Finding Monotone Patterns in Sublinear Time, Adaptively

Omri Ben-Eliezer
Massachusetts Institute of Technology, USA

Shoham Letzter
University College London, United Kingdom

Erik Waingarten
Stanford University, USA

Abstract

We investigate adaptive sublinear algorithms for finding monotone patterns in sequential data. Given fixed $2 \leq k \in \mathbb{N}$ and $\varepsilon > 0$, consider the problem of finding a length-k increasing subsequence in a sequence $f\colon [n] \to \mathbb{R}$, provided that f is ε-far from free of such subsequences. It was shown by Ben-Eliezer et al. [FOCS 2019] that the non-adaptive query complexity of the above task is $\Theta((\log n)^{\lfloor \log_2 k \rfloor})$. In this work, we break the non-adaptive lower bound, presenting an adaptive algorithm for this problem which makes $O(\log n)$ queries. This is optimal, matching the classical $\Omega(\log n)$ adaptive lower bound by Fischer [Inf. Comp. 2004] for monotonicity testing (which corresponds to the case $k = 2$). Equivalently, our result implies that testing whether a sequence decomposes into k monotone subsequences can be done with $O(\log n)$ queries.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases property testing, monotone patterns, monotone decomposition, adaptivity

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

Related Version The full version of this work is hosted on arXiv: 1911.01169.

Funding *Shoham Letzter*: Research supported by Dr. Max Rössler, the Walter Haefner Foundation and by the ETH Zurich Foundation.
Erik Waingarten: This work is supported by the National Science Foundation under Award No. 2002201 and Moses Charikar's Simons Investigator award.

1 Introduction

Pattern avoidance and detection in sequential data is a central problem in theoretical computer science and combinatorics [57], dating back to the seminal work of Knuth [38] (from a computer science perspective), and Simion and Schmidt [55] (from a combinatorial perspective). Studying the computational problem within the framework of sublinear algorithms, Newman, Rabinovich, Rajendraprasad, and Sohler [44, 45] considered the problem of property testing for forbidden order patterns in a sequence, where one of the central special cases they considered was that of *monotone patterns*. The property testing problem of detecting monotone patterns generalizes classical monotonicity testing in sequences, and is tightly connected to the longest increasing subsequence (LIS) problem [46].

For an integer $k \in \mathbb{N}$ and a sequence $f\colon [n] \to \mathbb{R}$, a *length-$k$ monotone subsequence of* f is a tuple of k indices, $(i_1, \dots, i_k) \in [n]^k$, such that $i_1 < \cdots < i_k$ and $f(i_1) < \cdots < f(i_k)$. More generally, for a permutation $\pi\colon [k] \to [k]$, a *π-pattern of f* is given by a tuple of k indices $i_1 < \cdots < i_k$ such that $f(i_{j_1}) < f(i_{j_2})$ whenever $j_1, j_2 \in [k]$ satisfy $\pi(j_1) < \pi(j_2)$. A sequence f is π-free if there are no subsequences of f with order pattern π. For a fixed $k \in \mathbb{N}$ and a pattern π of length k, the goal is to test whether a function $f\colon [n] \to \mathbb{R}$ is π-free or

ε-far from π-free (that is, any π-free function g differs from f on at least εn inputs). The algorithmic task proposed in [45] and studied in this paper is as follows.

> *For $2 \leq k \in \mathbb{N}$ and $\varepsilon > 0$, design a randomized algorithm that, given query access to a function $f\colon [n] \to \mathbb{R}$, distinguishes with probability at least $9/10$ between the case that f is free of length-k monotone subsequences and the case that it is ε-far from free of length-k monotone subsequences.*

The above algorithmic formulation is equivalent to the following property testing problem (with one-sided error). For a given $2 \leq k \in \mathbb{N}$ and $f\colon [n] \to \mathbb{R}$, test whether there exists a decomposition of f into fewer than k non-increasing subsequences, or f is ε-far from having such a decomposition. The equivalence of the two formulations is a consequence of Dilworth's theorem [22]. One direction is trivial: if there exists a length-k increasing subsequence $(i_1, \ldots, i_k) \in [n]^k$, then any partition of f into fewer than k subsequences must contain two indices i_j and $i_{j'}$ within the same subsequence, hence, the subsequences are not non-increasing. The other direction follows from considering the poset $([n], \prec_f)$, where $i \prec_f j$ iff $i \leq j$ and $f(i) \geq f(j)$; every anti-chain of $\prec_f$ is an increasing subsequence of f, and every chain of $\prec_f$ is a non-increasing subsequence. If there are no length-k increasing subsequences, the maximum anti-chain of $\prec_f$ has size at most $k-1$, and by Dilworth's theorem, there is a partition of $([n], \prec_f)$ into at most $k-1$ chains, i.e., non-increasing subsequences.[1]

This paper gives an algorithm with optimal dependence in n for the above problems. We state the main theorem next, and discuss connections to monotonicity testing and to the longest increasing subsequence (LIS) problem shortly after.

▶ **Theorem 1.** *Fix $k \in \mathbb{N}$ and $\varepsilon > 0$. There exists an algorithm that, given query access to a function $f\colon [n] \to \mathbb{R}$ which is ε-far from free of length-k monotone subsequences, outputs a length-k monotone subsequence of f with probability $9/10$, with query complexity and running time of*

$$\left(k^k \cdot (\log(1/\varepsilon))^k \cdot \frac{1}{\varepsilon} \cdot \log(1/\delta)\right)^{O(k)} \cdot \log n.$$

Thus, for fixed k and ε, the query complexity and running time are of order $O(\log n)$. The above result can be stated analogously in the language of monotone decompositions.

▶ **Corollary 2.** *Fix $k \in \mathbb{N}$ and $\varepsilon > 0$. There is an algorithm with query complexity and running time $O(\log n)$ for ε-testing whether a sequence $f\colon [n] \to \mathbb{R}$ is decomposable into k monotone subsequences.*

The algorithm underlying Theorem 1 is *adaptive*[2] and solves the testing problem with one-sided error, since a length-k monotone subsequence is evidence for not being free of such subsequences. The algorithm improves on a recent result of Ben-Eliezer, Canonne, Letzter and Waingarten [7] who gave a non-adaptive algorithm for finding length-k monotone patterns with query complexity $O_{k,\varepsilon}((\log n)^{\lfloor \log_2 k \rfloor})$, which in itself improved upon a $O_{k,\varepsilon}((\log n)^{O(k^2)})$ upper bound by [45]. The focus of [7] was on *non-adaptive* algorithms, and they gave a

---

[1] A similar equivalence, between being decomposable into k increasing (or decreasing) subsequences and not containing non-increasing (or non-decreasing, respectively) patterns of length $k+1$ holds as well. We note that all results stated here in terms of "strong" monotonicity, e.g., being increasing, will also hold for their "weak" monotonicity analogue, e.g., being non-decreasing.

[2] An algorithm is *non-adaptive* if its queries do not depend on the answers to previous queries, or, equivalently, if all queries to the function can be made in parallel. Otherwise, if the queries of an algorithm may depend on the outputs of previous queries, then the algorithm is *adaptive*.

lower bound of $\Omega\big((\log n)^{\lfloor \log_2 k \rfloor}\big)$ queries for non-adaptive algorithms achieving one-sided error. Hence, Theorem 1 implies a natural separation between the power of adaptive and non-adaptive algorithms for finding monotone subsequences.

Theorem 1 is optimal, even among two-sided error algorithms. In the case $k = 2$, corresponding to monotonicity testing, there is a $\Omega(\log n/\varepsilon)$ lower bound (as long as, say, $\varepsilon > n^{-0.99}$) for both non-adaptive and adaptive algorithms [25, 27, 15], even with two-sided error. A simple reduction suggested in [45] shows that the same lower bound (up to a multiplicative factor depending on k) holds for any fixed $k \geq 2$. Thus, an appealing consequence of Theorem 1 is that the natural generalization of monotonicity testing, which considers forbidden monotone patterns of fixed length longer than 2, does not affect the dependence on n in the query complexity by more than a constant factor. Interestingly, [27] shows that for any adaptive algorithm for monotonicity testing on $f\colon [n] \to \mathbb{R}$ there is a non-adaptive algorithm which is at least as good in terms of query complexity (even if we only restrict ourselves to one-sided error algorithms). That is, adaptivity does not help at all for $k = 2$. In contrast, the separation between our $O(\log n)$ adaptive upper bound and the $\Omega\big((\log n)^{\lfloor \log_2 k \rfloor}\big)$ non-adaptive lower bound of [7] means this is no longer true for $k \geq 4$.

While our work settles the dependence in n in the query complexity of adaptive monotone pattern testing, and [7] settles the non-adaptive dependence in n, the following interesting question remains wide open.

▶ **Question 3.** *What is the optimal dependence of the query complexity in k and ε for the monotone subsequence testing problem discussed in this paper?*

Thus far, all known (adaptive and non-adaptive) results on this problem have a $k^{O(k^2)}$ type dependence in k in the query complexity; see Theorem 3.1 in [45] and Lemma 3.2 in [7]. The best known dependence in ε is of the form $(1/\varepsilon)^{\log_2 k + O(1)}$ for fixed k [7].

On the role of adaptivity in order pattern detection

Harnessing adaptivity to improve algorithmic performance is a notoriously difficult problem in many branches of property testing, typically requiring a good structural understanding of the task at hand. In the context of testing for forbidden order patterns, non-adaptive algorithms are quite weak: the non-adaptive query complexity is $\Omega(n^{1/2})$ for all non-monotone order patterns [45], and as high as $n^{1-1/(k-\Theta(1))}$ for almost all patterns of length k [6]. A recent (and independent) work of [47] gave new adaptive algorithms for general patterns with query complexity $n^{o(1)}$ for fixed constant $\varepsilon > 0$ and $k \in \mathbb{N}$, showing that for non-monotone patterns, too, adaptive algorithms may significantly improve upon non-adaptive ones. We note that the query complexity obtained in [47] is not polylogarithmic in n, and so their result is incomparable to ours. Their proof techniques are also very different from ours: at the core of their proof is a sophisticated sparsification framework, which makes use of a beautiful result of Marcus and Tardos [40] on pattern-avoidance in matrices.

Connections to the Longest Increasing Subsequence (LIS) problem

As an immediate consequence, Theorem 1 gives an optimal testing algorithm for the longest increasing subsequence (LIS) problem in a certain regime. The classical LIS problem asks to determine, given a sequence $f\colon [n] \to \mathbb{R}$, the maximum k for which f contains a length-k increasing subsequence. It is very closely related to other fundamental algorithmic problems in sequences, such as computing the edit distance, Ulam distance, or distance from monotonicity (for example, the latter equals n minus the LIS length), and has been thoroughly investigated from the perspective of classical algorithms [29, 50], sublinear-time algorithms [49, 1, 54, 52, 46, 42, 2], streaming algorithms [34, 56, 30, 53, 24, 43], dynamic algorithms [18, 31, 39, 41] and massively parallel computation [36, 12]. In the property testing regime, the corresponding decision task is to distinguish between the case where f has LIS length at most k (where k is given as part of the input) and the case that f is ε-far from having such a LIS length. Theorem 1 in combination with the aforementioned $\Omega(\log n)$ lower bounds (which readily carry over to this setting) yield a tight bound on the query complexity of testing whether the LIS length is a constant.

► **Corollary 4.** *Fix $2 \leq k \in \mathbb{N}$ and $\varepsilon > 0$. The query complexity of ε-testing whether $f\colon [n] \to \mathbb{R}$ has LIS length at most k is $\Theta(\log n)$.*

1.1 Related Work

Considering general permutations π of length k and *exact* computation, [35] showed how to find a π-pattern in a sequence f in time $2^{O(k^2 \log k)}n$, later improved by [28] to $2^{O(k^2)}n$. In the regime $k = \Omega(\log n)$, an algorithm of [8] running in time $n^{k/4+o(k)}$ provides the state-of-the-art. The analogous *counting* problem has also been actively studied, see [26] and the references within.

For *approximate* computation of general patterns π, the works of [45, 6] investigate the query complexity of property testing for forbidden order patterns. When π is of length 2, the problem considered is equivalent to testing monotonicity, one of the most widely-studied problems in property testing, with works spanning the past two decades. Over the years, variants of monotonicity testing over various partially ordered sets have been considered, including the line $[n]$ [25, 27, 3, 48, 5], the Boolean hypercube $\{0,1\}^d$ [23, 10, 13, 14, 20, 19, 37, 4, 16, 21, 17], and the hypergrid $[n]^d$ [11, 15, 9]. We refer the reader to [32, Chapter 4] for more on monotonicity testing, and a general overview of the field of property testing (introduced in [51, 33]).

1.2 Main Ideas and Techniques

We now describe some intuition behind the proof of Theorem 1. We note that the algorithm considers several cases and combines ideas from [45] and [7] with new structural and algorithmic components. In this overview, technical details established in [45] and [7] are noted but excluded; the purpose is to highlight the challenges and novel ideas arising specifically from this work. (See the Appendix in the full-version of the work for a short technical overview of these previous results.)

Fix $k \in \mathbb{N}$ and $\varepsilon > 0$, and suppose that $f\colon [n] \to \mathbb{R}$ is ε-far from $(12\ldots k)$-free, that is, ε-far from free of length-k increasing subsequences. Notice that f must contain a collection $\mathcal{C}$ of at least $\varepsilon n/k$ pairwise-disjoint increasing subsequences of length k; indeed, otherwise, greedily eliminating these subsequences gives a $(12\ldots k)$-free function differing in strictly fewer than εn inputs.

For simplicity in this overview, assume that k is even and that all $\varepsilon n/k$ length-k increasing subsequences of f in $\mathcal{C}$, $(x_1, x_2, \ldots, x_k) \in [n]^k$, satisfy that $|x_{k/2+1} - x_{k/2}| \geq |x_{i+1} - x_i|$ for all $i \in [k-1]$ (the non-adaptive lower bound of $\Omega_\varepsilon((\log n)^{\lfloor \log_2 k \rfloor})$ holds even in this restricted case) – intuitively, the largest "gap" in successive indices is between the $k/2$-th and $(k/2+1)$-th position. A goal, common to [45, 7] and this work, is to recursively find a $(12\ldots k/2)$-pattern of indices $(i_1, \ldots, i_{k/2}) \in [n]^{k/2}$, as well as $(12\ldots k/2)$-pattern of indices $(i_{k/2+1}, \ldots, i_k) \in [n]^{k/2}$ that can be combined into one $(12\ldots k)$-pattern. Toward

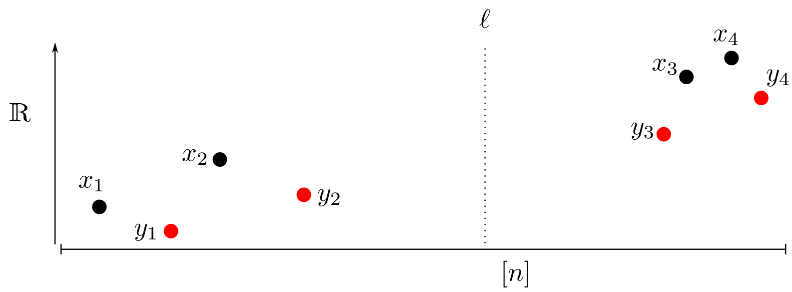

Figure 1 A sequence $f\colon [n] \to \mathbb{R}$ with two disjoint monotone subsequences of length 4, and an index $\ell \in [n]$. The sequences are $x = (x_1, x_2, x_3, x_4)$ and $y = (y_1, y_2, y_3, y_4)$. Note that both x and y have the largest gap between consecutive elements at index 2, i.e., $|x_3 - x_2|$ and $|y_3 - y_2|$ are the largest gaps between consecutive indices in x and y. Furthermore, ℓ cuts both x and y with slack.

this recursive approach, we say that an index $\ell \in [n]$ *cuts* $(x_1, \dots, x_k)$ *with slack* if

$$x_{k/2} + \frac{x_{k/2+1} - x_{k/2}}{3} \ \le\ \ell\ \le\ x_{k/2+1} - \frac{x_{k/2+1} - x_{k/2}}{3},$$

or, informally, if ℓ lies “roughly in the middle” between $x_{k/2}$ and $x_{k/2+1}$ – which, by the above assumption, form the largest gap among consecutive indices of the increasing subsequence (see Figure 1). The index $\ell \in [n]$ allows us to recurse on an interval before ℓ, as well as an interval after ℓ. Additionally, the *width* of $(x_1, \dots, x_k)$ is set to be $\lfloor \log(x_{k/2+1} - x_{k/2}) \rfloor$. We consider the subset of $\mathcal{C}$ consisting of length-k monotone subsequences of width w which are cut by ℓ with slack,

$$\mathcal{C}_{\ell,w} = \{(x_1, \dots, x_k) \in \mathcal{C} : \mathrm{width}(x_1, \dots, x_k) = w,\ \ell \text{ cuts } (x_1, \dots, x_k) \text{ with slack}\},$$

and note that if $(x_1, \dots, x_k) \in \mathcal{C}_{\ell,w}$, then $x_1, \dots, x_{k/2} \in [\ell - k \cdot 2^w, \ell]$ and $x_{k/2+1}, \dots, x_k \in [\ell, \ell + k \cdot 2^w]$, since $|x_{k/2+1} - x_{k/2}|$ was maximal. Motivated by this observation, the *density* of width-w copies in $\mathcal{C}$ around ℓ is measured by

$$\tau_{\mathcal{C}}(\ell, w) = \frac{1}{2^w} \cdot |\mathcal{C}_{\ell,w}|,$$

and the total density (over all widths) of $\mathcal{C}$ around ℓ is measured by

$$\tau_{\mathcal{C}}(\ell) = \sum_{w=1}^{\log n} \tau_{\mathcal{C}}(\ell, w).$$

The algorithms (ours and those in [45, 7]) proceed in a recursive manner. Each step considers an index $\ell \in [n]$ where the total density $\tau_{\mathcal{C}}(\ell)$ is high, namely at least $\Omega_k(\varepsilon)$, as well as a width w where $\tau_{\mathcal{C}}(\ell, w)$ is high. At a very high level, the algorithm can recurse on the sub-intervals $[\ell - k \cdot 2^w, \ell]$ and $[\ell, \ell + k \cdot 2^w]$, where the lower bound on $\tau_{\mathcal{C}}(\ell, w)$ implies sufficiently many increasing subsequences exist in each interval. If we choose the index ℓ and width w correctly, we have reduced the problem of finding a $(12 \dots k)$-pattern to finding two $(12 \dots k/2)$-patterns in subsequences of size $k \cdot 2^w$ to the left and right of ℓ which are themselves $\Omega_{\varepsilon,k}(1)$-far from free of $(12 \dots, k/2)$-patterns.

While ℓ may be chosen randomly, choosing the correct width w becomes analytically trickier, and is the step where the algorithms differ. The number of possible widths w is $\Theta(\log n)$ (since these are powers of 2 between 1 and n), and a *non-adaptive* algorithm cannot know what a correct choice of w is. The non-adaptive algorithms consider all $\Theta(\log n)$ options and recursively apply the algorithm for each width, thereby losing a $\Theta(\log n)$ factor in the query complexity at each recursive step. The main challenge of [45, 7] is obtaining the "best" lower bound on $\tau_{\mathcal{C}}(\ell, w)$ for some $w \in [\log n]$ and determining the number of recursive steps necessary. The fact that a non-adaptive algorithm must explore $\Omega(\log n)$ widths is inevitable, and what the non-adapive lower bound in [7] formalizes.

With adaptivity, the hope is that an algorithm considering an index $\ell \in [n]$ with $\tau_{\mathcal{C}}(\ell) = \Omega_k(\varepsilon)$ can choose *one* width w satisfying $\tau_{\mathcal{C}}(\ell, w) = \Omega_k(\varepsilon)$, and recurse only on that width. The algorithm may devote $\Theta_{k,\varepsilon}(\log n)$ queries to consider all $\Theta(\log n)$ possible widths, and the benefit is that recursing on a single width incurs a $\Theta_{k,\varepsilon}(\log n)$ *additive* loss in the query complexity, as opposed to the $\Theta_{k,\varepsilon}(\log n)$ multiplicative loss incurred by [45, 7]. We describe how we accomplish this next.

First, there is a simple $O_{k,\varepsilon}(\log n)$-query procedure which can choose a width $\widehat{w}$ where $\widehat{w} \geq w$. For example, for every possible width w_0, the algorithm queries $O_{k,\varepsilon}(1)$ randomly sampled indices from $[\ell - k \cdot 2^{w_0}, \ell]$ and $[\ell, \ell + k \cdot 2^{w_0}]$. Then, let $\widehat{w}$ be the largest w_0 where some increasing pair is found. The fact that the unknown $w \in [\log n]$ satisfies $\tau_{\mathcal{C}}(\ell, w) \geq \Omega_k(\varepsilon)$ implies that with high constant probability, there exists two $(x_1, \ldots, x_k), (y_1, \ldots, y_k) \in \mathcal{C}_{\ell,w}$ where indices x_1 and y_k are sampled and by an observation from [45], with high enough probability, $f(x_1) \leq f(y_k)$ (see the appendix in the full-version for a more thorough discussion on this point). This, in turn, implies $\widehat{w} \geq w$.

If the simple procedure happened to choose $\widehat{w}$ which is not much larger than w, then we may recurse on $\widehat{w}$, similarly to [45, 7]; we call this the *fitting* case. The problem is that $\widehat{w}$ may be too large, a case we refer to as *overshooting*. Consider the execution selecting a width $\widehat{w}$ which is too large, in particular, the "correct" width w satisfies $w \ll \widehat{w}$. Intuitively, the problem is the following: the promise that $\tau_{\mathcal{C}}(\ell, w) \geq \Omega_k(\varepsilon)$ ensures that the subsequence $[\ell - k \cdot 2^w, \ell + k \cdot 2^w]$ is sufficiently dense with $(12 \ldots k)$-patterns; however, when $\widehat{w}$ is much larger, the subsequence $[\ell - k \cdot 2^{\widehat{w}}, \ell + k \cdot 2^{\widehat{w}}]$ is much larger than the subsequence $[\ell - k \cdot 2^w, \ell + k \cdot 2^w]$; hence, the length-$k$ increasing subsequences in $[\ell - k \cdot 2^w, \ell + k \cdot 2^w]$ constitute a tiny (at most $O_k(2^{w - \widehat{w}})$) fraction of the interval $[\ell - k \cdot 2^{\widehat{w}}, \ell + k \cdot 2^{\widehat{w}}]$ the algorithm would recurse on.

Due to the density $\tau_{\mathcal{C}}(\ell, \widehat{w})$ being potentially very small, at this point, it is not clear how to proceed with our wrong (too large) choice of $\widehat{w}$ as the width to recurse on. To overcome this, we prove a robust structural theorem, drawing a much more favorable picture as to which widths are good for recursion. The robust structural theorem asserts the following. For sufficiently many possible $\ell \in [n]$ and widths w where $\tau_{\mathcal{C}}(\ell, w) \geq \Omega_k(\varepsilon)$, *every* interval J containing $[\ell - k \cdot 2^w, \ell + k \cdot 2^w]$ has $\Omega_k(\varepsilon|J|)$ pairwise-disjoint length-k increasing subsequences. At a high level, the prior structural results ensured that $[\ell - k \cdot 2^w, \ell + k \cdot 2^w]$ is dense with $(12 \ldots k)$-patterns cut by ℓ; our robust version ensures that any interval J containing $[\ell - k \cdot 2^w, \ell + k \cdot 2^w]$ remains dense with $(12 \ldots k)$-patterns. In particular, the choice of interval is robust to picking a width $\widehat{w}$ which is larger than w. These length-k increasing subsequences are not cut with slack by ℓ, a condition which was crucial for [45, 7]; however, the algorithm's choice of $\widehat{w}$ means it found an increasing pair at distance $\Theta_k(2^{\widehat{w}})$. We exploit this with an adaptive algorithm in a somewhat surprising manner, which we expand on now.

New algorithm when overshooting

Let $\ell \in [n]$ be an index with $\tau_{\mathcal{C}}(\ell) \geq \Omega_k(\varepsilon)$, and let w be the unknown width where $\tau_{\mathcal{C}}(\ell, w) \geq \Omega_k(\varepsilon)$ with the above-mentioned robustness property. Suppose that the widest increasing pair $(\boldsymbol{x}, \boldsymbol{y})$ found by the algorithm (which sets $\widehat{w} \approx \log_2 |\boldsymbol{y} - \boldsymbol{x}|$), satisfies $\widehat{w} \gg w$. Even though the algorithm has "committed" to a width $\widehat{w}$ which is too large, we will algorithmically exploit the fact that $(\boldsymbol{x}, \boldsymbol{y})$ is an increasing pair lying very far apart, and containing the interval $[\ell - k \cdot 2^w, \ell + k \cdot 2^w]$. Specifically, since $(\boldsymbol{x}, \boldsymbol{y})$ are very far away, the algorithm may fit $k-2$ intervals $J_1, \ldots, J_{k-2}$ between $\boldsymbol{x}$ and $\boldsymbol{y}$ which lie adjacent to each other, satisfying the following conditions:

- J_1 contains the interval $[\ell - k \cdot 2^w, \ell + k \cdot 2^w]$.
- J_{i+1} lies immediately after J_i, for any $i \in [k-3]$.
- $|J_{i+1}| \geq |J_i| \cdot \alpha_{k,\varepsilon}$ for all $i \in [k-3]$, and a large fixed constant $\alpha_{k,\varepsilon} > 1$.

A consequence of the robust structural theorem, and the fact that $J_1, \ldots, J_{k-2}$ have exponentially increasing lengths is that each J_i contains a collection $\mathcal{T}_i$ of $\Omega_k(\varepsilon|J_i|)$ disjoint length-k increasing subsequences. For each $i \in [k-2]$, define two sets $\mathcal{A}_i$ and $\mathcal{B}_i$ as follows. Let $\mathcal{A}_i$ be the collection of prefixes $(a_1, \ldots, a_{i+1})$ of $\mathcal{T}_i$ with $f(a_{i+1}) < f(\boldsymbol{y})$, and let $\mathcal{B}_i$ be the collection of suffixes $(a_{i+1}, \ldots, a_k)$ of $\mathcal{T}_i$ with $f(a_{i+1}) \geq f(\boldsymbol{y})$. As $|\mathcal{T}_i| = |\mathcal{A}_i| + |\mathcal{B}_i|$, one of $\mathcal{A}_i$ and $\mathcal{B}_i$ is large (i.e. has size at least $\Omega_k(\varepsilon|J_i|)$). This seemingly innocent combinatorial idea can be exploited non-trivially to find an increasing subsequence of length k. Specifically, the algorithm to handle overshooting aims to (recursively) find shorter increasing subsequences in $J_1, \ldots, J_{k-2}$, with the hope of combining them together into an increasing subsequence of length k. Concretely, for any $i \in [k-2]$, we make two recursive calls of our algorithm on J_i: one for an $(i+1)$-increasing subsequence in J_i, with values smaller than $f(\boldsymbol{y})$,[3] and a second one for a $(k-i)$-increasing subsequence in J_i whose values are at least $f(\boldsymbol{y})$. By induction, the first recursive call succeeds with good probability if $|\mathcal{A}_i|$ is large, while the second call succeeds with good probability if $|\mathcal{B}_i|$ is large. Since for any i either $|\mathcal{A}_i|$ or $|\mathcal{B}_i|$ must be large, at least one of the following must hold.

- $\mathcal{B}_1$ is large. In this case we are likely to find a length-$(k-1)$ monotone pattern in J_1 with values at least $f(\boldsymbol{y}) > f(\boldsymbol{x})$, which combines with $\boldsymbol{x}$ to form a length-k monotone pattern.
- $\mathcal{A}_{k-2}$ is large. Here we are likely to find a length-$(k-1)$ monotone pattern in J_{k-2} whose values lie below $f(\boldsymbol{y})$, which combines with $\boldsymbol{y}$ to form a length-k monotone pattern.
- There exists $i \in [k-3]$ where both $\mathcal{A}_i$ and $\mathcal{B}_{i+1}$ are large. Here we will find, with good probability, a length-$(i+1)$ monotone pattern in J_i with values below $f(\boldsymbol{y})$, and a length-$(k-i-1)$ monotone pattern in J_{i+1} with values at or above $f(\boldsymbol{y})$; together these two patterns combine to form a $(12\ldots k)$-pattern.

In all cases, a k-increasing subsequence is found with good probability. See Figure 2 for an example. The benefit is that the algorithm spends $\Theta_{k,\varepsilon}(\log n)$ queries to identify one fixed width $\widehat{w} \in [\log n]$. Then, there are $2(k-2)$ recursive calls each aiming to find an increasing subsequence of length strictly less than k. The $\Theta_{\varepsilon,k}(\log n)$ loss in the query complexity is additive per recursive step; this leads to the $\Theta_{\varepsilon,k}(\log n)$ query complexity bound which was impossible in the non-adaptive algorithms of [45, 7], as these had to explore all possible widths $\widehat{w} \in [\log n]$ in each recursive step.

Organization

The rest of the paper is organized as follows. Relevant notation can be found in Section 1.3. Section 2 establishes the stronger structural result required for our adaptive algorithm. Section 3 contains the new algorithmic components and the formal statements regarding the correctness of our algorithm and its query complexity. The appendices in the full-version provide a brief description of the previous (non-adaptive) testing results on $(12\ldots k)$-freeness

[3] Technically speaking, our algorithm can be configured to only look for increasing subsequences whose values lie in some range; we use this to make sure that shorter increasing subsequences obtained from the recursive calls of the algorithm can eventually be concatenated into a valid length-k one.

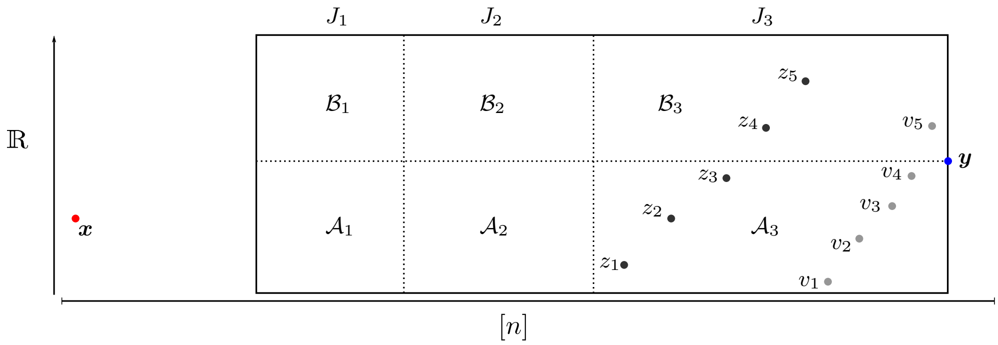

Figure 2 We consider the "overshooting case" for $k = 5$. Specifically, the algorithm considers an index $\ell \in [n]$ with $\tau_C(\ell) = \Omega_k(\varepsilon)$ and, for some unknown $w \in [\log n]$, $\tau_C(\ell, w) = \Omega_k(\varepsilon)$. Furthermore, in trying to identify a correct width $\widehat{w}$, the algorithm samples an increasing pair $(\boldsymbol{x}, \boldsymbol{y})$ with $\log_2 |\boldsymbol{x} - \boldsymbol{y}| \approx \widehat{w} \gg w$. The algorithm will consider at least $k - 2$ geometrically increasing intervals between $\boldsymbol{x}$ and $\boldsymbol{y}$; these are displayed as J_1, J_2, and J_3; by virtue of the robust structural theorem, each J_i contains $\Omega_k(\varepsilon|J_i|)$ disjoint length-k monotone subsequences. $\mathcal{A}_i$ contains those length-k monotone subsequences where the $(i+1)$-th index is above $f(\boldsymbol{y})$ and $\mathcal{B}_i$ contains those whose $(i+1)$-th index is below $f(\boldsymbol{y})$. As an example, $(z_1, z_2, z_3, z_4, z_5) \in \mathcal{B}_4$ and $(v_1, v_2, v_3, v_4, v_5) \in \mathcal{A}_4$. The crucial properties are: (i) for all $i \in [k-2]$ any $(12\ldots i)$-pattern in $\mathcal{A}_i$ and any $(12\ldots(k-i))$-pattern in $\mathcal{B}_{i+1}$ may be combined into a $(12\ldots k)$-pattern, (ii) any $(12\ldots(k-1))$-pattern in $\mathcal{B}_1$ may be combined with $\boldsymbol{x}$ since $f(\boldsymbol{y}) > f(\boldsymbol{x})$, and (iii) any $(12\ldots(k-1))$-pattern in $\mathcal{A}_4$ may be combined with $\boldsymbol{y}$. The reasoning may proceed as follows: if $|\mathcal{B}_1|$ is large, we find a $(12\ldots(k-1))$-pattern and combine it with $\boldsymbol{x}$; so, assume $|\mathcal{B}_1|$ is small, which implies $|\mathcal{A}_1|$ must be large. If $|\mathcal{B}_2|$ is large, then a (12)-pattern from $\mathcal{A}_1$ and a $(12\ldots(k-2))$-pattern from $\mathcal{B}_2$ may be combined; so assume $|\mathcal{B}_2|$ is small which implies $|\mathcal{A}_2|$ is large, ... Eventually, we deduce that we may assume $|\mathcal{A}_{k-2}|$ is large, and a $(12\ldots(k-1))$-pattern in $\mathcal{A}_{k-2}$ may be combined with $\boldsymbol{y}$.

from [45, 7], as well as the remaining proofs, relegated from the main body due to space constraints.

1.3 Notation

All logarithms considered are base 2. We consider functions $f\colon I \to \mathbb{R}$, where $I \subseteq [n]$, as the inputs and main objects of study. An *interval* in $[n]$ is a set $I \subseteq [n]$ of the form $I = \{a, a+1, \ldots, b\}$. At many places throughout the paper, we think of augmenting the image with a special character $*$ to consider $f\colon I \to \mathbb{R} \cup \{*\}$. The character $*$ can be thought of as a *masking* operation: In many cases, we will only be interested in entries x of f so that $f(x)$ lies in some prescribed (known in advance) range of values $R \subseteq \mathbb{R}$, so that entries outside this range will be marked by $*$. Whenever the algorithm queries $f(x)$ and observes $*$, it will interpret this as an incomparable value (with respect to ordering) in $\mathbb{R}$. As a result, $*$-values will never be part of monotone subsequences. We note that augmenting the image with $*$ was unnecessary in [45, 7] because they only considered non-adaptive algorithms. We say that for a fixed $f\colon I \to \mathbb{R} \cup \{*\}$, the set T is a collection of disjoint monotone subsequences of length k if it consists of tuples $(i_1, \ldots, i_k) \in I^k$, where $i_1 < \cdots < i_k$ and $f(i_1) < \cdots < f(i_k)$ (in particular, $f(i_1), \ldots, f(i_k) \neq *$), and furthermore, for any two tuples $(i_1, \ldots, i_k)$ and $(i'_1, \ldots, i'_k)$, their intersection (as sets) is empty. We also denote $E(T)$ as the union of indices in k-tuples of T, i.e., $E(T) = \cup_{(i_1,\ldots,i_k)\in T}\{i_1, \ldots, i_k\}$. Finally, we let $\mathrm{poly}(\cdot)$ denote a large enough polynomial whose degree is (bounded by) a universal constant.

2 Stronger Structural Dichotomy

In this section, we prove a robust structural dichotomy for functions $f\colon [n] \to \mathbb{R}$ that are ε-far from $(12\ldots k)$-free, which strengthens the dichotomy proved in [7]. In their paper, it is shown that any f which is ε-far from $(12\ldots k)$-free satisfies at least one of two conditions: either f contains many *growing suffixes*, or it can be decomposed into *splittable intervals*. In Section 2.1, we define and describe these notions and state the original (non-robust) structural result from [7]. Then, in Section 2.2, we establish a substantially stronger structural dichotomy, better suited for our purposes. The proof of the stronger dichotomy combines the original one as a black-box with additional combinatorial ideas.

2.1 The Non-Robust Structural Decomposition

For completeness, we first introduce the non-robust structural result from [7]. As the formal definitions are somewhat complicated, we start with an informal description of the growing suffixes and splittable intervals conditions. For the purpose of this discussion, let $\mathcal{C}$ be any collection of $\Theta_{k,\varepsilon}(n)$ disjoint $(12\ldots k)$-copies in f. We use the notation from Section 1.2.

- **Growing suffixes:** there exist $\Omega_{k,\varepsilon}(n)$ values of $\ell \in [n]$ where $\tau_{\mathcal{C}}(\ell) \geq \Theta_k(\varepsilon)$ and $\tau_{\mathcal{C}}(\ell, w) \ll \tau_{\mathcal{C}}(\ell)$ for every $w \in [\log n]$. In words, many $\ell \in [n]$ are such that the sum of local densities, $\tau_{\mathcal{C}}(\ell)$, of $(12\ldots k)$-patterns in intervals of growing widths is not too small, and furthermore, the densities are not concentrated on any small set of widths w. Any such ℓ is said to be the starting point of a growing suffix.
- **Splittable intervals (non-robust):** there exist $c \in [k-1]$ and a collection of pairwise-disjoint intervals $I_1, \ldots, I_s \subset [n]$ with $\sum_{i=1}^{s} |I_i| = \Theta_{k,\varepsilon}(n)$, so that each I_i contains a dense collection of disjoint $(12\ldots k)$-patterns of a particular structure. Specifically, each such interval I_i can be partitioned into three disjoint intervals L_i, M_i, R_i (in this order), each of size $\Omega_k(|I_i|)$, where I_i fully contains $\Omega_{k,\varepsilon}(|I_i|)$ disjoint copies of $(12\ldots k)$-patterns, in which the first c entries lie in L_i, the last $k-c$ entries lie in R_i (none of these entries lies in M_i), and every such c entry lies below every $c+1$ entry.

Informally, the non-robust structural dichotomy from [7] asserts that any f that is ε-far from $(12\ldots k)$-free either satisfies the growing suffixes condition, or the non-robust splittable intervals condition (or both). These two notions are formally defined next; the precise definition for growing suffixes is slightly more complicated than described above (but understanding it is not essential for this work, as the growing suffixes procedure from [7] will eventually only be used as a black box). For what follows, for an index $\ell \in [n]$ define $\eta_\ell = \lceil \log_2(n-\ell) \rceil$, and for any $t \in [\eta_\ell]$ set $S_t(\ell) = [a + 2^{t-1}, a + 2^t) \cap [n]$. Note that the intervals $S_1, \ldots, S_{\eta_\ell}$ are a partition of $(\ell, n]$ into intervals of geometrically increasing length (except for maybe the last one). Finally, the tuple $S(\ell) = (S_t(\ell))_{t \in [\eta_\ell]}$ is called the *growing suffix* starting at ℓ.

▶ **Definition 5** (Growing suffixes (see [7], Definition 2.4)). *Let $\alpha, \beta \in [0,1]$. We say that an index $\ell \in [n]$ starts an (α, β)-growing suffix if, when considering the collection of intervals $S(\ell) = \{S_t(\ell) : t \in [\eta_\ell]\}$, for each $t \in [\eta_\ell]$ there is a subset $D_t(\ell) \subseteq S_t(\ell)$ of indices such that the following properties hold.*

1. *We have $|D_t(\ell)|/|S_t(\ell)| \leq \alpha$ for all $t \in [\eta_\ell]$, and $\sum_{t=1}^{\eta_\ell} |D_t(\ell)|/|S_t(\ell)| \geq \beta$.*
2. *For every $t, t' \in [\eta_a]$ where $t < t'$, if $a \in D_t(\ell)$ and $a' \in D_{t'}(\ell)$, then $f(a) < f(a')$.*

The second definition, also from [7], describes the (non-robust) splittable intervals setting.

▶ **Definition 6** (Splittable intervals (see [7], Definition 2.5)). *Let* $\alpha, \beta \in (0,1]$ *and* $c \in [k-1]$. *Let* $I \subseteq [n]$ *be an interval, let* $T \subseteq I^k$ *be a set of disjoint, length-*k *monotone subsequences of* f *lying in* I, *and define*

$$T^{(L)} = \{(i_1, \ldots, i_c) \in I^c : (i_1, \ldots, i_c) \text{ is a prefix of a } k\text{-tuple in } T\}, \text{ and}$$
$$T^{(R)} = \{(j_1, \ldots, j_{k-c}) \in I^{k-c} : (j_1, \ldots, j_{k-c}) \text{ is a suffix of a } k\text{-tuple in } T\}.$$

We say that the pair (I, T) *is* (c, α, β)-splittable *if* $|T|/|I| \geq \beta$; $f(i_c) < f(j_1)$ *for every* $(i_1, \ldots, i_c) \in T^{(L)}$ *and* $(j_1, \ldots, j_{k-c}) \in T^{(R)}$; *and there is a partition of* I *into three consecutive intervals* $L, M, R \subseteq I$ *(that appear in this order, from left to right) of size at least* $\alpha|I|$, *satisfying* $T^{(L)} \subseteq L^c$ *and* $T^{(R)} \subseteq R^{k-c}$.

A collection of disjoint interval-tuple pairs $(I_1, T_1), \ldots, (I_s, T_s)$ *is called a* (c, α, β)-splittable collection of T *if each* (I_j, T_j) *is* (c, α, β)*-splittable and the sets* $(T_j : j \in [s])$ *partition* T.

The following theorem presents the growing suffixes versus (non-robust) splittable intervals dichotomy, which is among the main structural results of [7]. We remark that in their paper, the theorem is stated with respect to two parameters, k, k_0; for our purpose it suffices to set $k_0 = k$. Also, here we allow f to take the value $*$, which is not the case in [7]. Nevertheless, as their proof takes into account only the elements of a given family T^0 of disjoint length-k increasing subsequences, which in particular are non-$*$ elements, the same proof works here.

▶ **Theorem 7** ([7], Theorem 2.2). *Let* $k, n \in \mathbb{N}$, $\varepsilon \in (0,1)$, *and* $C > 0$, *and let* $I \subseteq [n]$ *be an interval. Let* $f\colon I \to \mathbb{R} \cup \{*\}$ *be a function and let* $T^0 \subseteq I^k$ *be a set of at least* $\varepsilon|I|$ *disjoint monotone subsequences of* f *of length* k. *Then there exist* $\alpha \in (0,1)$ *and* $p > 0$ *satisfying* $\alpha \geq \Omega(\varepsilon/k^5)$ *and* $p \leq \text{poly}(k \log(1/\varepsilon))$ *such that at least one of the following conditions holds.*

1. ***Growing suffixes:*** *There exists a set* $H \subseteq [n]$, *of indices that start an* $(\alpha, Ck\alpha)$*-growing suffix, satisfying* $\alpha|H| \geq (\varepsilon/p)n$.
2. ***Splittable intervals (non-robust):*** *There exist a positive integer* $c < k$, *a set* T *of disjoint length-*k *monotone subsequences satisfying* $E(T) \subseteq E(T^0)$, *and a* $(c, 1/(6k), \alpha)$*-splittable collection of* T *consisting of disjoint interval-tuple pairs* $(I_1, T_1), \ldots, (I_s, T_s)$, *such that* $\alpha \sum_{h=1}^{s} |I_h| \geq |T^0|/p$.

2.2 Robustifying the Structural Result

We are now ready to establish the robust structural foundations – specifically, a *growing suffixes* versus *robust splittable intervals* dichotomy – lying at the heart of our adaptive algorithm. The next lemma will eventually imply that the splittable intervals condition can be robustified by merely throwing away a subset of "bad" splittable intervals.

▶ **Lemma 8.** *Let* $\alpha \in (0,1)$ *and let* $I \subset \mathbb{N}$ *be an interval. Suppose that* $I_1, \ldots, I_s \subset I$ *are disjoint intervals such that* $\sum_{h=1}^{s} |I_h| \geq \alpha|I|$. *Then there exists a set* $G \subset [s]$ *such that* $\sum_{h \in G} |I_h| \geq (\alpha/4)|I|$, *and for every interval* $J \subset I$ *that contains an interval* I_h *with* $h \in G$, $\sum_{h \in [s]\colon I_h \subset J} |I_h| \geq (\alpha/4)|J|$.

The full proof appears in the Appendix of the full verion. The idea is to consider a minimal subset $\mathcal{J}$ of the collection of all "problematic" intervals J which *do not satisfy* the conditions of the lemma. For each $J \in \mathcal{J}$, less than an $\alpha/4$-fraction of J is covered by intervals from $\mathcal{I} = \{I_1, \ldots, I_s\}$. Conversely, as we show, the minimality of $\mathcal{J}$ entails that any element in I is covered by at most three intervals from $\mathcal{J}$. The combination of these conditions implies that, if we remove from $\mathcal{I}$ all intervals I_j contained in some interval

$J \in \mathcal{J}$, then at the end of the process $\sum_{I_j \in \mathcal{I}} |I_j| = \Omega(\alpha|I|)$, and no "problematic" choices of J survive. Thus, the set of surviving intervals from $\mathcal{I}$ satisfy the conditions of the lemma.

The robust version of the structural dichotomy is stated below; for the proof, combining the basic structural dichotomy with the last lemma, see the appendices of the full-version.

▶ **Theorem 9** (Robust structural theorem). *Let $k, n \in \mathbb{N}$, $\varepsilon \in (0,1)$, and $C > 0$, and let $I \subseteq [n]$ be an interval. Let $f: I \to \mathbb{R} \cup \{*\}$ be an array and let $T^0 \subseteq I^k$ be a set of at least $\varepsilon|I|$ disjoint length-k monotone subsequences of f. Then there exist $\alpha \in (0,1)$ and $p > 0$ with $\alpha \geq \Omega(\varepsilon/k^5)$ and $p \leq \text{poly}(k \log(1/\varepsilon))$ such that at least one of the following holds.*

1. ***Growing suffixes:*** *There exists a set $H \subseteq [n]$, of indices that start an $(\alpha, Ck\alpha)$-growing suffix, satisfying $\alpha|H| \geq (\varepsilon/p)n$.*
2. ***Robust splittable intervals:*** *There exist an integer c with $1 \leq c < k$, a set T, with $E(T) \subseteq E(T^0)$, of disjoint length-k monotone subsequences, and a $(c, 1/(6k), \alpha)$-splittable collection of T, consisting of disjoint interval-tuple pairs $(I_1, T_1), \ldots, (I_s, T_s)$, such that*

$$\alpha \sum_{h=1}^{s} |I_h| \geq \frac{\varepsilon}{p} \cdot |I|.$$

Moreover, if $J \subset I$ is an interval where $J \supset I_h$ for some $h \in [s]$, then J contains at least $(\varepsilon/p)|J|$ disjoint $(12 \ldots k)$-patterns from T^0.

3 The Algorithm

In this section we prove the existence of a randomized algorithm, `Find-Monotone`$_k(f, \varepsilon, \delta)$, that receives as input a function $f: I \to \mathbb{R} \cup \{*\}$ (where $I \subset \mathbb{N}$ is an interval), and parameters $\varepsilon, \delta \in (0,1)$, and satisfies the following: if f contains $\varepsilon|I|$ disjoint $(12 \ldots k)$-patterns, then the algorithm outputs such a pattern with probability at least $1 - \delta$. The running time is $O_{k,\varepsilon}(\log n)$. The algorithm is described in Figure 5 below. It uses three subroutines: `Sample-Suffix`, `Find-Within-Interval`, and `Find-Good-Split`, the first of which is given in [7], and the latter two are described below, in Figures 3 and 4. The majority of the section is devoted to the proof that `Find-Monotone` indeed outputs a $(12 \ldots k)$-pattern with high probability as claimed. Specifically, we shall prove the following theorem.

▶ **Theorem 10.** *Let $k \in \mathbb{N}$. The randomized algorithm* `Find-Monotone`$_k(f, \varepsilon, \delta)$*, described in Figure 5, satisfies the following. Given a function $f: I \to \mathbb{R} \cup \{*\}$ and parameters $\varepsilon, \delta \in (0,1)$, if f contains at least $\varepsilon|I|$ disjoint $(12 \ldots k)$-patterns, then* `Find-Monotone`$_k(f, \varepsilon, \delta)$ *outputs a $(12 \ldots k)$-pattern of f with probability at least $1 - \delta$.*

Our proof proceeds by induction on k. It relies on Lemmas 12, 13, 14, the former is taken from [7] whereas the proofs of the latter two assume that Theorem 10 holds for smaller k. We first state and prove these lemmas, and then we prove Theorem 10.

To complete the picture, in the following lemma we provide an upper bound on the query complexity and running time of `Find-Monotone`. For the proof, see the appendices of the full version.

▶ **Lemma 11.** *Let $f: I \to \mathbb{R} \cup \{*\}$, where I is an interval of length at most n. The query complexity and running time of* `Find-Monotone`$_k(f, \varepsilon, \delta)$ *are at most*

$$\left(k^k \cdot (\log(1/\varepsilon))^k \cdot \frac{1}{\varepsilon} \cdot \log(1/\delta)\right)^{O(k)} \cdot \log n.$$

3.1 The `Sample-Suffix` Sub-Routine

We restate Lemma 3.1 from [7] which gives the $\texttt{Sample-Suffix}_k$ subroutine, with a few adaptations to fit our needs.

▶ **Lemma 12** ([7]). *Fix $k \in \mathbb{N}$ and let $C > 0$ be a large enough constant. There exists a non-adaptive and randomized algorithm, $\texttt{Sample-Suffix}_k(f, \varepsilon, \delta)$ which takes three inputs: query access to a function $f : I \to \mathbb{R} \cup \{*\}$, where $I \subset \mathbb{N}$ is an interval, a parameter $\varepsilon \in (0, 1)$, and an error probability bound $\delta \in (0, 1)$. Suppose there exists $\alpha \in (0, 1)$, and a set $H \subseteq I$ of $(\alpha, Ck\alpha)$-growing suffixes of f satisfying $\alpha|H| \geq \varepsilon|I|$. Then, $\texttt{Sample-Suffix}_k(f, \varepsilon, \delta)$ finds a length-k monotone subsequence of f with probability at least $1 - \delta$. The query complexity of $\texttt{Sample-Suffix}_k(f, \varepsilon, \delta)$ is at most $\log(1/\delta) \cdot \mathrm{polylog}(1/\varepsilon) \cdot \frac{1}{\varepsilon} \cdot \log n$.*

For additional technical remarks about Lemma 12 and `Sample-Suffix`, see the appendices of the full versions.

3.2 Handling Overshooting: The `Find-Within-Interval` Sub-Routine

In this section, we describe the `Find-Within-Interval` subroutine, addressing the overshooting case as explained in Section 1.2. As the algorithm may appear unintuitive, let us

Subroutine $\texttt{Find-Within-Interval}_k(f, \varepsilon, \delta, x, y, \mathcal{J})$.

Input: Query access to a function $f : I \to \mathbb{R} \cup \{*\}$, parameters $\varepsilon, \delta \in (0, 1)$, two inputs $x, y \in I$ where $x < y$ and $f(x) < f(y)$, and $\mathcal{J} = (J_1, \ldots, J_{k-2})$ which is a collection of disjoint intervals appearing in order inside $[x, y]$.

Output: a sequence $i_1 < \ldots < i_k$ with $f(i_1) < \ldots < f(i_k)$, or fail.

1. For every $\kappa \in [k-2]$, let $f_\kappa, f'_\kappa : J_\kappa \to \mathbb{R} \cup \{*\}$ be given by:

$$f_\kappa(i) = \begin{cases} f(i) & f(i) < f(y) \\ * & \text{o.w.} \end{cases} \quad \text{and} \quad f'_\kappa(i) = \begin{cases} f(i) & f(i) \geq f(y) \\ * & \text{o.w.} \end{cases} \tag{1}$$

2. Call $\texttt{Find-Monotone}_{\kappa+1}(f_\kappa, \varepsilon/2, \delta/(2k))$ for every $\kappa \in [k-2]$.
3. Call $\texttt{Find-Monotone}_{k-\kappa}(f'_\kappa, \varepsilon/2, \delta/(2k))$ for every $\kappa \in [k-2]$.
4. Consider the set of all indices that are output in Lines 2 and 3, together with x and y. If $\mathcal{S}$ contains a length-k increasing subsequence among these indices, output it. Otherwise, output fail.

Figure 3 Description of the `Find-Within-Interval` subroutine.

remind the reader of the setup in which this subroutine is relevant (see also Section 1.2). By Theorem 9, either the growing suffixes condition or the splittable intervals condition hold. The former case is handled by Lemma 12, so we assume that the latter holds. Now assume that we sampled an element $\boldsymbol{x}$ which is the first element of a length-c increasing subsequence from a set L_i as described in Definition 6. We then sample, uniformly at random, elements $\boldsymbol{y}$ from $[\boldsymbol{x}, \boldsymbol{x} + 2^t]$. The splittable intervals condition implies that we will find, with high probability, an element $\boldsymbol{y}$ which is the last element of a length-$(k-c)$ increasing subsequence from R_i. In particular, $f(\boldsymbol{y}) > f(\boldsymbol{x})$. However, even if we did indeed sample such $\boldsymbol{y}$, we may have sampled many other values of $\boldsymbol{y}'$ with $f(\boldsymbol{y}') > f(\boldsymbol{x})$, and we do not know of a way of determining which of these values is the "correct" one. Instead, we take $\boldsymbol{y}_0$ to be the largest sampled $\boldsymbol{y}'$ such that $f(\boldsymbol{y}') > f(\boldsymbol{x})$. The case where $\boldsymbol{y}_0$ is close to $\boldsymbol{y}$ is taken care of by Lemma 13, so we assume that $\boldsymbol{y}_0$ is much larger than $\boldsymbol{y}$.

We now have elements $\boldsymbol{x}$ and $\boldsymbol{y}_0$, and all that we know is that they contain a large portion of an interval I_i from the splittable intervals condition. It is not hard to see (this is shown in the proof of Theorem 10) that $[\boldsymbol{x}, \boldsymbol{y}_0]$ can be partitioned into $k-2$ intervals $J_1, \ldots, J_{k-2}$, each of which contains many disjoint length-k increasing subsequences. To continue, out only hope is use the induction hypothesis to find shorter increasing subsequences in the intervals. For example, if there are many disjoint length-$(k-1)$ increasing subsequences in J_1 that lie above $\boldsymbol{x}$, then one such subsequence is likely to be detected by a recursive call to the main algorithm, and together with $\boldsymbol{x}$ it will form a length-k increasing subsequence. If there are few such length-$(k-1)$ subsequences, this means that there are many disjoint length-2 increasing subsequences in J_1 that lie below $\boldsymbol{x}$ (because for every length-k increasing subsequence, either its $(k-1)$-suffix lies above $\boldsymbol{x}$, or its 2-prefix lies above $\boldsymbol{x}$). We can then use a recursive call to detect such a sequence, and hope to complete it to a length-k subsequence using a length-$(k-2)$ subsequnece from J_2 that lies above $\boldsymbol{x}$. Continuing with this logic, it follows that with high probability we can find an increasing subsequence of length k using $\boldsymbol{x}$ and J_1, J_i and J_{i+1} for some i, or J_{k-2} and $\boldsymbol{y}_0$.

▶ **Lemma 13.** *Consider the randomized algorithm,* `Find-Within-Interval`$_k(f, \varepsilon, \delta, x, y, \mathcal{J})$, *described in Figure 3, which takes six inputs:*

- *Query access to a function $f\colon I \to \mathbb{R} \cup \{*\}$,*
- *Two parameters $\varepsilon, \delta \in (0, 1)$,*
- *Two points $x, y \in I$ where $x < y$ and $f(x) < f(y)$, and*
- *A collection $\mathcal{J} = (J_1, \ldots, J_{k-2})$ of $k-2$ disjoint intervals that appear in order (i.e., J_κ comes before $J_{\kappa+1}$) within the interval $[x, y]$,*

and outputs either a length-k increasing subsequence of f, or fail.

Suppose that for every $\kappa \in [k-2]$, the function $f|_{J_\kappa}\colon J_\kappa \to \mathbb{R} \cup \{*\}$, contains $\varepsilon|J_\kappa|$ disjoint $(12\ldots k)$-patterns. Then, assuming that Theorem 10 holds for every k' with $1 \le k' < k$, the procedure `Find-Within-Interval`$_k(f, \varepsilon, \delta, x, y, \mathcal{J})$ *outputs a length-k monotone subsequence of f with probability at least $1 - \delta$.*

The full proof appears in the appendices of the full version.

3.3 Handling the Fitting Case: The `Find-Good-Split` Sub-Routine

In this section, we describe the `Find-Good-Split` subroutine, which corresponds to the fitting case from Section 1.2. The proof of the lemma below appears in the appendices of the full version.

▶ **Lemma 14.** *Consider the randomized algorithm* `Find-Good-Split`$_k(f, \varepsilon, \delta, c, \xi)$, *described in Figure 4, which takes as input five parameters: (i) query access to a function $f\colon I \to \mathbb{R} \cup \{*\}$; (ii) two parameters $\varepsilon, \delta \in (0, 1)$; (iii) an integer $c \in [k-1]$; and (iv) a parameter $\xi \in (0, 1]$; and outputs either a length-k increasing subsequence or* fail.

Suppose that there exists an interval-tuple pair (I', T) which is $(c, 1/(6k), \varepsilon)$-splittable and $|I'|/|I| \ge \xi$. Then, the algorithms `Find-Good-Split`$_k(f, \varepsilon, \delta, c, \xi)$ *finds a $(12\ldots k)$-pattern of f with probability $1 - \delta$.*

3.4 The Main Algorithm

Consider the description of the main algorithm in Figure 5. The proof uses Lemma 12, Lemma 13, and Lemma 14.

Subroutine $\texttt{Find-Good-Split}_k(f, \varepsilon, \delta, c, \xi)$.

Input: Query access to a function $f: I \to \mathbb{R} \cup \{*\}$, parameters $\varepsilon, \delta \in (0,1)$, and $c \in [k-1]$. We let $c_1 > 1$ be a large enough (absolute) constant.

Output: a sequence $i_1 < \ldots < i_k$ with $f(i_1) < \ldots < f(i_k)$, or fail.

1. Repeat the following procedure $t = c_1 k/(\varepsilon\xi^2) \cdot \log(1/\delta)$ times:
   a. Sample $\boldsymbol{w}, \boldsymbol{z} \sim I$, and consider the functions $f_{\boldsymbol{z},\boldsymbol{w}}: I \cap (-\infty, \boldsymbol{z}) \to \mathbb{R} \cup \{*\}$ and $f'_{\boldsymbol{z},\boldsymbol{w}}: I \cap [\boldsymbol{z}, \infty) \to \mathbb{R} \cup \{*\}$ given by

$$f_{\boldsymbol{z},\boldsymbol{w}}(i) = \begin{cases} f(i) & f(i) < f(\boldsymbol{w}) \\ * & \text{o.w.} \end{cases} \quad \text{and } f'_{\boldsymbol{z},\boldsymbol{w}}(i) = \begin{cases} f(i) & f(i) \geq f(\boldsymbol{w}) \\ * & \text{o.w.} \end{cases}. \quad (2)$$

   b. Run $\texttt{Find-Monotone}_c(f_{\boldsymbol{z},\boldsymbol{w}}, \varepsilon\xi/3, \delta/3)$ and $\texttt{Find-Monotone}_{k-c}(f'_{\boldsymbol{z},\boldsymbol{w}}, \varepsilon\xi/3, \delta/3)$.
2. If both runs of Line 1b are successful for some iteration and some $\boldsymbol{w}$, $\boldsymbol{z}$ and c, then we output the combination of their outputs which forms a length-k increasing subsequence of f; otherwise, output fail.

Figure 4 Description of the `Find-Good-Split` subroutine.

Proof of Theorem 10. The proof is by induction on k. For the base case of $k = 1$, recall that f has at least $\varepsilon|I|$ non-$*$ values. Thus, with probability at least $1 - \delta$, a non-$*$ value is observed after sampling $\boldsymbol{x} \sim I$ at least $(1/\varepsilon) \cdot \log(1/\delta)$ times. It follows that with probability at least $1 - \delta$, Line 2a of our main algorithm, given in Figure 5, samples $\boldsymbol{x} \neq *$ in one of its iterations. We next proceed to the inductive Step: namely, we prove Theorem 10 for $k \geq 2$, under the assumption that it holds for every k' with $1 \leq k' < k$.

Let $p = P(k \log(1/\varepsilon))$ (recall that $P(\cdot)$ is a polynomial of sufficiently large (constant) degree). Apply Theorem 9 to f.

Suppose, first, that (1) of Theorem 9 holds. So, there exists a set $H \subset [n]$ of indices that start an $(\alpha, Ck\alpha)$-growing suffix, with $\alpha|H| \geq (\varepsilon/p)n$, for some $\alpha \in (0,1)$. By Lemma 12, the call for $\texttt{Sample-Suffix}_k(f, \varepsilon/p, \delta)$ in Line 1 outputs a length-k monotone subsequence of f with probability at least $1 - \delta$.

Now suppose that (2) of Theorem 9 holds, and let $(I_1, T_1), \ldots, (I_s, T_s)$ be a $(c, 1/(6k), \alpha)$-splittable collection for some $\alpha \geq \Omega(\varepsilon/k^5)$ and $c \in [k-1]$, satisfying the robust splittable intervals condition and, moreover, that any $J \subset I$ with $J \supset I_h$ for some $h \in [s]$ contains $(\varepsilon/p)|J|$ disjoint $(12 \ldots k)$-patterns. Let `Event` be the event that, for a particular iteration of Lines 2a and 2b, $\boldsymbol{x}$ is the 1-entry of some k-tuple from T_h, for some $h \in [s]$, and $\boldsymbol{y}_t$ is the $(c+1)$-entry of some (possibly other) k-tuple in T_h, where t is such that $|I_h| \leq 2^t < 2|I_h|$.

▷ Claim 15. $\mathbf{Pr}[\texttt{Event}] \geq \varepsilon\alpha/(2p)$.

Proof. For each $h \in [s]$, let A_h and B_h be the collections of 1- and $(c+1)$-entries of patterns in T_h. Then

$$\sum_{h=1}^{s} |A_h| = \sum_{h=1}^{s} |T_h| \geq \alpha \sum_{h=1}^{s} |I_h| \geq \frac{\varepsilon}{p} \cdot |I|.$$

The first inequality follows from the assumption that (I_h, T_h) is $(c, 1/(6k), \alpha)$-splittable, and the second inequality follows from the assumption that the robust splittable condition of Theorem 9 holds.

Subroutine `Find-Monotone`$_k(f, \varepsilon, \delta)$.

Input: Query access to a function $f\colon I \to \mathbb{R} \cup \{*\}$, parameters $\varepsilon, \delta \in (0,1)$. We let $c_1, c_2, c_3 > 0$ be large enough constants, and let $p = P(k \log(1/\varepsilon))$, where $P\colon \mathbb{R} \to \mathbb{R}$ is a polynomial of large enough (constant) degree.

Output: a sequence $i_1 < \ldots < i_k$ with $f(i_1) < \ldots < f(i_k)$, or fail.

1. Run `Sample-Suffix`$_k(f, \varepsilon/p, \delta)$.
2. Repeat the following for $c_1 \log(1/\delta) \cdot p \cdot k^5/\varepsilon^2$ many iterations:
   - **a.** Sample $\boldsymbol{x} \sim I$ uniformly at random. If $f(\boldsymbol{x}) = *$, proceed to the next iteration. Otherwise, if $k = 1$ output $\boldsymbol{x}$ and proceed to Step 3, and if $k \geq 2$ proceed to the next step.
   - **b.** For each $t \in [\log n]$, sample $\boldsymbol{y}_t \sim [\boldsymbol{x} + 2^t/(12k), \boldsymbol{x} + 2^t]$ uniformly at random. If there exists at least one t where $f(\boldsymbol{y}_t) > f(\boldsymbol{x})$, set
   $$\boldsymbol{y} = \max\{\boldsymbol{y}_t : t \in [\log n] \text{ and } f(\boldsymbol{y}_t) > f(\boldsymbol{x})\}, \tag{3}$$
   let $t^* \in [\log n]$ be the index for which $\boldsymbol{y}_{t^*} = \boldsymbol{y}$, and continue to the next line. Otherwise, i.e. if $f(\boldsymbol{y}_t) \not> f(\boldsymbol{x})$ for every t, continue to the next iteration.
   - **c.** If $k = 2$, output $(\boldsymbol{x}, \boldsymbol{y})$ and proceed to Step 3. If $k > 2$, continue to the next line.
   - **d.** Here $k \geq 3$. Set $\ell = 4p/\varepsilon$ and perform the following.
     - **i.** Consider the collection $\mathcal{J}$ of $k-2$ intervals $J_1, \ldots, J_{k-2}$ appearing in order within $[\boldsymbol{x}, \boldsymbol{y}]$, given by setting, for every $i \in [k-2]$,
     $$J_i = \left[\boldsymbol{x} + \frac{2^{t^*}}{12k} \cdot \ell^{-(k-1-i)}, \boldsymbol{x} + \frac{2^{t^*}}{12k} \cdot \ell^{-(k-2-i)}\right), \tag{4}$$
     and run `Find-Within-Interval`$_k(f, \varepsilon/2p, \delta/2, \boldsymbol{x}, \boldsymbol{y}, \mathcal{J})$.
     - **ii.** For each $t' \in [t^* - 3k \log \ell, t^*]$ do the following.
     Consider the interval $J_{t'} = [\boldsymbol{x} - 2^{t'}, \boldsymbol{x} + 2^{t'}]$, and the restricted function $g_{t'}\colon J_{t'} \to \mathbb{R} \cup \{*\}$ given by $g_{t'} = f|_{J_{t'}}$. For every $c_0 \in [k-1]$, run `Find-Good-Split`$_k(g_{t'}, \varepsilon/(c_2 k^5), \delta/2, c_0, 1/4)$.
3. If a length-k monotone subsequence of f is found, output it. Otherwise, output fail.

Figure 5 Description of the `Find-Monotone` subroutine.

As a result, the probability over the draw of $\boldsymbol{x} \sim I$ in Line 2a that $\boldsymbol{x} \in A_h$ is at least ε/p. Fix such an $\boldsymbol{x}$, and consider $t \in [\log n]$ for which $|I_h| \leq 2^t < 2|I_h|$. Notice that $B_h \subset [\boldsymbol{x} + 2^t/(12k), \boldsymbol{x} + 2^t]$ since $2^{t-1} \leq |I_h| < 2^t$, and that the distance between any index of A_h and B_h is at least $|I_h|/(6k) \geq 2^t/(12k)$ since (I_h, T_h) is $(c, 1/(6k), \alpha)$-splittable. Therefore, the probability over the draw of $\boldsymbol{y}_t \sim [\boldsymbol{x} + 2^t/(12k), \boldsymbol{x} + 2^t]$ that $\boldsymbol{y}_t \in B_h$ is at least $|B_h|/2^t \geq |T_h|/(2|I_h|) \geq \alpha/2$. ◀

By the previous claim, since we have $c_1 \cdot \log(1/\delta) \cdot p \cdot k^5/\varepsilon^2$ iterations of Lines 2a and 2b, with probability at least $1 - \delta/2$, `Event` holds in some iteration (using the lower bound $\alpha \geq \Omega(\varepsilon/k^5)$ and the choice of c_1 as a large constant).

Consider the first execution of Line 2a and Line 2b where `Event` holds (assuming such an execution exists). Let $h \in [s]$ and $t \in [\log n]$ be the corresponding parameters, i.e., h and t are set so $\boldsymbol{x}$ is the first index of a k-tuple in T_h, $\boldsymbol{y}_t$ is the $(c+1)$-th index in another k-tuple in T_h, and $|I_h| \le 2^t < 2|I_h|$. We consider this iteration of Line 2, and assume that `Event` holds with these parameters for the rest of the proof. Notice that $\boldsymbol{y}$, as defined in (3), satisfies $\boldsymbol{y} \ge \boldsymbol{y}_t$ (as $f(\boldsymbol{y}) > f(\boldsymbol{x})$) and hence $t^* \ge t$.

Note that if $k = 2$, the pair $(\boldsymbol{x}, \boldsymbol{y})$, which is a (12)-pattern in f, is output in Line 2c, so the proof is complete in this case. From now on, we assume that $k \ge 3$. We break up the analysis into two cases: $t^* \ge t + 3k \log \ell$ and $t^* < t + 3k \log \ell$.

Suppose $t^* \ge t + 3k \log \ell$. We now observe a few facts about the collection $\mathcal{J}$ specified in (4). First, notice that $J_1, \dots, J_{k-2}$ appear in order from left-to-right, and they lie in $[\boldsymbol{x}, \boldsymbol{y}]$ (as $\boldsymbol{y} = \boldsymbol{y}_{t^*} \in [\boldsymbol{x} + 2^{t^*}/(12k), 2^{t^*}]$). Second, in the next claim we show that for every $i \in [k-2]$, the interval J_i contains $(\varepsilon/2p)|J_i|$ disjoint $(12\dots k)$-patterns.

▷ Claim 16. J_i contains $(\varepsilon/2p)|J_i|$ disjoint $(12\dots k)$-patterns.

Proof. Let J_i' be the interval given by $J_i' = I_h \cup \left[\boldsymbol{x}, \boldsymbol{x} + \frac{2^{t^*}}{12k} \cdot \ell^{-(k-2-i)}\right]$. Observe that

$$|J_i' \setminus J_i| \le 2^t + \frac{2^{t^*}}{12k} \cdot \ell^{-(k-1-i)} \le \frac{2^{t^*}}{6k} \cdot \ell^{-(k-1-i)} = \frac{2}{\ell} \cdot \frac{2^{t^*}}{12k} \cdot \ell^{-(k-2-i)} \ge \frac{2}{\ell} \cdot |J_i'| = \frac{\varepsilon}{2p} \cdot |J_i'|,$$

where for the second inequality we used the bound $t^* - t \ge 3k \log \ell \ge \log(12) + \log k + (k-2)\log \ell$, and that $\ell = 4p/\varepsilon$. By Theorem 9, J_i' contains at least $(\varepsilon/p)|J_i'|$ disjoint $(12\dots k)$-patterns in f. Hence, the number of disjoint $(12\dots k)$-patterns in J_i is at least:

$$\frac{\varepsilon}{p} \cdot |J_i'| - |J_i' \setminus J_i| \ge \frac{\varepsilon}{2p} \cdot |J_i'| \ge \frac{\varepsilon}{2p} \cdot |J_i|,$$

as required. ◀

By Lemma 13, Line 2(d)i outputs a $(12\dots k)$-pattern in f with probability at least $1 - \delta/2$. By a union bound, we obtain the desired result.

Suppose, on the other hand, that $t^* \le t + 3k \log \ell$. In this case, as $2^{t-1} \le |I_h| \le 2^{t^*}$ (by choice of t), for one of the values of t' considered in Line 2(d)ii we have $2^{t'-1} \le |I_h| < 2^{t'}$; fix this t'. The interval $J_{t'}$, defined in Line 2(d)ii, hence satisfies $|I_h|/|J_{t'}| \ge 1/4$. As a result, and since $I_h \subset J_{t'}$ (because $t \le t^*$), the function $g \colon J \to \mathbb{R} \cup \{*\}$ contains an interval-tuple pair (I_h, T_h) which is $(c, 1/(6k), \alpha)$-splittable. By Lemma 14, once Line 2(d)ii considers $c_0 = c$, the sub-routine $\texttt{Find-Good-Split}_k(g, \varepsilon/(c_2 k^5), \delta/2, c, 1/4)$ will output a $(12\dots k)$-pattern of $g_{t'}$ (which is also a $(12\dots k)$-pattern of f) with probability at least $1 - \delta/2$. Hence, we obtain the result by a union bound. ◀

References

1 Nir Ailon, Bernard Chazelle, Seshadhri Comandur, and Ding Liu. Estimating the distance to a monotone function. *Random Structures and Algorithms*, 31(3):371–383, 2007.

2 Alexandr Andoni, Negev Shekel Nosatzki, Sandip Sinha, and Clifford Stein. Estimating the longest increasing subsequence in nearly optimal time. *arXiv preprint arXiv:2112.05106*, 2021.

3 Aleksandrs Belovs. Adaptive Lower Bound for Testing Monotonicity on the Line. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, pages 31:1–31:10, 2018.

4 Aleksandrs Belovs and Eric Blais. Quantum algorithm for monotonicity testing on the hypercube. *Theory of Computing*, 11(16):403–412, 2015.

5 Omri Ben-Eliezer. Testing local properties of arrays. In *Proceedings of the 10th Conference on Innovations in Theoretical Computer Science (ITCS)*, pages 11:1–11:20, 2019.

6 Omri Ben-Eliezer and Clément L. Canonne. Improved bounds for testing forbidden order patterns. In *Proceedings of the 29th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2093–2112, 2018.

7 Omri Ben-Eliezer, Clément L. Canonne, Shoham Letzter, and Erik Waingarten. Finding monotone patterns in sublinear time. In *Proceedings of the 60th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 1469–1494. 2019.

8 Benjamin Aram Berendsohn, László Kozma, and Dániel Marx. Finding and Counting Permutations via CSPs. In *14th International Symposium on Parameterized and Exact Computation (IPEC 2019)*, volume 148 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 1:1–1:16. 2019.

9 Hadley Black, Deeparnab Chakrabarty, and C. Seshadhri. A $o(d) \cdot \text{polylog} n$ monotonicity tester for boolean functions over the hypergrid $[n]^d$. In *Proceedings of the 29th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2133–2151, 2018.

10 Eric Blais, Joshua Brody, and Kevin Matulef. Property testing lower bounds via communication complexity. *Computational Complexity*, 21(2):311–358, 2012.

11 Eric Blais, Sofya Raskhodnikova, and Grigory Yaroslavtsev. Lower bounds for testing properties of functions over hypergrid domains. In *Proceedings of the 29th Conference on Computational Complexity (CCC)*, pages 309–320, 2014.

12 Mahdi Boroujeni and Saeed Seddighin. Improved MPC algorithms for edit distance and ulam distance. In *The 31st ACM Symposium on Parallelism in Algorithms and Architectures*, pages 31–40, 2019.

13 Jop Briët, Sourav Chakraborty, David García-Soriano, and Arie Matsliah. Monotonicity testing and shortest-path routing on the cube. *Combinatorica*, 32(1):35–53, 2012.

14 Deeparnab Chakrabarty and C. Seshadhri. Optimal bounds for monotonicity and Lipschitz testing over hypercubes and hypergrids. In *Proceedings of the 45th ACM Symposium on the Theory of Computing (STOC)*, pages 419–428, 2013.

15 Deeparnab Chakrabarty and C. Seshadhri. An optimal lower bound for monotonicity testing over hypergrids. *Theory of Computing*, 10(17):453–464, 2014.

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

17 Deeparnab Chakrabarty and C Seshadhri. Adaptive boolean monotonicity testing in total influence time. In *Proceedings of the 10th Conference on Innovations in Theoretical Computer Science (ITCS)*, pages 20:1–20:7, 2019.

18 Alex Chen, Timothy Chu, and Nathan Pinsker. The dynamic longest increasing subsequence problem. *arXiv preprint arXiv:1309.7724*, 2013.

19 Xi Chen, Anindya De, Rocco A. Servedio, and Li-Yang Tan. Boolean function monotonicity testing requires (almost) $n^{1/2}$ non-adaptive queries. In *Proceedings of the 47th ACM Symposium on the Theory of Computing (STOC)*, pages 519–528, 2015.

20 Xi Chen, Rocco A. Servedio, and Li-Yang Tan. New algorithms and lower bounds for monotonicity testing. In *Proceedings of the 55th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 285–295, 2014.

21 Xi Chen, Erik Waingarten, and Jinyu Xie. Beyond Talagrand functions: new lower bounds for testing monotonicity and unateness. In *Proceedings of the 49th ACM Symposium on the Theory of Computing (STOC)*, pages 523–536, 2017.

22 Robert P. Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 51(1):161–166, 1950.

23 Yevgeniy Dodis, Oded Goldreich, Eric Lehman, Sofya Raskhodnikova, Dana Ron, and Alex Samorodnitsky. Improved testing algorithms for monotonicity. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, pages 97–108, 1999.

24 Funda Ergün and Hossein Jowhari. On the monotonicity of a data stream. *Combinatorica*, 35(6):641–653, 2015.

25 Funda Ergün, Sampath Kannan, S. Ravi Kumar, Ronitt Rubinfeld, and Mahesh Vishwanthan. Spot-checkers. *Journal of Computer and System Sciences*, 60(3):717–751, 2000.

26 Chaim Even-Zohar and Calvin Leng. Counting small permutation patterns. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2288–2302, 2021.

27 Eldar Fischer. On the strength of comparisons in property testing. *Information and Computation*, 189(1):107–116, 2004.

28 Jacob Fox. Stanley–Wilf limits are typically exponential. *arXiv:1310-8378*, 2013.

29 Michael L. Fredman. On computing the length of longest increasing subsequences. *Discrete Mathematics*, 11(1):29–35, 1975.

30 Anna Gál and Parikshit Gopalan. Lower bounds on streaming algorithms for approximating the length of the longest increasing subsequence. *SICOMP*, 39(8):3463–3479, 2010. Short version in FOCS'07.

31 Paweł Gawrychowski and Wojciech Janczewski. Fully dynamic approximation of LIS in polylogarithmic time. In *Proceedings of the 53rd ACM Symposium on the Theory of Computing (STOC)*, pages 654–667, 2021.

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

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

34 Parikshit Gopalan, T. S. Jayram, Robert Krauthgamer, and Ravi Kumar. Estimating the sortedness of a data stream. In *Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 318–327, 2007.

35 Sylvain Guillemot and Dániel Marx. Finding small patterns in permutations in linear time. In *Proceedings of the 25th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 82–101, 2014.

36 Sungjin Im, Benjamin Moseley, and Xiaorui Sun. Efficient massively parallel methods for dynamic programming. In *Proceedings of the 49th ACM Symposium on the Theory of Computing (STOC)*, pages 798–811, 2017.

37 Subhash Khot, Dor Minzer, and Muli Safra. On monotonicity testing and boolean isoperimetric type theorems. In *Proceedings of the 56th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 52–58, 2015.

38 Donald E. Knuth. *The Art of Computer Programming, Volume I: Fundamental Algorithms.* Addison-Wesley, 1968.

39 Tomasz Kociumaka and Saeed Seddighin. Improved dynamic algorithms for longest increasing subsequence. In *53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 640–653, 2021.

40 Adam Marcus and Gábor Tardos. Excluded permutation matrices and the Stanley–Wilf conjecture. *Journal of Combinatorial Theory, Series A*, 107:153–160, 2004.

41 Michael Mitzenmacher and Saeed Seddighin. Dynamic algorithms for LIS and distance to monotonicity. In *Proceedings of the 52nd ACM Symposium on the Theory of Computing (STOC)*, pages 671–684, 2020.

42 Michael Mitzenmacher and Saeed Seddighin. Improved sublinear time algorithm for longest increasing subsequence. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1934–1947, 2021.

43 Timothy Naumovitz and Michael E. Saks. A polylogarithmic space deterministic streaming algorithm for approximating distance to monotonicity. In *Proceedings of the 26th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1252–1262, 2015.

44 Ilan Newman, Yuri Rabinovich, Deepak Rajendraprasad, and Christian Sohler. Testing for forbidden order patterns in an array. In *Proceedings of the 28th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1582–1597, 2017.

45 Ilan Newman, Yuri Rabinovich, Deepak Rajendraprasad, and Christian Sohler. Testing for forbidden order patterns in an array. *Random Structures and Algorithms*, 55:402–426, 2019. Extended abstract in SODA 2017 [44].

46 Ilan Newman and Nithin Varma. New sublinear algorithms and lower bounds for LIS estimation. In *48th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 100:1–100:20, 2021.

47 Ilan Newman and Nitin Varma. Strongly sublinear algorithms for testing pattern freeness. *arXiv preprint arXiv:2106.04856*, 2021. To appear in ICALP 2022.

48 Ramesh Krishnan S. Pallavoor, Sofya Raskhodnikova, and Nithin M. Varma. Parameterized property testing of functions. *ACM Transactions on Computation Theory*, 9(4):17:1–17:19, 2018.

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

50 Prakash Ramanan. Tight $\omega(n \log n)$ lower bound for finding a longest increasing subsequence. *International Journal of Computer Mathematics*, 65(3–4):161–164, 1997.

51 Ronitt Rubinfeld and Madhu Sudan. Robust characterization of polynomials with applications to program testing. *SIAM Journal on Computing*, 25(2):252–271, 1996.

52 Aviad Rubinstein, Saeed Seddighin, Zhao Song, and Xiaorui Sun. Approximation algorithms for LCS and LIS with truly improved running times. In *Proceedings of the 60th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 1121–1145. 2019.

53 Michael Saks and C. Seshadhri. Space efficient streaming algorithms for the distance to monotonicity and asymmetric edit distance. In *Proceedings of the 24th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1698–1709, 2013.

54 Michael E. Saks and C. Seshadhri. Estimating the longest increasing sequence in polylogarithmic time. *SIAM J. Comput.*, 46(2):774–823, 2017. Short version in FOCS 2010.

55 Rodica Simion and Frank W. Schmidt. Restricted permutations. *European Journal of Combinatorics*, 6(4):383 – 406, 1985.

56 Xiaoming Sun and David P. Woodruff. The communication and streaming complexity of computing the longest common and increasing subsequences. In *Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 336–345, 2007.

57 Vincent Vatter. Permutation classes. In *Handbook of Enumerative Combinatorics*, pages 777–858. Chapman and Hall/CRC, 2015.