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

Xi Chen
Columbia University
New York, USA
xichen@cs.columbia.edu

Yumou Fei
Peking University
Beijing, China
feiym2002@stu.pku.edu.cn

Shyamal Patel
Columbia University
New York, USA
shyamalpatelb@gmail.com

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 draw $\tilde{\Omega}(n)$ samples when the algorithm is sample-based.

CCS CONCEPTS

• **Theory of computation** → **Streaming, sublinear and near linear time algorithms**; **Sketching and sampling**; **Lower bounds and information complexity**; **Randomness, geometry and discrete structures**.

KEYWORDS

Distribution-Free Property Testing, Decision Lists

ACM Reference Format:
Xi Chen, Yumou Fei, and Shyamal Patel. 2024. Distribution-Free Testing of Decision Lists with a Sublinear Number of Queries. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing (STOC '24), June 24–28, 2024, Vancouver, BC, Canada.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3618260.3649717

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 [12], 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 [13] shows that the VC dimension of a class essentially captures the number of samples needed for its PAC learning [15], which gives a tight bound of $\Theta(n)$ samples for learning decision lists. Moreover, this bound is tight even if we give the learner query access to the underlying decision list [14].

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 [7] and has been studied extensively [1–6, 9–11]. While testing is known to be no harder than proper learning [7], 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 [6], 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.[3] In [5], Dolev and Ron obtained a distribution-free testing algorithm for conjunctions with $\tilde{O}(\sqrt{n})$ queries. Later in [4], 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,[4] on the other hand, Blais, Ferreira Pinto Jr. and Harms [2] 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 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 [3] 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:

STOC '24, June 24–28, 2024, Vancouver, BC, Canada

ACM ISBN 979-8-4007-0383-6/24/06
https://doi.org/10.1145/3618260.3649717

[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] Recall that conjunctions are a subclass of decision lists, which in turn are a subclass of halfspaces.
[4] A tester is sample-based if it can only draw samples $x_1, \ldots, x_q \sim \mathcal{D}$ and receive $f(x_1), \ldots, f(x_q)$.

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.*[5]

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 [8] if all literals α_i in the list are positive) and then proving a reduction to testing general decision lists.

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. The same lower bound also applies to testing monotone decision lists.*

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. The same lower bound also applies to testing monotone conjunctions, conjunctions, and monotone decision lists.*

As a warm-up for our main algorithm behind Theorem 1.1, we give an optimal distribution-free testing algorithm for *total orderings*, which highlights, in a simplified setting, some of the most crucial ideas behind the main algorithm for monotone decision lists. To our knowledge, this is also the first tester for total orderings in the distribution-free setting. The input consists of 1) oracle access to a comparison function $<_\sigma$ over $[n]$ (i.e., one can pick $i \neq j \in [n]$ to reveal whether $i <_\sigma j$ or $j <_\sigma i$); and 2) sampling access to a distribution $\mathcal{D}$ over the set of $\binom{n}{2}$ many 2-subsets of $[n]$. The goal is to determine whether $<_\sigma$ is a total ordering or ε-far from total orderings with respect to $\mathcal{D}$. Equivalently, $<_\sigma$ can be considered as a tournament graph G_σ over $[n]$ and the algorithm is given oracle access to it (i.e., one can pick $u \neq v \in [n]$ and query whether (u, v) or (v, u) is in G_σ. The goal is to decide whether G_σ is acyclic or ε-far from acyclic with respect to $\mathcal{D}$ (i.e., any feedback edge set of G_σ has probability mass at least ε in $\mathcal{D}$, where a feedback edge set is a set of edges such that the graph G_σ becomes acyclic after its removal).

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.*

The paper is organized as follows. In Section 2, we introduce the preliminaries, including two birthday-paradox-type lemmas. In Section 3, we prove the upper bound part of Theorem 1.4, which serves as a warm-up for the proof of Theorem 1.1. For space reasons, we omit the proofs of the reduction to general decision lists, Theorem 1.2, and Theorem 1.3. We refer the interested reader to the Arxiv version of the paper.

Techincal Overview. We start by describing an easy $\Omega(\sqrt{n})$ one-sided[6] lower bound for total orderings. We construct a distribution

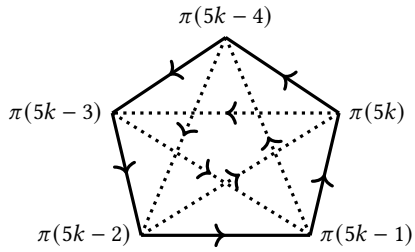

Figure 1: One-side Lower Bound Construction for Total Orderings. An edge from x to y indicates that $x <_\sigma y$. The solid edges in the figure denote those in the support of $\mathcal{D}_{NO}$.

$\mathcal{D}_{\text{NO}}$ over pairs $(<_\sigma, \mathcal{D})$ such that $<_\sigma$ is far from total orderings under $\mathcal{D}$. It suffices to show that no deterministic algorithm with $o(\sqrt{n})$ queries can find a violation in $(<_\sigma, \mathcal{D}) \sim \mathcal{D}_{\text{NO}}$ (or equivalently, a (directed) cycle in the tournament graph G_σ) with probability at least 2/3.

To draw $(<_\sigma, \mathcal{D}) \sim \mathcal{D}_{NO}$ [7] we first draw a random permutation π over $[n]$ and use it partition $[n]$ into $n/5$ groups, where the k-th group V_k consists of vertices $\pi(5k-4), \ldots, \pi(5k)$, for each $k \in [n/5]$. The comparison function $<_\sigma$ over each group V_k is set according to Figure 1. Across two different groups, $<_\sigma$ is made to be consistent with a total ordering, namely, $\pi(x) <_\sigma \pi(y)$ if $\lceil x/5 \rceil < \lceil y/5 \rceil$. Finally the distribution $\mathcal{D}$ is uniform over edges $\{\pi(5k-4), \pi(5k-3)\}, \{\pi(5k-3), \pi(5k-2)\}, \{\pi(5k-2), \pi(5k-1)\}, \{\pi(5k-1), \pi(5k)\}, \{\pi(5k), \pi(5k-4)\}$ of each group $k \in [n/5]$. We write E_k to denote the set of these five edges in the k-th group V_k.

Clearly, to make $<_\sigma$ into a total ordering, one must change at least one edge in each E_k, so $<_\sigma$ is $(1/5)$-far from total orderings. On the other hand, in order for a one-sided algorithm to reject, it must find a cycle in V_k for some k. Using a birthday paradox argument, with only $o(\sqrt{n})$ samples, edges sampled from $\mathcal{D}$ most likely lie in distinct groups. When this happens, it is unlikely for the algorithm to find a cycle using $o(\sqrt{n})$ queries to the black-box oracle. 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 x with large support (ignoring some details, testing total orderings can be thought of as testing decision lists with the restriction that the algorithm can only query the function f on strings x with support size 2).

We now use instances in $\mathcal{D}_{\text{NO}}$ to discuss ideas behind our $\tilde{O}(\sqrt{n}/\varepsilon)$-query tester for total orderings. In particular, consider a one-sided tester that aims to find a violation (i.e., a cycle in G_σ) in $(<_\sigma, \mathcal{D})$ from $\mathcal{D}_{\text{NO}}$. It must draw $\Omega(\sqrt{n})$ samples from $\mathcal{D}$. After doing so, it is likely to have drawn two edges from the same E_k, say $\{\pi(5k-4, 5k-3)\}$ and $\{\pi(5k-2, 5k-1)\}$ for some $k \in [n/5]$. If the algorithm continues to query the rest of four edges between these four vertices, then a cycle will be found as desired. That said, the algorithm does not know which pair of edges sampled from

[5] 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.

[6] Recall that a testing algorithm is *one-sided* if it never rejects $(<_\sigma, \mathcal{D})$ when $<_\sigma$ is a total ordering.

[7] As it will become clear soon, partitioning $[n]$ into triangles would already yield the $\Omega(\sqrt{n})$ one-sided lower bound. The more involved construction of $\mathcal{D}_{\text{NO}}$ here poses extra challenges to motivate discussion on some of the most crucial ideas behind our testing algorithm for total orderings.

$\mathcal{D}$ lies in the same group, and working on all pairs would require $\Omega(n)$ queries.[8]

To circumvent this issue, we create a "sketch" to attack $(<_\sigma, \mathcal{D})$ from $\mathcal{D}_{\text{NO}}$ as follows:

(1) Sample $\sqrt{n}$ vertices from $[n]$ uniformly at random; sort them into $\ell_1 <_\sigma \ell_2 <_\sigma \cdots <_\sigma \ell_{\sqrt{n}}$ using $O(\sqrt{n}\log n)$ queries on $<_\sigma$;
(2) Partition $[n]$ into *blocks* $B_0, \ldots, B_{\sqrt{n}}$ where B_i consists of all $k \in [n]$ such that running binary search of k on $\ell_1, \ldots, \ell_{\sqrt{n}}$ sandwiches it in $\ell_i <_\sigma k <_\sigma \ell_{i+1}$.

We note the following properties of the sketch:

(1) We cannot afford to compute the blocks but given any $k \in [n]$, it is easy to find the block B_i that contains k with $O(\log n)$ queries (by just running binary search);
(2) With high probability (over samples used to build the sketch), every B_i is of size $\tilde{O}(\sqrt{n})$.

With this sketch in hand, we can use it to find a violation in $(<_\sigma, \mathcal{D})$ from $\mathcal{D}_{\text{NO}}$ efficiently by (1) sampling $O(\sqrt{n})$ edges from $\mathcal{D}$ so that with high probability two edges $\{u, v\}$ and $\{u', v'\}$ from the same group are sampled; (2) find the block of every vertex in the $O(\sqrt{n})$ edges sampled in (1); let U denote this set of $O(\sqrt{n})$ vertices; (3) for every block B_i and every two vertices in $U \cap B_i$, query $<_\sigma$ on them and reject if a cycle is found within $U \cap B_i$ for some i. Given that most likely u, v, u', v' lie in the same block, the algorithm finds a violation with high probability; its query complexity is at most $\tilde{O}(\sqrt{n})$ because $|U \cap B_i|$ can be bounded from above by $O(\log n)$ with high probability. So the savings come from the fact that we only query edges between vertices in the same block.

The algorithm for the general case (rather than just dealing with instances from $\mathcal{D}_{\text{NO}}$) follows the same high level idea. It starts by building a sketch but the vertices $\ell_1, \ldots, \ell_{\sqrt{n}}$ used to build it are no longer sampled uniformly but from a natural distribution $\mathcal{D}^*$ over $[n]$ defined from $\mathcal{D}$: to draw $\ell \sim \mathcal{D}^*$, one first draws an edge from $\mathcal{D}$ and then set ℓ to be one of its two vertices uniformly. Accordingly, the second property of the sketch becomes that every B_i has probability mass at most $O(1/\sqrt{n})$ in $\mathcal{D}^*$. With such a sketch in hand, we consider cycles in $<_\sigma$. Since $<_\sigma$ is ε-far from total orderings under $\mathcal{D}$, we can divide cycles in G_σ into two types: those with vertices lying in multiple blocks (called *long* cycles) and those that are completely contained within a single block (called *local* cycles), and consider two cases: the distance to total orderings mainly comes from long cycles or local cycles. To deal with the case where there are many violating long cycles, we show that $\{u, v\} \sim \mathcal{D}$ satisfies $u <_\sigma v, u \in B_i, v \in B_j$ but $i > j$ with probability $\Omega(\varepsilon)$. As a result, drawing $O(1/\varepsilon)$ samples from $\mathcal{D}$ and finding buckets of their vertices leads to a violation with high probability. The case of local cycles, on the other hand, is the case with instances of $\mathcal{D}_{\text{NO}}$. To this end we use a birthday paradox lemma in Section 2 to show that with $\tilde{O}(\sqrt{n})$ samples U from $\mathcal{D}^*$, some $U \cap B_i$ contains a cycle with high probability, which can be found by brute-force search of each $U \cap B_i$.

Unfortunately, several aspects of this approach break when attempting to adapt the algorithm to monotone decision lists. Note that a monotone decision list $f : \{0,1\}^n \to \{0,1\}$ 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 two strings x, y with $f(x) \neq f(y)$: If $f(x \vee y) = f(x)$, then the rule that fires in x is ranked higher. To accommodate this in the sketch, we bucket elements of $[n]$ into blocks $B_0, \ldots, B_{\sqrt{n}}$ that now have alternating values, i.e. all indices $k \in B_i$ have that $f(e_k) = i \mod 2$, where $e_k \in \{0,1\}^n$ denotes the string in which the only 1-entry is k. However, even for a monotone decision list f, blocks $B_0, \ldots, B_{\sqrt{n}}$ of a sketch no longer guarantee that all elements in B_i are ranked higher than those in B_{i+1}; only a weaker guarantee holds that all elements in B_i are ranked higher than those in B_j for all $j > i+1$.

The primary challenge when testing monotone decision lists is determining what constitutes a violation. In the case of total orderings, each comparison provides a concrete bit, indicating that one element is larger than the other under $<_\sigma$, and a violation is 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 the algorithm that some zero rule fired in x is ranked higher than all the one rules fired in x. To address this, we design a procedure that determines the value of the maximum element $k \in \text{supp}(x)$. However, this procedure is effective only for blocks B_i that contain a small number of indices, say n^δ for some small constant $\delta > 0$ (the number of queries made by the algorithm is linear in n^δ so is efficient only when δ is small). Once we identify the maximum elements, cycles in an associated hypergraph naturally leads to violations. As a simple example, let x and y be two strings with $f(x) = 0$ and $f(y) = 1$. Let k, ℓ be maximum elements in x and y, respectively. If in addition we have $\ell \in \text{supp}(x)$ and $k \in \text{supp}(y)$, then we get a violation because being the maximum element in x, k should be ranked higher than ℓ but on the other hand, y tells us that ℓ is ranked higher.

Nevertheless, this procedure is insufficient for testing since many blocks in the sketch may have more than n^δ 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 and B_i is a large block, 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 large block. 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 algorithm to test 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 it is not clear how to find r exactly, 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 sample strings and try each of them out as the candidate default string r.

2 PRELIMINARIES

Notation. Given a positive integer n, we write $[n]$ to denote $\{1, \ldots, n\}$. Given two integers $a \leq b$, we write $[a : b]$ to denote

[8] Note that if the algorithm receives two samples that are consecutive in the same group, then it certainly knows this because they share a vertex; the construction, however, makes sure that the triangle they form is never a cycle.

the set of integers $\{a, \ldots, b\}$ 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 $\sum_{p \in P} \mathcal{D}(p)$. We will denote by $\mathrm{supp}(\mathcal{D})$ the set $\{p \in S : \mathcal{D}(p) > 0\}$. 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).

For any string $x \in \{0,1\}^n$, let $\mathrm{supp}(x)$ denote the set $\{i \in [n] : x_i = 1\}$. 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, v)[9], where π is a permutation over $[n]$ and $v \in \{0,1\}^{n+1}$, such that $f(x) = v_j$ if j is the smallest integer in $[n]$ such that $x_{\pi(j)} = 1$, and $f(x) = v_{n+1}$ when $x = 0^n$. Variable $i \in [n]$ is said to be a b-rule variable if $v_j = b$ 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 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 $\min_\pi(x) < \min_\pi(y)$. As such, for a decision list $\succ_f$ naturally corresponds to the ordering of the rules in the decision list.

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, v), where $\pi : [n] \to [n]$ is a permutation over $[n]$, $\mu \in \{0,1\}^n$, and $v \in \{0,1\}^{n+1}$, such that $f(x) = v_j$ if j is the smallest integer in $[n]$ such that $x_{\pi(j)} = \mu_{\pi(j)}$, and $f(x) = v_{n+1}$ if no such j exists. Similarly, we say variable $i \in [n]$ is a b-rule variable if $v_j = b$ 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 $f(z_1), \ldots, f(z_q)$.

Birthday Paradox Lemmas. As highlighted earlier in the sketch of our algorithms, birthday paradox arguments play an important role in the analysis. In particular, we will need two birthday paradox lemmas, 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 [5] for the distribution-free testing of monomials, though without an explicit statement. We omit the proofs and include them in the full version of the paper.

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 samples 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

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

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

(2) reject with probability at least 2/3 if $<_\sigma$ is ε-far from total orderings with respect to $\mathcal{D}$, i.e. for any total ordering $<_\tau$

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

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 proof for decision lists presented in the Arxiv version to show that any tester must make $\Omega(\sqrt{n})$ 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:

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 $i \in [0:k]$,

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

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 $u \in [n]$

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

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

We omit the proof for brevity and refer the interested reader to the Arxiv version.

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 $\ell =$ FINDBLOCK($<_\sigma, \mathcal{S}, u$).

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 $\mathcal{D}(R) \geq \varepsilon$.*

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**

For brevity, we omit the proof.

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:

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

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

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**

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 $\mathcal{D}(C_1) + \mathcal{D}^*(C_2) \geq \varepsilon/2$.*

We omit the proof and refer the interested reader to the Arxiv version.

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:

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

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

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

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

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

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 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. □

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 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. □

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.*

A testing algorithm for general decision lists will follow 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.

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**

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 (except when T is either empty, consists only of 0-strings of f or only 1-strings of f, 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

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

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

Algorithm 7 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

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

Let $m = |T|$, $T_0 \leftarrow \{x \in T : f(x) = 0\}$ and $T_1 \leftarrow \{x \in T : f(x) = 1\}$
for i from 1 to m **do**
  if both T_0 and T_1 are nonempty **then**
    Let $b = f(\vee_{x \in T_0 \cup T_1} x)$; Set $x^{(i)} \leftarrow$ FindRep$(f, T_b, T_{\overline{b}})$ and $T_b \leftarrow T_b \setminus \{x^{(i)}\}$
  else
    Let b be such that $T_b \neq \emptyset$; Set $x^{(i)} \leftarrow$ an arbitrary string in T_b and $T_b \leftarrow T_b \setminus \{x^{(i)}\}$
  end if
end for
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 $j \in I_{\ell+1}$
Check if $k \geq 2$ and $\mathcal{S} = (s^{(1)}, \ldots, s^{(k)})$ is consistent with f, where $s^{(\ell)} \leftarrow \vee_{i \in I_\ell} x^{(i)}$
return $\mathcal{S}$ if so and **return** nil otherwise

We summarize properties of FindRep in the following lemma:

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

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

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

We now describe the procedure 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. *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.*

We omit the proof. 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

$$B_{f,\mathcal{S},\ell} = \big\{ i \in [n] : \textsc{FindBlock}(f, \mathcal{S}, e_i) = \ell \big\},$$

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

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

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

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

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 Preprocess$(f, \mathcal{D}, \varepsilon)$). The probability that Preprocess$(f, \mathcal{D}, \varepsilon)$ returns a sketch $\mathcal{S}$ that is not scattered is $o_n(1)$ over the randomness of T^*.*

Due to space constraints, we omit the proof here. The preprocessing stage ends by running FindBigBlocks$(f, \mathcal{D}, \varepsilon, \mathcal{S})$ (Algorithm 10):

Lemma 4.9. *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 $\mathcal{L}$:

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

Then we have

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

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

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

Due to space constraints, we again omit the proof. 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 $j \in \mathrm{supp}(x)$ such that $f(e_j) \ne f(x)$, 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 $i \in \mathrm{supp}(x)$ or* nil*; whenever it returns an $i \in \mathrm{supp}(x)$, we always have*

$$\textsc{FindBlock}(f, \mathcal{S}, e_i) = \textsc{FindBlock}(f, \mathcal{S}, x) \quad \text{and} \quad f(x) = f(e_i). \tag{4.2}$$

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 $i \in \mathrm{supp}(x)$ returned additionally satisfies

$$f(e_i \vee e_j) = f(e_i), \quad \text{for all } j \in \mathrm{supp}(x).$$

We omit the proof.

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

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

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

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

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

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

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

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

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 $(u, W) \in V(H)$,

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

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.*

We omit the proof. 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
$$\textsc{FindBlock}(f, \mathcal{S}, e_v) \leq \textsc{FindBlock}(f, \mathcal{S}, e_u) - 2;$$
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 FindBlock$(f, \mathcal{S}, e_u)$, 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
$$\big|\textsc{FindBlock}(f, \mathcal{S}, e_u) - \textsc{FindBlock}(f, \mathcal{S}, e_v)\big| = 1$$
and FindBlock$(f, \mathcal{S}, e_u)$, 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
$$\textsc{FindBlock}(f, \mathcal{S}, e_w) + 2 = \textsc{FindBlock}(f, \mathcal{S}, e_v) + 1 = \textsc{FindBlock}(f, \mathcal{S}, e_u) \quad (4.3)$$
and FindBlock$(f, \mathcal{S}, e_u)$, FindBlock$(f, \mathcal{S}, e_v)$, 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 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)
$$\max_{(u,W) \in C} \textsc{FindBlock}(f, \mathcal{S}, e_u) = \min_{(u,W) \in C} \textsc{FindBlock}(f, \mathcal{S}, e_u) + 1. \quad (4.4)$$

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

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 $c \in \{0, 1, \ldots, 5\}$.*

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 $\mathcal{D} \circ \varphi^{-1}(\text{nil}) \geq \epsilon/2$, or for some $c \in [5]$, any feedback vertex set U of type-c cycles in H has $\mathcal{D} \circ \varphi^{-1}(U) \geq \Omega(\varepsilon)$.*

For brevity, we omit the proofs.

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 $\mathcal{D} \circ \varphi^{-1}(\text{nil}) \geq \epsilon/2$ 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. *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 TestType-3 rejects with probability at least 0.9.*

Lemma 4.20. *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 TestType-4 rejects with probability at least 0.9.*

Lemma 4.21. *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 TestType-5 rejects with probability at least 0.9.*

Due to space constraints, we defer the proofs to the full version. Theorem 4.1 now follows directly:

Proof of Theorem 4.1. The overall query complexity of 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 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, 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 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 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 MaxIndex$(f, \mathcal{S}, x) = \text{nil}$ 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 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. □

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 $f(x) \neq f(y)$,

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

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$(f, \mathcal{S}, \mathcal{L}, y) \in \text{supp}(x)$

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) $\left|\text{FindBlock}(f, \mathcal{S}, x) - \text{FindBlock}(f, \mathcal{S}, y)\right| = 1$;
ii) FindBlock$(f, \mathcal{S}, x)$, FindBlock$(f, \mathcal{S}, y) \notin \mathcal{L} \cup N(\mathcal{L})$;
iii) MaxIndex$(f, \mathcal{S}, \mathcal{L}, y) \in \text{supp}(x)$; 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

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

ACKNOWLEDGEMENTS

Xi Chen is supported in part by NSF grants IIS-1838154, CCF-2106429, and CCF-2107187. Shyamal Patel is supported in part by NSF grants IIS-1838154, CCF-2106429, CCF-2107187, CCF-2218677, ONR grant ONR-13533312, and an NSF Graduate Student Fellowship.

REFERENCES

[1] N. Ailon and B. Chazelle. 2006. Information theory in property testing and monotonicity testing in higher dimension. *Information and Computation* 204, 11 (2006), 1704–1717.
[2] Eric Blais, Renato Ferreira Pinto Jr, and Nathaniel Harms. 2021. VC dimension and distribution-free sample-based testing. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*. 504–517.
[3] Xi Chen and Shyamal Patel. 2022. Distribution-free Testing for Halfspaces (Almost) Requires PAC Learning. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 1715–1743.
[4] Xi Chen and Jinyu Xie. 2016. Tight bounds for the distribution-free testing of monotone conjunctions. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 54–71.
[5] E. Dolev and D. Ron. 2011. Distribution-Free Testing for Monomials with a Sublinear Number of Queries. *Theory of Computing* 7, 1 (2011), 155–176.
[6] D. Glasner and R. Servedio. 2009. Distribution-Free Testing Lower Bound for Basic Boolean Functions. *Theory of Computing* 5, 10 (2009), 191–216.
[7] Oded Goldreich, Shari Goldwasser, and Dana Ron. 1998. Property testing and its connection to learning and approximation. *Journal of the ACM (JACM)* 45, 4 (1998), 653–750.
[8] David Guijarro, Victor Lavin, and Vijay Raghavan. 2001. Monotone term decision lists. *Theoretical Computer Science* 259, 1-2 (2001), 549–575.
[9] S. Halevy and E. Kushilevitz. 2007. Distribution-Free Property-Testing. *SIAM J. Comput.* 37, 4 (2007), 1107–1138.
[10] S. Halevy and E. Kushilevitz. 2008. Distribution-Free Connectivity Testing for Sparse Graphs. *Algorithmica* 51, 1 (2008), 24–48.
[11] S. Halevy and E. Kushilevitz. 2008. Testing monotonicity over graph products. *Random Structures & Algorithms* 33, 1 (2008), 44–67.
[12] Ronald L. Rivest. 1987. Learning decision lists. *Machine Learning* 2, 3 (1987), 229––246.
[13] Shai Shalev-Shwartz and Shai Ben-David. 2014. *Understanding machine learning: From theory to algorithms*. Cambridge university press.
[14] György Turán. 1993. Lower bounds for PAC learning with queries. In *Proceedings of the sixth annual conference on Computational learning theory*. 384–391.
[15] L.G. Valiant. 1984. A Theory of the Learnable. *Commun. ACM* 27, 11 (1984), 1134–1142.

Received 13-NOV-2023; accepted 2024-02-11