbf{s}_m\sim\mathcal{D}_{\boldsymbol{\pi}}}[\mathsf{ALG}(\mathbf{g}, \mathbf{s}_1, ..., \mathbf{s}_m)\text{ accepts } \wedge E_{\mathbf{s},Q(\mathsf{ALG})}].$$

But now note that $\Pr_{\boldsymbol{\pi},\mathbf{s}_1,...,\mathbf{s}_m\sim\mathcal{D}_{\boldsymbol{\pi}}}[\mathsf{ALG}(\mathbf{g}, \mathbf{s}_1, ..., \mathbf{s}_m)\text{ accepts } \wedge E_{\mathbf{s},Q(\mathsf{ALG})}]$ is equal to

$$\begin{aligned}
&= \Pr_{\boldsymbol{\pi},\mathbf{s}_1,...,\mathbf{s}_m\sim\mathcal{D}_{\boldsymbol{\pi}}}[\mathsf{ALG}(\mathbf{g}, \mathbf{s}_1, ..., \mathbf{s}_m)\text{ accepts } \wedge E_{\mathbf{s},Q(\mathsf{ALG})}] + \Pr_{\boldsymbol{\pi},\mathbf{s}_1,...,\mathbf{s}_m\sim\mathcal{D}_{\boldsymbol{\pi}}}[\mathsf{ALG}(\mathbf{g}, \mathbf{s}_1, ..., \mathbf{s}_m)\text{ accepts } \wedge \neg E_{\mathbf{s},Q(\mathsf{ALG})}] \\
&< \Pr_{\boldsymbol{\pi},\mathbf{s}_1,...,\mathbf{s}_m\sim\mathcal{D}_{\boldsymbol{\pi}}}[\mathsf{ALG}(\mathbf{g}, \mathbf{s}_1, ..., \mathbf{s}_m)\text{ accepts } \wedge E_{\mathbf{s},Q(\mathsf{ALG})}] + \frac{1}{3} \\
&= \Pr_{\boldsymbol{\pi},\mathbf{s}_1,...,\mathbf{s}_m\sim\mathcal{D}_{\boldsymbol{\pi}}}[\mathsf{ALG}(\mathbf{f}, \mathbf{s}_1, ..., \mathbf{s}_m),\text{ accepts } \wedge E_{\mathbf{s},Q(\mathsf{ALG})}] + \frac{1}{3} \\
&\le \frac{1}{3} + \frac{1}{3}.
\end{aligned}$$

□

Currently, this lemma is too weak to use as an algorithm can simply query $\bigvee_{i\in[n]} e_i$ and see everything. To circumvent this, we'll show that we can assume the algorithm never makes large queries with many unseen vertices.

Lemma 6.3. *Suppose that ALG is an algorithm that distinguishes with probability $5/6$ between $\mathcal{D}_{YES}$ and $\mathcal{D}_{NO}$ with at most $\frac{n}{100}$ samples and queries. It then follows that there is an algorithm ALG' that distinguishes between $\mathcal{D}_{YES}$ and $\mathcal{D}_{NO}$ with probability $4/5$ such that ALG' never queries more than $O(\log(n))$ unknown vertices at a time. Moreover, ALG' uses at at most as many samples and queries as ALG.*

Proof. Since ALG is a deterministic algorithm, it corresponds to a decision tree. Consider the following transformation: Let $\mathsf{ALG}_0 := \mathsf{ALG}$. To construct ALG_{i+1}, find a maximal set $\mathcal{Q}$ of queries $S \subseteq [n]$ in ALG_i such that (1) S contains more than $100\log(n)$ unseen vertices and (2) no set S has an ancestor with property (1). We get ALG_{i+1} by taking the decision tree for ALG and assuming the answer to these queries is 1.

By the bound on the sample and query complexity, ALG has depth at most $\frac{n}{100}$. We claim that $\mathsf{ALG}_{\frac{n}{100}}$ will make no queries with over $100\log(n)$ unseen vertices. Indeed, suppose that S is query in ALG_k then we claim that S must occur at depth at most $\frac{n}{100} - k$ in ALG_k. We prove the claim by induction. The base case of $k = 0$ holds trivially. Now suppose the statement holds for k. We remove a maximal set of bad queries $\mathcal{Q}$ to get ALG_{k+1}. Note that this means that any bad query that isn't removed must have it's depth decreased by one (as it must have an ancestor in $\mathcal{Q}$). This implies that after $\frac{n}{100}$ steps there are no bad queries.

Now we claim that

$$\Pr_{\boldsymbol{\pi}, \mathbf{s} \sim \mathcal{D}_{\boldsymbol{\pi}}}[\mathsf{ALG}_k(g_{\boldsymbol{\pi}}, \mathbf{s}) \text{ accepts}] \geq \Pr_{\boldsymbol{\pi}, \mathbf{s} \sim \mathcal{D}_{\boldsymbol{\pi}}}[\mathsf{ALG}_{k-1}(g_{\boldsymbol{\pi}}, \mathbf{s}) \text{ accepts}] - n^{-10}.$$

Call a path p in ALG_{k-1} bad if the first time it queries a vertex with more than $O(\log(n))$ vertices it assumes the answer is zero and good otherwise. Now fix an accepting path p in ALG_{k-1}. If p is good then there is a corresponding path p' in ALG_k such that

$$\Pr_{\boldsymbol{\pi}, \mathbf{s} \sim \mathcal{D}_{\boldsymbol{\pi}}}[\mathsf{ALG}_k(g_{\boldsymbol{\pi}}, \mathbf{s}) \text{ follows } p'] \geq \Pr_{\boldsymbol{\pi}, \mathbf{s} \sim \mathcal{D}_{\boldsymbol{\pi}}}[\mathsf{ALG}_{k-1}(g_{\boldsymbol{\pi}}, \mathbf{s}) \text{ follows } p]$$

Now fix a set $S \in \mathcal{Q}$. We will bound the mass of all paths from ALG_{k-1} that assume the query on S outputs 1. Indeed, let p_S be the path leading to S. Note that conditioned on the at most $\frac{n}{100}$ vertices we've seen so far in p, any unseen vertex has at least a 0.49 chance of being in $x_{\boldsymbol{\pi}(n-1)}, x_{\boldsymbol{\pi}(n-2)}, \ldots x_{\boldsymbol{\pi}(n/2)}$. So,

$$\Pr_{\boldsymbol{\pi}, \mathbf{s} \sim \mathcal{D}_{\boldsymbol{\pi}}}[g_{\boldsymbol{\pi}}(S) = 0 | \mathsf{ALG}(g_{\boldsymbol{\pi}}, \mathbf{s}) \text{ follows } p_S] \leq .51^{100\log(n)} \leq n^{-10}$$

It now follows that

$$\sum_{p: p \text{ bad}} \Pr_{\boldsymbol{\pi}, \mathbf{s} \sim \mathcal{D}_{\boldsymbol{\pi}}}[\mathsf{ALG}(g_{\boldsymbol{\pi}}, \mathbf{s}) \text{ follows } p] \leq \sum_{S \in \mathcal{Q}} \Pr_{\boldsymbol{\pi}, \mathbf{s} \sim \mathcal{D}_{\boldsymbol{\pi}}}[g_{\boldsymbol{\pi}}(S) = 0 | \mathsf{ALG}(g_{\boldsymbol{\pi}}, \mathbf{s}) \text{ follows } p_S] \cdot \Pr_{\boldsymbol{\pi}, \mathbf{s} \sim \mathcal{D}_{\boldsymbol{\pi}}}[\mathsf{ALG}(g_{\boldsymbol{\pi}}, \mathbf{s}) \text{ follows } p_S]$$

which is at most n^{-10}. By a similar argument we have that

$$\Pr_{\boldsymbol{\pi}, \mathbf{s} \sim \mathcal{D}_{\boldsymbol{\pi}}}[\mathsf{ALG}_k(f_{\boldsymbol{\pi}}, \mathbf{s}) \text{ rejects}] \geq \Pr_{\boldsymbol{\pi}, \mathbf{s} \sim \mathcal{D}_{\boldsymbol{\pi}}}[\mathsf{ALG}_{k-1}(f_{\boldsymbol{\pi}}, \mathbf{s}) \text{ rejects}] - n^{-10}.$$

□

Theorem 6.4. *Any distribution-free tester for decision lists must make $\tilde{\Omega}(\sqrt{n})$ queries + samples.*

Proof. We argue that any algorithm that makes fewer than $\frac{\sqrt{n}}{100000\log(n)}$ queries and samples cannot see a non-sampled pair in a cycle with good probability. By Lemma 6.3, we assume our algorithm never queries more than $100\log(n)$ unseen vertices at a time.

Start by observing that the set of seen vertices will have size at most $\sqrt{n}/1000$. Moreover, if we haven't seen a vertex, we note that it is distributed uniformly. We can then lower bound the probability that we never query an non-sampled pair of vertices by

$$\left(1 - \frac{4|S|}{n - |S|}\right)^{\sqrt{n}/1000} \geq \left(1 - \frac{1}{25\sqrt{n}}\right)^{\sqrt{n}/1000} \geq \frac{1}{\sqrt[40]{e}} > 9/10.$$

On the other hand, by a birthday paradox argument, the probability that we sample two edges from a cycle is at most $o_n(1)$ if we draw at most $\frac{\sqrt{n}}{100000\log(n)}$ samples. The theorem now follows by Lemma 6.2. $\square$

7 Sample-Based Lower Bounds

Our proof of the sample-based lower bounds relies on the support-size-distinction framework of [BFH21].

Definition 7.1 (Support-size distinction, [BFH21])**.** Fix positive integer n and real numbers α, β with $0 < \alpha < \beta < 1$. Define $\mathsf{SSD}(n, \alpha, \beta)$ to be the minimum integer m such that there is a randomized algorithm $\mathcal{A}'$ that for any distribution $\mathcal{D}$ over $[n]$, takes m samples $x^{(1)}, \ldots, x^{(m)}$ from $\mathcal{D}$ and achieves the following:

- If $|\operatorname{supp}(\mathcal{D})| \le \alpha n$ and $\mathcal{D}(\{x\}) \ge 1/n$ for all $x \in \operatorname{supp}(\mathcal{D})$, then

$$\Pr_{\mathcal{A}',\, x^{(1)},\ldots,x^{(m)}}\left[\mathcal{A}'(x^{(1)}, \ldots, x^{(m)}) = 1\right] \ge 2/3.$$

- If $|\operatorname{supp}(\mathcal{D})| \ge \beta n$ and $\mathcal{D}(\{x\}) \ge 1/n$ for all $x \in \operatorname{supp}(\mathcal{D})$, then

$$\Pr_{\mathcal{A}',\, x^{(1)},\ldots,x^{(m)}}\left[\mathcal{A}'(x^{(1)}, \ldots, x^{(m)}) = 0\right] \ge 2/3.$$

The following lower bound on $\mathsf{SSD}(n, \alpha, \beta)$ essentially follows from [WY19] but only explicitly appears in [BFH21].

Lemma 7.2 ([WY19, BFH21])**.** *There exists a constant C such that, for any $\delta \ge C(\log n)^{1/2} n^{-1/4}$ and $\alpha, 1-\beta \ge \delta$,*

$$\mathsf{SSD}(n, \alpha, \beta) = \Omega\left(\frac{n\delta^2}{\log n}\right).$$

We also need the following corollary of the Sauer-Shelah lemma.

Lemma 7.3 (Lemma 2.7 of [BFH21])**.** *Let $\mathcal{X}$ be a finite set, and let $\mathcal{H}$ be a class of functions $\mathcal{X} \to \{0,1\}$ with $\mathsf{VC}(\mathcal{H}) = d$. If $T \subset \mathcal{X}$ has size $|T| \ge 4d$, then a uniformly random function $f : T \to \{0,1\}$ satisfies*

$$\Pr_f\left[\left|\{x \in T : f(x) \ne h(x)\}\right| \ge \frac{|T|}{100} \text{ for all } h \in \mathcal{H}\right] \ge 1 - e^{-d/10}.$$

We then state our main structural lemma.

Lemma 7.4. *Let $m = \frac{n}{10\log n}$, and let $y^{(1)}, y^{(2)}, \ldots, y^{(m)}$ be independent samples from the uniform distribution on $\{y \in \{0,1\}^n : \|y\|_1 = n - \log n\}$. Then the set $\{y^{(1)}, y^{(2)}, \ldots, y^{(m)}\}$ is shattered by $\mathcal{MCONJ}$ with probability at least $1 - O(n^{-1})$.*

Proof. For each $k \in [m]$, define the event

$$\Gamma_k := \left\{\operatorname{supp}(y^{(k)}) \supset \bigcap\nolimits_{j \in [m]\setminus\{k\}} \operatorname{supp}(y^{(j)})\right\}.$$

For each $k \in [m]$, the intersection $\bigcap_{j\in[m]\setminus\{k\}} \mathrm{supp}(y^{(j)})$ has cardinality at least $n-(m-1)\cdot\log n \geq \frac{9}{10}n$. Since $y^{(k)}$ is independent with the samples $\{y^{(j)}\}_{j\in[m]\setminus\{k\}}$, we know that

$$\Pr[\Gamma_k] = \frac{\left|\left\{y \in \{0,1\}^n : \|y\|_1 = n - \log n, \text{ and } \mathrm{supp}(y) \supset \bigcap_{j\in[m]\setminus\{k\}} \mathrm{supp}(y^{(j)})\right\}\right|}{\left|\left\{y \in \{0,1\}^n : \|y\|_1 = n - \log n\right\}\right|} \leq \frac{\binom{n/10}{\log n}}{\binom{n}{\log n}} \leq \frac{1}{10^{\log n}}.$$

It then follows from union bound that

$$\Pr\left[\bigcup_{k\in[m]} \Gamma_k\right] \leq \frac{m}{10^{\log n}} = O(n^{-1}).$$

Let $\Gamma = \bigcup_{k\in[m]} \Gamma_k$. It now suffices to show that, on the complement event Γ^c, the set $\{y^{(1)}, \ldots, y^{(m)}\}$ is always shattered by $\mathcal{MCONJ}$. Let $b^{(1)}, \ldots, b^{(m)} \in \{0,1\}$ be any sequence of bits, and whenver Γ^c holds we will define a monotone conjunction f such that $f(y^{(k)}) = b^{(k)}$ for each $k \in [m]$. Let

$$E = \bigcap_{j\in[m]:\, b^{(j)}=1} \mathrm{supp}(y^{(j)}),$$

and for $x \in \{0,1\}^n$, let $f(x) = \bigwedge_{i\in E} x_i$. For $k \in [m]$, it follows directly from the definition of f that if $b^{(k)} = 1$ then $f(y^{(k)}) = b^{(k)}$. If $b^{(k)} = 0$, then due to the condition of Γ^c we have

$$\mathrm{supp}(y^{(k)}) \not\supset \bigcap_{j\in[m]:\, b^{(j)}=1} \mathrm{supp}(y^{(j)}).$$

By the definition of f, this implies that $f(y^{(k)}) = 0 = b^{(k)}$. $\square$

Now we are ready to prove a reduction from sample-based testing of monotone conjunctions to the problem of support-size distinction.

Lemma 7.5. *Let $\varepsilon \leq 1/250$ be a fixed positive number. Assume that $\mathcal{A}$ is a randomized sample-based algorithm that for any function $f : \{0,1\}^n \to \{0,1\}$ and any distribution $\mathcal{D}$ over $\{0,1\}^n$, when given as input a sequence $S = \{(y^{(k)}, f(y^{(k)}))\}_{k\in[m]}$ such that $y^{(k)} \overset{i.i.d.}{\sim} \mathcal{D}$, achieves the following:*

- *If $f \in \mathcal{MCONJ}$ then $\Pr_{\mathcal{A},S}[\mathcal{A}(S) = 1] \geq 5/6$; and,*
- *If $\mathsf{dist}_{\mathcal{D}}(f, \mathcal{LTF}) \geq \varepsilon$ then $\Pr_{\mathcal{A},S}[\mathcal{A}(S) = 0] \geq 5/6$.*

Then we must have $m = \Omega(n/\log^3 n)$.

Proof. We will construct a support-size-distinguisher out of $\mathcal{A}$. Let $\mathcal{D}$ be a distribution over $[10n]$ with $\mathcal{D}(\{x\}) \geq (10n)^{-1}$ for each $x \in \mathrm{supp}(\mathcal{D})$. Consider the following algorithm:

We then show that Algorithm 22 can distinguish between the case where $\mathcal{D}$ has a small support-size and the case where it has a large support-size.

Small-support-size case. If $\mathcal{D}$ has support size at most $\frac{n}{10\log n}$, then by Lemma 7.4, the image $\varphi(\mathrm{supp}(\mathcal{D}))$ is shattered by $\mathcal{MCONJ}$ with probability at least $1 - O(n^{-1})$. So with probability at least $1 - O(n^{-1})$, the sequence S constructed in step 4 of Algorithm 22 is consistent with a monotone conjunction. Thus by assumption we have

$$\Pr_{\mathcal{A},\varphi,f,x^{(1)},\ldots,x^{(m)}}[\mathcal{A}(S) = 1] \geq \frac{5}{6}(1 - O(n^{-1})) \geq \frac{2}{3}.$$

Algorithm 22 Reduction to support-size distinction

Input: Sample access to $\mathcal{D}$ over $[10n]$

1: Draw samples $x^{(1)}, x^{(2)}, \ldots, x^{(m)} \in [10n]$.

2: Let φ be a uniformly random function from $[10n]$ to the set $\{y \in \{0,1\}^n : \|y\|_1 = n - \log n\}$.

3: Let f be a uniformly random function from $\{y \in \{0,1\}^n : \|y\|_1 = n - \log n\}$ to $\{0,1\}$.

4: Construct a sequence
$$S = \left\{ \left(\varphi(x^{(k)}), f\left(\varphi(x^{(k)})\right) \right) \right\}_{k \in [m]}.$$

5: **return** $\mathcal{A}(S)$.

Large-support-size case. If $\mathcal{D}$ has support size at least $5n$, then by Chernoff bound we have
$$\Pr_{\varphi}\left[\left|\varphi(\mathrm{supp}(\mathcal{D}))\right| \leq 4n\right] \leq e^{-n/10}. \tag{7.1}$$

For any fixed φ such that $\left|\varphi(\mathrm{supp}(\mathcal{D}))\right| \geq 4n$, using Lemma 7.3 with $\mathcal{X} := \{y \in \{0,1\}^n : \|y\|_1 = n - \log n\}$, $\mathcal{H} := \mathcal{LTF}$ and $T := \varphi(\mathrm{supp}(\mathcal{D}))$, we have
$$\Pr_{f}\left[\left|\{x \in T : f(x) \neq h(x)\}\right| \geq \frac{|T|}{100} \text{ for all } h \in \mathcal{LTF}\right] \geq 1 - e^{-n/10}.$$

For each element of T, by assumption we have $\mathcal{D} \circ \varphi^{-1}(\{x\}) \geq (10n)^{-1}$. So the previous display implies that for any fixed φ such that $\left|\varphi(\mathrm{supp}(\mathcal{D}))\right| \geq 4n$,
$$\Pr_{f}\left[\mathsf{dist}_{\mathcal{D} \circ \varphi^{-1}}(f, \mathcal{LTF}) \geq \frac{1}{250}\right] \geq 1 - e^{-n/10}.$$

Combining with (7.1), we see that
$$\Pr_{f,\varphi}\left[\mathsf{dist}_{\mathcal{D} \circ \varphi^{-1}}(f, \mathcal{LTF}) \geq \frac{1}{250}\right] \geq 1 - 2e^{-n/10}.$$

Since $\varphi(x^{(1)}), \ldots, \varphi(x^{(m)})$ are independent samples from the push-forward distribution $\mathcal{D} \circ \varphi^{-1}$, by the guarantee of $\mathcal{A}$, the previous display implies
$$\Pr_{\mathcal{A}, \varphi, f, x^{(1)}, \ldots, x^{(m)}}[\mathcal{A}(S) = 0] \geq \frac{5}{6}(1 - 2e^{-n/10}) \geq \frac{2}{3}.$$

In conclusion, we have shown that $\mathcal{A}$ solves the problem (Definition 7.1) on $[10n]$ with $\alpha = (100 \log n)^{-1}$ and $\beta = 1/2$ with m samples. Using Lemma 7.2 with $\delta = (100 \log n)^{-1}$, we conclude that
$$m \geq \Omega\left(\frac{10n \cdot \delta^2}{\log(10n)}\right) = \Omega\left(\frac{n}{\log^3 n}\right). \qquad \square$$

8 Birthday Paradox Lemmas

8.1 Bipartite Birthday Paradox

The main lemma (Lemma 2.1) to be proved in this subsection has been previously incorporated as a crucial component of the analysis in [DR11], though without explicit statement. In this subsection, we provide an alternative proof of this lemma, drawing upon the classical birthday paradox from probability theory.

Lemma 8.1. *Let $\mathcal{D}$ be a probability distribution over a ground set B, and let $T \subseteq B$ be a subset such that $\mathcal{D}(T) \geq \varepsilon$ and $\mathcal{D}(\{t\}) \geq p$ for each element $t \in T$, for some $\epsilon, p > 0$. Let $X_1, \ldots, X_m$ be a sequence of m samples drawn independently from $\mathcal{D}$ and let S be the set they form. We have*

$$\Pr_S \left[\mathcal{D}(S \cap T) \leq \min\left\{\frac{\varepsilon p m}{4}, \frac{\varepsilon}{2}\right\}\right] \leq e^{-\varepsilon m/16}.$$

Proof. Let $S_j = \{X_1, \ldots, X_j\} \cap T$. For each $j \in [m]$, we define a random variable

$$Y_j = \begin{cases} 1 & \text{if } \mathcal{D}(T \setminus S_{j-1}) < \varepsilon/2 \\ \mathcal{D}(S_j \setminus S_{j-1}) & \text{otherwise.} \end{cases}$$

We clearly have $\Pr[Y_j \geq p \mid X_1, \ldots, X_{j-1}] \geq \varepsilon/2$. So $p^{-1}(Y_1 + \cdots + Y_m)$ stochastically dominates the sum of m independent Bernoulli random variables with parameter $\varepsilon/2$. By the multiplicative Chernoff bound, we have

$$\Pr\left[p^{-1}(Y_1 + \cdots + Y_m) \leq \varepsilon m/4\right] \leq e^{-\varepsilon m/16}.$$

The lemma follows from the fact that $Y_1 + \cdots + Y_m = \mathcal{D}(S \cap T)$ on the event $\mathcal{D}(S \cap T) \leq \varepsilon/2$. $\square$

Lemma 8.2 (Classical birthday paradox)**.** *Let n be a positive integer, and let $\mathcal{D}$ be a distribution over $[n+1]$. Let $p_i = \mathcal{D}(\{i\})$ for each $i \in [n+1]$, and let $\epsilon = p_1 + \cdots + p_n$. Let S and S' be two sets of samples of size m and m', respectively, drawn independently from $\mathcal{D}$, with m and m' satisfying $m \cdot m' \geq 100n/\varepsilon^2$ and that $m, m' \geq 200/\varepsilon$. Then with probability at least* 0.99*, there exists an $i \in [n]$ such that i appears in both S and S'.*

Proof. Let $T = \{i \in [n] : p_i \geq \varepsilon/(2n)\}$. Since $\mathcal{D}([n] \setminus T) < n \cdot \varepsilon/(2n) = \varepsilon/2$ and $\mathcal{D}([n]) = \varepsilon$, it follows that $\mathcal{D}(T) \geq \varepsilon/2$. Applying Lemma 8.1, we have

$$\Pr_S \left[\mathcal{D}(S \cap T) \leq \min\left\{\frac{\varepsilon^2 m}{16n}, \frac{\varepsilon}{4}\right\}\right] \leq e^{-\varepsilon m/32} \leq \frac{1}{500}.$$

Furthermore, we have

$$\begin{aligned} &\Pr_{S'} \left[S' \cap (S \cap T) = \emptyset \,\middle|\, \mathcal{D}(S \cap T) \geq \min\left\{\frac{\varepsilon^2 m}{16n}, \frac{\varepsilon}{4}\right\}\right] \\ &\qquad \leq \left(1 - \min\left\{\frac{\varepsilon^2 m}{16n}, \frac{\varepsilon}{4}\right\}\right)^{m'} \leq \exp\left(-\min\left\{\frac{\varepsilon^2 m \cdot m'}{16n}, \frac{\varepsilon m'}{4}\right\}\right) \leq \frac{1}{500}. \end{aligned}$$

Combining these two inequalities, we have that $\Pr_{S,S'}[S \cap S' = \emptyset] \leq 0.01$. $\square$

The ingredient we need in order to extend the classical birthday paradox to the bipartite graph version stated below in Lemma 2.1 is the well-known relationship between fractional matchings and vertex covers in bipartite graphs.

Lemma 2.1. *Let $G = (U, V, E)$ be a bipartite graph, with probability distributions μ on $U \cup \{\#\}$ and ν on $V \cup \{\#\}$. Assume that any vertex cover $C = C_1 \sqcup C_2$ of G, where $C_1 \subset U$ and $C_2 \subset V$, has $\mu(C_1) + \nu(C_2) \geq \varepsilon$. Let S be a set of m independent samples from μ and S' be a set of m' independent samples from ν, with m and m' satisfying $m \cdot m' \geq 100|U|/\epsilon^2$ and $m, m' \geq 100/\epsilon$. With probability at least* 0.99*, there exist $x \in S$ and $y \in S'$ such that (x, y) is an edge in G.*

Proof. Let P be the function over $U \cup V$ given by $P(u) = \mu(u)$ for all $u \in U$ and $P(v) = \nu(v)$ for all $v \in V$. Consider the following linear program on variables $\{x_e : e \in E\}$:

$$\begin{aligned} \text{maximize} \quad & \sum_{e \in E} x_e \\ \text{subject to} \quad & \sum_{e \in E : w \in e} x_e \le P(w), \quad \text{for each } w \in U \cup V \\ & x_e \ge 0, \quad \text{for all } e \in E. \end{aligned} \tag{8.1}$$

It's easy to see that the linear program has a maximum. Let $(\lambda_e)_{e \in E}$ be an optimal solution to the linear program. By linear programming duality, the optimal value $\sum_{e \in E} \lambda_e$ is equal to that of the following dual program over variables $\{y_w : w \in U \cup V\}$:

$$\begin{aligned} \text{minimize} \quad & \sum_{w \in U} P(w) y_w \\ \text{subject to} \quad & y_u + y_v \ge 1, \quad \text{for each edge } \{u, v\} \in E \\ & y_w \ge 0, \quad \text{for all } w \in U \cup V. \end{aligned}$$

Since all extreme points of the fractional vertex-cover polyhedron (of any bipartite graph) are 0/1-vectors (refer to [LP09, Theorem 7.1.3]), the optimal value of the dual program is the weight of a vertex cover and is thus at least ε by the assumption of the lemma. Hence we have $\sum_{e \in E} \lambda_e \ge \varepsilon$.

We then apply the birthday paradox argument. Let $U = \{u_1, \ldots, u_n\}$. Consider the distribution $\mathcal{D}$ on $[n+1]$ defined by

$$\mathcal{D}(n+1) = 1 - \sum_{e \in E} \lambda_e \quad \text{and} \quad \mathcal{D}(i) = \sum_{e \in E : u_i \in e} \lambda_e, \quad \text{for each } i \in [n],$$

We can then construct a Markov kernel K from $[n+1]$ to $V \cup \{\#\}$ such that $\mathcal{D}(\{i\}) \cdot K(v|i) = \lambda_{\{u_i, v\}}$ for all $i \in \operatorname{supp}(\mathcal{D})$ and $v \in V$. Also let $\varphi : [n+1] \to U \cup \{\#\}$ be the map that sends i to u_i for $i \in [n]$ and sends $(n+1)$ to $\#$. By (8.1), the push-forward measure $\mathcal{D} \circ \varphi^{-1}$ is dominated by μ on U, and the push-forward measure $\mathcal{D}K$ is dominated by ν on V. Therefore, using $\sum_{e \in E} \lambda_e \ge \varepsilon$, the desired conclusion follows from Lemma 8.2 and stochastic domination. $\square$

8.2 Hypergraph Birthday Paradox

We will prove a hypergraph version of the bipartite graph birthday paradox in this subsection. We begin with a couple of calculus facts.

Proposition 8.3. *Let k be a positive integer. The function $f : (0, +\infty) \to (0, 1)$ defined by*

$$f(x) = \ln\left(1 - (1 - e^{-x})^k\right)$$

is concave in x.

Proof. We have

$$f'(x) = -\frac{k e^{-x} (1 - e^{-x})^{k-1}}{1 - (1 - e^{-x})^k} = -\frac{k}{\sum_{\ell=0}^{k-1} (1 - e^{-x})^{-d}},$$

which is decreasing in x. $\square$

Proposition 8.4. *Suppose k is a positive integer and $a_1, \ldots, a_k$ are positive numbers such that $a_1 + \cdots + a_k = 1$. The function $g : (0, +\infty) \to (0, 1)$ defined by*

$$g(x) = \left(1 - \prod_{j=1}^{k}(1 - e^{-x^{-a_j}})\right)^x$$

is non-increasing in x.

Proof. Let $\rho : (0, 1) \to (0, 1)$ be defined by $\rho(x) = \frac{1-x}{x} \ln\left(\frac{1}{1-x}\right)$. By taking the derivative, it is easy to see that ρ is a decreasing function on $(0, 1)$. Define the function $h_j(x) = 1 - e^{-x^{-a_j}}$ for each $j \in [k]$, and let $h(x) = \prod_{j=1}^{k} h_j(x)$. We have

$$\frac{h'(x)}{h(x)} = -\sum_{j=1}^{k} a_j x^{-a_j - 1} \frac{e^{-x^{-a_j}}}{1 - e^{-x^{-a_j}}} = -\frac{1}{x}\sum_{j=1}^{k} a_j \cdot \rho(h_j(x)).$$

For each $j \in [k]$ and $x \in (0, 1)$, since $h_j(x) \geq h(x)$ we have $\rho(h_j(x)) \leq \rho(h(x))$. So

$$\frac{h'(x)}{h(x)} \geq -\frac{1}{x} \cdot \rho(h(x)) = -\frac{\rho(h(x))}{x} = \frac{(1 - h(x))\ln(1 - h(x))}{x \cdot h(x)}.$$

Therefore

$$\frac{\mathrm{d}}{\mathrm{d}x} \ln(g(x)) = \ln(1 - h(x)) - \frac{x \cdot h'(x)}{1 - h(x)} \leq 0,$$

and thus g is non-increasing. □

We use the preceding propositions to prove a birthday paradox where we bound the probability of k people sharing a birthday. To make the random variables arising from the balls-into-bins procedure mutually independent, we use the Poisson approximation as in [Mit96, Theorem 2.11].

Lemma 8.5 (Classical birthday paradox)**.** *Let n and k be two positive integers. Let $p_1, \ldots, p_{n+1}$ be nonnegative numbers that sum to $1/k$ and $\mathcal{D}$ be the probability distribution over $[n+1] \times [k]$ with $\mathcal{D}\big(\{(i, j)\}\big) = p_i$ for $i \in [n+1]$ and $j \in [k]$. Let $\epsilon = p_1 + \cdots + p_n$ and m be a positive integer with*

$$m \geq \frac{10kn^{(k-1)/k}}{\epsilon}.$$

Draw m independent samples from $\mathcal{D}$, and let $Y_{i,j}$ be the number of (i, j) drawn. Then

$$\Pr\Big[\forall i \in [n], \exists j \in [k] \text{ such that } Y_{ij} = 0\Big] \leq \frac{1}{100}.$$

Proof. Let $(Z_{ij})_{i\in[n+1],\, j\in[k]}$ be independent random variables such that each Z_{ij} follows the Poisson distribution with mean mp_i. It is well known that the conditional joint distribution of $(Z_{ij})_{i\in[n+1],\, j\in[k]}$ on the event $\left\{\sum_{i=1}^{n+1}\sum_{j=1}^{n} Z_{ij} = m\right\}$ is exactly the joint distribution of $(Y_{ij})_{i\in[n+1],\, j\in[k]}$. Let function $f : \mathbb{Z}_{\geq 0}^{(n+1)k} \to \{0, 1\}$ be defined as

$$f\left((x_{ij})_{i\in[n+1],\, j\in[k]}\right) = \mathbb{1}\big\{\forall i \in [n], \exists j \in [k] \text{ such that } x_{ij} = 0\big\}.$$

Then since $\mathbb{E}\left[f\left((Y_{ij})_{i\in[n+1],\,j\in[k]}\right)\right]$ is decreasing in m, we have

$$\begin{aligned}\mathbb{E}\left[f\left((Z_{ij})_{i\in[n+1],\,j\in[k]}\right)\right] &\geq \sum_{t=0}^{m}\mathbb{E}\left[f\left((Z_{ij})_{i\in[n+1],\,j\in[k]}\right)\Big|\sum\nolimits_{i,j} Z_{ij}=t\right]\cdot\Pr\left[\sum\nolimits_{i,j} Z_{ij}=t\right]\\ &\geq \mathbb{E}\left[f\left((Z_{ij})_{i\in[n+1],\,j\in[k]}\right)\Big|\sum\nolimits_{i,j} Z_{ij}=m\right]\cdot\Pr\left[\sum\nolimits_{i,j} Z_{ij}\leq m\right]\\ &= \mathbb{E}\left[f\left((Y_{ij})_{i\in[n+1],\,j\in[k]}\right)\right]\cdot\Pr\left[\sum\nolimits_{i,j} Z_{ij}\leq m\right]. \end{aligned} \tag{8.2}$$

Since $\sum_{i,j} Z_{ij}$ follows a Poisson distribution with mean m, it is easy to see (for example by the Berry-Esseen theorem) that

$$\Pr\left[\sum\nolimits_{i,j} Z_{ij}\leq m\right]\geq\frac{1}{4}. \tag{8.3}$$

By independence between the variables Z_{ij}, we have

$$\begin{aligned}\mathbb{E}\left[f\left((Z_{ij})_{i\in[n+1],\,j\in[k]}\right)\right] &= \prod_{i=1}^{n}\Pr\left[\exists j\in[k] \text{ such that } Z_{ij}=0\right]\\ &= \prod_{i=1}^{n}\left(1-(1-e^{-mp_i})^k\right)\\ &\leq \left(1-(1-e^{-\varepsilon m/n})^k\right)^n && \text{(by Proposition 8.3)}\\ &\leq \left(1-(1-e^{-10kn^{-1/k}})^k\right)^n && \text{(since } m\geq 10k\varepsilon^{-1}n^{(k-1)/k}\text{)}\\ &\leq 1-(1-e^{-10k})^k\leq\frac{1}{400} && \text{(by Proposition 8.4).} \end{aligned} \tag{8.4}$$

The conclusion follows by combining (8.2), (8.3) and (8.4). □

The hypergraph version of Lemma 2.1 follows by a similar linear programming argument.

Lemma 2.2. *Let $G=(V,E)$ be a k-uniform hypergraph and let μ be a probability distribution over $V\cup\{\#\}$ such that any vertex cover C of G has $\mu(C)\geq\varepsilon$. Let S be a set of m sample from μ with*

$$m\geq\frac{10k^2|V|^{(k-1)/k}}{\epsilon}.$$

Then S contains an edge in G with probability at least 0.99.

Proof. Consider the following linear program over variables $\{x_e : e\in E\}$:

$$\begin{aligned} \text{maximize} \quad & \sum_{e\in E} x_e\\ \text{subject to} \quad & \sum_{e\in E: v\in e} x_e\leq\mu(v), \quad \text{for each } v\in V\\ & x_e\geq 0, \quad \text{for all } e\in E. \end{aligned}$$

Let $(\lambda_e)_{e\in E}$ be an optimal solution. Then

$$C=\left\{v\in V : \sum_{e\in E: v\in e}\lambda_e=\mu(v)\right\}$$

must be a vertex cover of H. So by assumption

$$\varepsilon \le \sum_{v \in C} \mu(v) \le k \sum_{e \in E} \lambda_e.$$

By Carathéodory's theorem there exists another optimal solution $(\lambda'_e)_{e \in E}$ to the linear program such that $\big|\{e \in E : \lambda'_e \neq 0\}\big| \le |V|$. Furthermore, we still have $\sum_{e \in E} \lambda'_e = \sum_{e \in E} \lambda_e \ge \varepsilon/k$.

We are now ready for the birthday paradox argument. Let n be the size of the set $\{e \in E : \lambda'_e \neq 0\}$ and denote its elements by $e_1, \ldots, e_n$. There obviously exists a map $\varphi : [n+1] \times [k] \to V \cup \{\#\}$ that maps the set $\{(i,j) : j \in [k]\}$ one-to-one onto the vertices of e_i for each $i \in [n]$, and maps the set $\{(n+1, j) : j \in [k]\}$ to $\{\#\}$. Considering the distribution $\mathcal{D}$ on $[n+1] \times [k]$ defined by

$$\begin{aligned}\mathcal{D}\big(\{(i,j)\}\big) &= \lambda'_{e_i}, \text{ for all } i \in [n] \text{ and } j \in [k],\\ \mathcal{D}\big(\{(n+1,j)\}\big) &= \left(\frac{1}{k} - \sum_{i=1}^{n} \lambda'_{e_i}\right), \text{ for all } j \in [k].\end{aligned}$$

Using $\sum_i \lambda'_{e_i} \ge \varepsilon/k$ and $n \le |V|$, the lemma follows by Lemma 8.5 and stochastic domination. □

References

[AC06] N. Ailon and B. Chazelle. Information theory in property testing and monotonicity testing in higher dimension. *Information and Computation*, 204(11):1704–1717, 2006.

[BFH21] Eric Blais, Renato Ferreira Pinto Jr, and Nathaniel Harms. VC dimension and distribution-free sample-based testing. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 504–517, 2021.

[CP22] Xi Chen and Shyamal Patel. *Distribution-free Testing for Halfspaces (Almost) Requires PAC Learning*, pages 1715–1743. 2022.

[CX16] Xi Chen and Jinyu Xie. Tight bounds for the distribution-free testing of monotone conjunctions. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 54–71. SIAM, 2016.

[DR11] E. Dolev and D. Ron. Distribution-free testing for monomials with a sublinear number of queries. *Theory of Computing*, 7(1):155–176, 2011.

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

[GLR01] David Guijarro, Victor Lavin, and Vijay Raghavan. Monotone term decision lists. *Theoretical Computer Science*, 259(1-2):549–575, 2001.

[GS09] D. Glasner and R. Servedio. Distribution-free testing lower bound for basic boolean functions. *Theory of Computing*, 5(10):191–216, 2009.

[HK07] S. Halevy and E. Kushilevitz. Distribution-free property-testing. *SIAM Journal on Computing*, 37(4):1107–1138, 2007.

[HK08a] S. Halevy and E. Kushilevitz. Distribution-free connectivity testing for sparse graphs. *Algorithmica*, 51(1):24–48, 2008.

[HK08b] S. Halevy and E. Kushilevitz. Testing monotonicity over graph products. *Random Structures & Algorithms*, 33(1):44–67, 2008.

[LP09] László Lovász and Michael D Plummer. *Matching theory*, volume 367. American Mathematical Soc., 2009.

[Mit96] Michael Mitzenmacher. The power of two choices in randomized load balancing. *PhD thesis, University of California at Berkeley*, 1996.

[Riv87] Ronald L. Rivest. Learning decision lists. *Machine Learning*, 2(3):229—246, 1987.

[SSBD14] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[Val84] L.G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[WY19] Yihong Wu and Pengkun Yang. Chebyshev polynomials, moment matching, and optimal estimation of the unseen. *Annals of Statistics*, 47(2):857–883, 2019.