Strongly Sublinear Algorithms for Testing Pattern Freeness

Ilan Newman
Department of Computer Science, University of Haifa, Israel

Nithin Varma
Chennai Mathematical Institute, India

Abstract

For a permutation $\pi : [k] \to [k]$, a function $f : [n] \to \mathbb{R}$ contains a π-appearance if there exists $1 \leq i_1 < i_2 < \cdots < i_k \leq n$ such that for all $s, t \in [k]$, $f(i_s) < f(i_t)$ if and only if $\pi(s) < \pi(t)$. The function is π-free if it has no π-appearances. In this paper, we investigate the problem of testing whether an input function f is π-free or whether f differs on at least εn values from every π-free function. This is a generalization of the well-studied monotonicity testing and was first studied by Newman, Rabinovich, Rajendraprasad and Sohler [28]. We show that for all constants $k \in \mathbb{N}$, $\varepsilon \in (0, 1)$, and permutation $\pi : [k] \to [k]$, there is a one-sided error ε-testing algorithm for π-freeness of functions $f : [n] \to \mathbb{R}$ that makes $\tilde{O}(n^{o(1)})$ queries. We improve significantly upon the previous best upper bound $O(n^{1-1/(k-1)})$ by Ben-Eliezer and Canonne [7]. Our algorithm is adaptive, while the earlier best upper bound is known to be tight for nonadaptive algorithms.

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

Keywords and phrases Property testing, Pattern freeness, Sublinear algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.98

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2106.04856 [30]

Funding *Ilan Newman*: The author was supported by the Israel Science Foundation, grant number 379/21.

1 Introduction

Given a permutation $\pi : [k] \to [k]$, a function $f : [n] \to \mathbb{R}$ contains a π-appearance if there exists $1 \leq i_1 < i_2 < \cdots < i_k \leq n$ such that for all $s, t \in [k]$ it holds that $f(i_s) < f(i_t)$ if and only if $\pi(s) < \pi(t)$. In other words, the function restricted to the indices $\{i_1, \ldots, i_k\}$ respects the ordering in π. The function is π-free if it has no π-appearance. For instance, the set of all real-valued monotone non-decreasing functions over $[n]$ is $(2, 1)$-free. The notion of π-freeness is well-studied in combinatorics, where the famous Stanley-Wilf conjecture about the bound on the number of π-free permutations $f : [n] \to [n]$ has spawned a lot of work [13, 14, 5, 25, 3], ultimately culminating in a proof by Marcus and Tardos [26]. The problem of designing algorithms to determine whether a given permutation $f : [n] \to [n]$ is π-free is an active area of research [2, 1, 10], with linear time algorithms for constant k [23, 20]. Apart from the theoretical interest, practical motivations to study π-freeness include the study of motifs and patterns in time series analysis [11, 32, 24].

In this paper, we study property testing of π-freeness, first studied by Newman, Rabinovich, Rajendraprasad and Sohler [28]. Specifically, given $\varepsilon \in (0, 1)$, an ε-testing algorithm for π-freeness accepts an input function f that is π-free, and rejects if at least εn values of

49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 98; pp. 98:1–98:20


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

f need to be changed for it to become π-free[1]. This problem is a generalization of the well-studied monotonicity testing on the line ($(2,1)$-freeness), which was one of the first works in combinatorial property testing, and is still being studied actively [17, 18, 12, 15, 6, 16, 31].

Newman, Rabinovich, Rajendraprasad and Sohler [28] showed that for a general permutation π of length k, the problem of π-freeness can be ε-tested using a nonadaptive[2] algorithm of query complexity $O_{k,\varepsilon}(n^{1-1/k})$.[3] Additionally, they showed that, for nonadaptive algorithms, one cannot obtain a significant improvement on this upper bound for $k \geq 4$. In a subsequent work, Ben-Eliezer and Canonne [7] improved this upper bound to $O_{k,\varepsilon}(n^{1-1/(k-1)})$, which they showed to be tight for nonadaptive algorithms. For monotone permutations π of length k, namely, either $(1, 2, \ldots, k)$ or $(k, k-1, \ldots, 1)$, [28] presented an algorithm with query complexity $(\varepsilon^{-1} \log n)^{O(k^2)}$ to ε-test π-freeness. This was improved, in a sequence of works [8, 9], to $O_{k,\varepsilon}(\log n)$, which is optimal for constant ε even for the special case of testing $(2,1)$-freeness [19].

Despite the extensive study and advances in testing freeness of monotone permutations, improving the complexity of testing freeness of arbitrary permutations has remained open all this while. For arbitrary permutations of length at most 3, [28] gave an adaptive algorithm for testing freeness with query complexity $(\varepsilon^{-1} \log n)^{O(1)}$. However, the case of general $k > 3$ has remained elusive. In particular, the techniques of [28] for $k = 3$ do not seem to generalize even for $k = 4$.

As remarked above, optimal *nonadaptive* algorithms are known for any k [7], but, their complexity tends to be linear in the input length as k grows. For the special case of $(2,1)$-freeness, it is well-known that adaptivity does not help at all in improving the complexity of testing [18, 19]. Adaptivity is known to help somewhat for the case of testing freeness of monotone permutations of length k, where, every nonadaptive algorithm has query complexity $\Omega((\log n)^{\log k})$ [8], and the $O_{k,\varepsilon}(\log n)$-query algorithm of Ben-Eliezer, Letzter, and Waingarten [9] is adaptive. Adaptivity significantly helps in testing freeness of arbitrary permutations of length 3 as shown by [28] and [7].

Our results. In this work, we give adaptive ε-testing algorithms for π-freeness of permutations π of arbitrary constant length k with complexity $\tilde{O}_{k,\varepsilon}(n^{o(1)})$. Hence, testing π-freeness has quite efficient sublinear algorithms even for relatively large patterns. Our result shows a strong separation between adaptive and nonadaptive algorithms for testing pattern freeness.

▶ **Theorem 1.1.** *Let $\varepsilon \in (0,1)$, $k \in \mathbb{N}$ and $\pi : [k] \to [k]$ be a permutation. There exists an ε-tester for π-freeness of functions $f : [n] \to \mathbb{R}$ with query complexity $\tilde{O}_{k,\varepsilon}(n^{O(1/\log\log\log n)})$.*

Discussion of our techniques. The algorithm that we design has one-sided error and rejects only if it finds a π-appearance in the input function $f : [n] \to \mathbb{R}$. In the following paragraphs, we present some ideas behind a $\tilde{O}(\sqrt{n})$-query algorithm for detecting a π-appearance in a function f that is ε-far from π-free, for a permutation π of length 4. The case of length-4 permutations is not much different from the general case (where, we additionally recurse on problems of smaller length patterns). The $\tilde{O}(\sqrt{n})$ queries algorithm, however, is much

[1] Algorithms in this area are typically randomized, and the decisions to accept and reject are with high constant probability. See [33, 22] for definitions of property testing.

[2] An algorithm whose queries do not depend on the answers to previous queries is a nonadaptive algorithm. It is adaptive otherwise.

[3] Throughout this work, we are interested in the parameter regime of constant $\varepsilon \in (0,1)$ and k. The notation $O_{k,\varepsilon}(\cdot)$ hides a factor that is an arbitrary function of these parameters.

simpler than the general one, but it outlines many of the ideas involved in the latter one. A more detailed description appears in Section 3. The formal description of the general algorithm is given in Section 4. The correctness proof and query complexity analysis of the general algorithm can be found in the full version [30].

For a parameter $\varepsilon \in (0,1)$, a function f is ε-far from π-free if at least an εn of its values needs to be changed in order to make it π-free. In other words, the Hamming distance of f from the closest π-free real-valued function over $[n]$ is at least εn. A folklore fact is that the Hamming distance and the deletion distance of f to π-freeness are equal, where the deletion distance of f to π-freeness is the cardinality of the smallest set $S \subseteq [n]$ such that f restricted to $[n] \setminus S$ is π-free. By virtue of this equality, a function that is ε-far from π-free has a matching of π-appearances of cardinality at least $\varepsilon n/4$. This observation facilitates our algorithm and all previous algorithms on testing π-freeness, including monotonicity testers.

The basic ingredient in our algorithms is the use of a natural representation of $f : [n] \to \mathbb{R}$ by a Boolean function over a grid $[n] \times R(f)$, where $R(f)$ denotes the range of f. Specifically, we visualize the function as a grid of n points in $\mathbb{R}^2$, such that for each $i \in [n]$, the point $(i, f(i))$ is a marked point of the grid. We denote this grid with marked points as G_n. This view has been useful in the design of approximation algorithms for the related and fundamental problem of estimating the length of Longest Increasing Subsequence (LIS) in a real-valued array [35, 34, 27, 29]. Adopting this view, for any permutation π, a π-appearance at places $(i_1, \ldots, i_k)$ in f corresponds naturally to a k-tuple of points $\{i_j, f(i_j)\},\ j = 1 \ldots k$ in G_n, for which their relative order (in G_n) forms a π-appearance. The converse is also true: every π-appearance in the Boolean grid G_n corresponds to a π-appearance in f.

We note that the grid G_n is neither known to nor directly accessible by the algorithm, and in particular, $R(f)$ is not assumed to be known. A main first step in our algorithm is to approximate the grid G_n by a coarser $m \times m$ grid of boxes, $G_{m,m}$, for $m << n$, a parameter that will determine the query complexity. The grid $G_{m,m}$ is defined as follows. Suppose that we have a partition of $R(f)$ into m disjoint contiguous intervals of increasing values, referred here as "layers", $I_1, \ldots, I_m$, and let $S_1, \ldots, S_m$ be a partition of $[n]$ into m contiguous intervals of equal length, referred to as "stripes". These two partitions decompose G_n and the f-points in it into m^2 boxes which form the grid of cells $G_{m,m}$. The (i,j)-th cell of this grid is the Cartesian product $S_i \times I_j$, and is denoted $\mathsf{box}(S_i, I_j)$. We view the non-empty cells in $G_{m,m}$ as a coarse approximation of G_n (and of the input function, equivalently). The grid $G_{m,m}$ has a natural order on its boxes (viewed as points in $[m] \times [m]$).

While $G_{m,m}$ is also not directly accessible to the algorithm, it can be well-approximated very efficiently. Fixing m, we use sampling by $\tilde{O}(m)$ queries to identify and *mark* the boxes in $G_{m,m}$ that contain a non-negligible density of points of G_n. This provides a good enough approximation of the grid $G_{m,m}$. For the rest of this top-level explanation, assume that we have fixed $m << n$, and we know $G_{m,m}$; that is, we know the the number of points of G_n belonging to each box in $G_{m,m}$, but not necessarily the points themselves.

If we find k nonempty boxes in $G_{m,m}$ that form a π-appearance when viewed as points in the $[m] \times [m]$ grid, then G_n (and hence f) contains a π-appearance for any set of k points that is formed by selecting one point from each of the corresponding boxes. See Figure 1(A) for such a situation, for $\pi = (3,2,1,4)$. We first detect such π-appearances by our knowledge of $G_{m,m}$. However, the converse is not true: it could be that G_n contains many π-appearances, where the corresponding points, called "legs", are in boxes that share layers or stripes, and hence do not form π-appearances in $G_{m,m}$. See e.g., Figure 1(B) for such an appearance for $\pi = (3,2,1,4)$. Thus, assuming that the function is far from being π-free, and no π-appearances are detected in $G_{m,m}$, there must be many π-appearances in

which some legs share a layer or a stripe in $G_{m,m}$. In this case, the seminal result of Marcus and Tardos [26], implies that only $O(m)$ of the boxes in $G_{m,m}$ are non-empty. An averaging argument implies that if f is ε-far from being π-free, then after deleting layers or stripes in $G_{m,m}$ with $\omega(1)$ dense boxes, we are still left with a partial function (on the undeleted points) that is ε'-far from being π-free, for a large enough ε'.

Now, to be specific, consider $\pi = (3, 2, 1, 4)$ although all the following ideas work for any specific 4-length permutation. Any π-appearance has its four legs spread over at most 4 marked boxes. This implies that there are only constantly many non-isomorphic ways of arranging the marked boxes containing any particular π-appearance (in terms of the order relation among the marked boxes, and the way the legs of the π-appearance are included in them). These constantly many ways are called "configurations" in the sequel. Thus any π-appearance is consistent with a certain configuration. Additionally, in case multiple points in a π-appearance share some marked boxes, this appearance induces the appearances of permutations of length smaller than 4 in each box (which are sub-permutations ν of π). If a constant fraction of the π-appearances are spread across multiple marked boxes, there will be many such ν-appearances in the marked boxes in the coarse grid. Hence, one phase of our algorithm will run tests for ν-appearances for smaller patterns ν (which can be done in $\operatorname{polylog} n$ queries using known testers for patterns of length at most 3) on each marked box, and combine these ν-appearances to detect a π-appearance, if any. This phase, while seemingly simple will require extra care, as combining sub-patterns appearances into a global π-appearance is not always possible. This is a major issue in the general case for $k > 4$.

The simpler case is when there is a constant fraction of π-appearances such that all 4 points of each such appearance belong to a single marked box. This can be solved by randomly sampling a few marked boxes and querying all the points in them to see if there are any π-appearances. A special treatment has to be made in the case a constant fraction of the π-appearances have their legs belonging to the same layer or the same stripe. But this will be an easy extension of the "one-box" case.

To obtain the desired query complexity, consider first setting $m = \tilde{O}(\sqrt{n})$. Getting a good enough estimate of $G_{m,m}$ as described above take $\tilde{O}(m) = \tilde{O}(\sqrt{n})$ queries. Then, testing each box for ν-freeness, for smaller permutations ν takes $\operatorname{polylog} n$ per test, but since this is done for all marked boxes, this step also takes $\tilde{O}(m) = \tilde{O}(\sqrt{n})$. Finally, in the last simpler case, we may just query all indices in a sampled box that contains at most $n/m = \sqrt{n}$ indices, by our setting of m. This results in a $\tilde{O}(\sqrt{n})$-query tester for π-freeness.

To obtain a better complexity, we reduce the value of m, and, in the last step, we randomly sample a few marked boxes and run the algorithm recursively. This is so, since, in the last step, we are in the case that for a constant fraction of the π-appearances, all four legs of each π-appearance belong to a single marked box (or a constant number of marked boxes sharing a layer or stripe). The depth of recursion depends monotonically on n/m and the larger it is the smaller is the query complexity. The bound we describe in this article is $n^{O(1/\log\log n \log n)}$ which is due to the exponential deterioration of the distance parameter ε in each recursive call. Our algorithm for permutations of length $k > 4$ uses, in addition to the self-recursion, a recursion on k too.

Finally, even though it was not explicitly mentioned, we call ν-freeness or π-freeness algorithms on marked boxes (or a collection of constantly many marked boxes sharing a layer or stripe) and not the entire grid. Since we do not know which points belong to the marked boxes, but only know that their density is significant, we can access points in them only via sampling and treating points that fall outside the desired box as being *erased*. This necessitates the use of erasure-resilient testers [16]. Such testers are known for all permutation patterns of length at most 3 [16, 29, 28]. In addition, the basic tester we design is also erasure-resilient, which makes it possible for it to be called recursively.

Some additional complications we had to overcome. In the recursive algorithm for k-length permutation freeness, $k \geq 4$, we need to find ν-appearances that are restricted to appear in specific configurations, for smaller length permutations ν. To exemplify this notion, consider testing $\nu = (1, 3, 2)$-freeness. In the usual (unrestricted) case, $f : [n] \mapsto \mathbb{R}$ has a ν-appearance if the values at any three indices have a ν-consistent order. In a restricted case, we may ask ourselves whether f is free of ν-appearances where the indices corresponding to the $1, 3$-legs of a ν-appearance are at most $n/2$ (that is in the first half of $[n]$), while the index corresponding to the 2-leg is larger than $n/2$. This latter property seems at least as hard to test as the unrestricted one. In particular, for the ν-appearance as described above, it could be that while f is far from being ν-free in the usual sense, it is still free of having restricted ν-appearances. In our algorithm, we need to test (at lower recursion levels) freeness from such restricted appearances. The extra restriction is discussed in Section 3 and Section 4.

Open questions. The major open question is to determine the exact (asymptotic) complexity of testing π-freeness of arbitrary permutations $\pi : [k] \to [k]$, $k \geq 3$. While the gaps for $k = 3$ are relatively small (within polylog n range), the gaps are yet much larger for $k \geq 4$. We do not have any reason to think that the upper bound obtained in this draft is tight. We did not try to optimize the exponent of n in the $\tilde{O}(n^{o(1)})$ expression, but the current methods do not seem to bring down the query complexity to polylog n. We conjecture, however, that the query complexity is polylog n for all constant k. Another open question is whether the complexity of a two-sided error testing might be lower than of one-sided error testing.

Finally, Newman and Varma [29] used lower bounds on testing pattern freeness of monotone patterns of length $k \geq 3$ (for nonadaptive algorithms), to obtain lower bounds on the query complexity of nonadaptive algorithms for LIS estimation. Proving any lower bound better than $\Omega(\log n)$ for adaptively testing freeness, for arbitrary permutations of length k for $k \geq 3$, may translate in a similar way to lower bounds on adaptive algorithms for LIS estimation.

Other definitions of π-freeness. In the definition of π-freeness, we required strict inequalities on function values to have an occurrence of the pattern. A natural variant is to allow weak inequalities, that is – for a set indices $1 \leq i_1 < i_2 \cdots < i_k \leq n$ a *weak*-π appearance is when for all $s, t \in [k]$ it holds that $f(i_s) \leq f(i_t)$ if and only if $\pi(s) < \pi(t)$. Such a relaxed requirement would mean that having a collection of k or more equal values is already a π-appearance for any pattern π. For monotone patterns of length k, the deletion distance equals to the Hamming distance, for any k, for this relaxed definition as well. We do not know if this is true for larger k for non-monotone patterns in general, although we suspect that the Hamming distance is never larger than the deletion distance by more than a constant factor. Proving this will be enough to make our results true for testing freeness of any constant size forbidden permutation, even with the relaxed definition. We show that the Hamming distance is equal to the deletion distance for patterns of length at most 4. Hence, Theorem 1.1 also holds for weak-π-freeness for $k \leq 4$.

Another variant that may seem related is when the forbidden order pattern is not necessarily a permutation (that is, arbitrary function from $[k]$ to $[k]$ which is not one-to-one). For example, for the 4-pattern $\alpha = (1, 2, 3, 1)$, an α-appearance in f at indices $i_1 < i_2 < i_3 < i_4$ is when $f(i_1) < f(i_2) < f(i_3)$ and $f(i_4) = f(i_1)$, as dictated by the order in α. For testing freeness of such patterns, $\Omega(\sqrt{n})$ adaptive lower bounds exist (due to a simple probabilistic argument) even for the very simple case of $(1, 1)$-freeness, which corresponds to the property of being a one-to-one function.

An interesting point to mention, in this context, is that for testing freeness of forbidden permutations, a major tool that we use is the Marcus-Tardos bound. Namely, that the number of 1's in an $m \times m$ Boolean matrix that does not contain a specific permutation matrix of order k is $O(m)$. For non-permutation patterns, similar bounds are not true in general anymore, but do hold in many cases (or hold in a weak sense, e.g., only slightly more than linear). In such cases, the Marcus-Tardos bound could have allowed relatively efficient testing. However, the lower bounds hinted above for the $(1,1)$-pattern makes the testing problem completely different from that of testing forbidden permutation patterns.

Another area where we have significant gaps in our knowledge is about testing for pattern freeness for functions of bounded or restricted range (for the special case of $(2,1)$-freeness, such a study was initiated by Pallavoor, Raskhodnikova and Varma [31] and followed upon by others [6, 29]). We do know that in the very extreme case, that is, for functions from the line $[n]$ to a constant-sized range, pattern freeness is testable in constant time even for much more general class of forbidden patterns [4].

Lastly, if we restrict our attention to functions $f : [n] \to [n]$ that are themselves permutations, Fox and Wei [21] argued that for some special types of distance measures such as the rectangular-distance and Kendall's tau distance, testing π-freeness can be done in constant query complexity. Testing π-freeness w.r.t. the Hamming or deletion distances is very different, and still remains open for this setting.

Organization. Section 2 contains the notation, important definitions, and a discussion of some key concepts related to testing π-freeness. Section 3 contains a high level overview of an $\tilde{O}(\sqrt{n})$-query algorithm for patterns of length 4. The formal description of our π-freeness tester for permutations π of length $k \geq 4$ and the proof ideas for a special case appear in Section 4. All the missing proofs can be found in the full version [30].

2 Preliminaries and discussion

For a function $f : [n] \to \mathbb{R}$, we denote by $R(f)$ the image of f. We often refer to the elements of the domain $[n]$ as *indices*, and the elements of $R(f)$ as *values*. For $S \subseteq [n]$, $f|_S$ denotes the restriction of f to S. Throughout, n will denote the domain size of the function f.

We often refer to events in a probability space. For ease of representation, we will say that an event E occurs with high probability, denoted "w.h.p.", if $\Pr(E) > 1 - n^{-\log n}$, to avoid specifying accurate constants.

Let $\mathcal{S}_k$ denote the set of all permutations of length k. We view $\pi = (a_1, \ldots, a_k) \in \mathcal{S}_k$ as a function (and not as a cyclus), that is, where $\pi(i) = a_i,\ i \in [k]$. We refer to a_i as the ith value in π, and as the a_i-leg of π. Thus e.g., for $\pi = (4,1,2,3)$, the first value is 4, and the third is 2, while the 4-leg of π is at the first place and its 1-leg is at the second place. We often refer to $\pi \in \mathcal{S}_k$ as a k-pattern.

2.1 Deletion distance vs. Hamming distance

Let $f : [n] \to \mathbb{R}$. The deletion distance of f from being π-free is $\mathrm{Ddist}_\pi(f) = \min\{|S| :\ S \subseteq [n],\ f|_{[n]\setminus S}$ is π-free$\}$. Namely, it is the cardinality of the smallest set $S \subseteq [n]$ that intersects each π-appearance in f. The Hamming distance of f from being π-free, $\mathrm{Hdist}_\pi(f)$ is the minimum of $\mathrm{dist}(f, f') = |\{i :\ i \in [n],\ f(i) \neq f(i')\}|$ over all functions $f' : [n] \to \mathbb{R}$ that are π-free. For $0 \leq \varepsilon < 1$ we say that f is ε-far from π-freeness in deletion distance, or Hamming distance, if $dist_\pi(f) \geq \varepsilon n$, and otherwise we say that f is ε-close to π-freeness, where $dist_\pi(f)$ is the corresponding distance.

▷ Claim 2.1. $\text{Ddist}_\pi(f) = \text{Hdist}_\pi(f)$

Claim 2.1 is extremely important for testing π-freeness, and is what gives rise to *all* testers of monotonicity, as well as π-freeness that are known. This is due to the fact that the tests are really designed for the deletion distance, rather then the Hamming distance. The folklore observation made in Claim 2.2 facilitates such tests, and Claim 2.1 makes the tests work also for the Hamming distance. Due to Claim 2.1, we say that a function f is ε-far from π-free without specifying the distance measure.

Let $\pi \in \mathcal{S}_k$ and $f : [n] \to \mathbb{R}$. A matching of π-appearances in f is a collection of π-appearances that are pairwise disjoint as sets of indices in $[n]$. The following claim is folklore and immediate from the fact that the size of a minimum vertex cover of a k-uniform hypergraph is at most k times the cardinality of a maximal matching.

▷ Claim 2.2. Let $\pi \in \mathcal{S}_k$. If $f : [n] \to \mathbb{R}$ is ε-far from being π-free, then there exists a matching of π-tuples of size at least $\varepsilon n/k$.

All our algorithms have one-sided error, i.e., they always accept functions that are π-free. For functions that are far from being π-free, using Claim 2.2, our algorithms aim to detect some π-appearance, providing a witness for the function to not be π-free. Hence, in the description below, and throughout the analysis of the algorithms, the input function is assumed to be ε-far from π-free.

2.2 Viewing a function as a grid of points

Let $f : [n] \to \mathbb{R}$. We view f as points in an $n \times |R(f)|$ grid G_n. The horizontal axis of G_n is labeled with the indices in $[n]$. The vertical axis of G_n represents the image $R(f)$ and is labeled with the distinct values in $R(f)$ in increasing order, $r_1 < r_2 < \ldots < r_{n'}$, where $|R(f)| = n' \leq n$. We refer to an index-value pair $(i, f(i)), i \in [n]$ in the grid as a *point*. The grid has n points, to which our algorithms do not have direct access. In particular, we do not assume that $R(f)$ is known. The function is one-to-one if $|R(f)| = n$.

Note that if M is a matching of π-appearances in f, then M defines a corresponding matching of π-appearances in G_n. We will always consider this alternative view, where the matching M is a set of disjoint π-appearances in the grid G_n.

2.2.1 Coarse grid of boxes

For a pair of subsets (S, I), where $S \subseteq [n]$ and $I \subseteq R(f)$, we denote by $\mathsf{box}(S, I)$, the subgrid $S \times I$ of G_n with the set of points in G_n (corresponding to f) in this subgrid. In most cases, S and I will be intervals in $[n]$ and $R(f)$, respectively, and hence the name *box*. The *length* of $\mathsf{box}(S, I)$ is defined to be $|S|$. A box is *nonempty* if it contains at least one point and is *empty* otherwise.

Consider an arbitrary collection of pairwise disjoint contiguous value intervals $\mathcal{L} = \{I_1, \ldots I_m\}$, such that $I \subseteq \cup_{i\in[m]} I_i$. The set $\mathcal{L}$ naturally defines a partition of the points in $\mathsf{box}(S, I)$ into m horizontal *layers*, $L_i = \{(j, f(j)) : j \in S, f(j) \in I_i\},\ i \in [m]$. A layer is *multi-valued* if it has two points with different values. It is said to be *single-valued* otherwise.

Assume that, in addition to a set of layers $\mathcal{L}$, we have a partition of S into disjoint intervals $S = \cup_1^m S_i$ where $S_i = [a_i, b_i]$, and $b_i < a_{i+1},\ i = 1, \ldots m-1$, then $\mathcal{S} = (S_1, \ldots S_m)$ partitions $\mathsf{box}(S, I)$ and the points in it, into m vertical *stripes* $\{\text{St}(S)\}_{S\in\mathcal{S}}$ where $\text{St}(S) = \mathsf{box}(S, I)$ contains the points $\{(i, f(i)) : i \in S\}$. The layering $\mathcal{L}$ together with the stripes $\mathcal{S}$ partition $\mathsf{box}(S, I)$ into a coarse grid $G_{m,m}$ of boxes $\{\mathsf{box}(S_i, I_j)\}_{i,j\in[m]}$ that is isomorphic to the

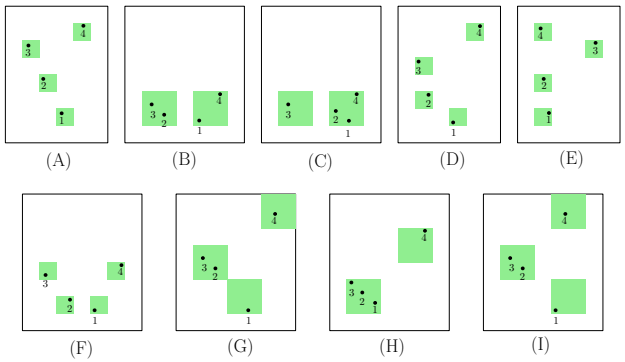

Figure 1 Each rectangle represents a different grid G_n, where the green shaded boxes correspond to some nonempty boxes in those grids. Each figure represents a different configuration type with respect to the appearance of some 4-length pattern. The dots and the numbers indicate possible splittings of the 4 legs of π. *Figure (E) represents the pattern* $(4, 2, 1, 3)$ *and all others represent the pattern* $(3, 2, 1, 4)$.

grid $[m] \times [m]$. Note that $\mathsf{box}(S, I)$ could even be the entire grid G_n. Given such a grid $G_{m,m}$, the layer of $\mathsf{box}(S_i, I_j)$, denoted $L(\mathsf{box}(S_i, I_j))$, is $\mathsf{box}(S, I_j)$ and its stripe, denoted $\mathrm{St}(\mathsf{box}(S_i, I_j))$, is $\mathsf{box}(S_i, I)$.

We say that *layer L is below layer L'*, and write $L < L'$, if the largest value of a point in L is less than the smallest value of a point in L'. For stripes $\mathrm{St}(S), \mathrm{St}(S')$, we write $\mathrm{St}(S) < \mathrm{St}(S')$ if the largest index in S is smaller than the smallest index in S'. For the grid $G_{m,m}$ and two boxes B_1, B_2 in it, $B_1 < B_2$ if $L(B_1) < L(B_2)$ and $\mathrm{St}(B_1) < \mathrm{St}(B_2)$.

2.2.2 Patterns among and within nonempty boxes

Consider a coarse grid of boxes, $G_{m,m}$, defined as above on the grid of points G_n. There is a natural homomorphism from the points in G_n to the nonempty boxes in $G_{m,m}$ where those points fall. For f and a grid of boxes $G_{m,m}$ as above, we refer to this homomorphism implicitly. This homomorphism defines when $G_{m,m}$ contains a π-appearance in a natural way. For example, consider the permutation $\pi = (3, 2, 1, 4) \in \mathcal{S}_4$. We say that $G_{m,m}$ *contains* π if there are nonempty boxes B_1, B_2, B_3, B_4 such that $\mathrm{St}(B_1) < \mathrm{St}(B_2) < \mathrm{St}(B_3) < \mathrm{St}(B_4)$ and $L(B_3) < L(B_2) < L(B_1) < L(B_4)$ (see Figure 1(A)).

▶ **Observation 2.3.** *Let $\mathcal{L}, \mathcal{S}$ be a partition of G_n into layers and stripes as above, with $|\mathcal{L}| = m$, $|\mathcal{S}| = m$ then if $G_{m,m}$ contains π then G_n (and equivalently f) has a π-appearance.*

The converse of Observation 2.3 is not true; G_n may contain a π-appearance while $G_{m,m}$ does not. This happens when some of the boxes that contain the π-appearance share a layer or a stripe. Two boxes are *directly-connected* if they share a layer or a stripe. The transitive closure of the relation *directly-connected* is called *connected*. An arrangement of boxes where every two boxes are connected is called a *connected component*, or simply, a component. The size of a connected component is the number of boxes in it.

For $\pi \in \mathcal{S}_k$, a π-appearance in G_n implies that the k points corresponding to such a π-appearance are in $i \leq k$ distinct components in $G_{m,m}$, where the jth component C_j may contain b_j boxes each containing at least one point of the corresponding π-appearance. We refer to the π-values in the corresponding boxes of the components as *legs*. For example, for $\pi = (3, 2, 1, 4)$, the π-appearance shown in Figure 1(B) is contained in two boxes that share the same layer, and hence form one component. The left box contains the $3, 2$ legs of the π-appearance and other contains the $1, 4$ legs. A different 1-component 2-boxed appearance in the same two boxes has 3 appearing in B_1 and all the other legs in B_2 as in Figure 1(C).

Examples for $\pi = (3, 2, 1, 4)$-appearances with two components C_1, C_2 are illustrated in Figure 1(F) and Figure 1(H). In the first, C_1, C_2 contain 2 boxes each, where C_1 contains the $(3, 4)$ legs of the appearance, each in one box, and C_2 contains the $(1, 2)$ legs. In the second, each component is 1-boxed, where the first contains the $(3, 2, 1)$-legs and the other contains the 4-leg of the appearance. Figure 1(A) contains a $(3, 2, 1, 4)$-appearance in 4 components. Some other possible appearances with 1 component and 3 components are illustrated in Figure 1(B), Figure 1(C), Figure 1(D) and Figure 1(G).

To sum up, each π-appearance in G_n defines an arrangement of nonempty boxes in $G_{m,m}$ that contain the legs of that appearance. This arrangement is defined by the relative order of the layers and stripes among the boxes, and has at most k components. Such a box-arrangement that can contain the legs of a π-appearance is called a *configuration*. Note that there may be many different π-appearances in distinct boxes, all having the same configuration $\mathcal{C}$. Namely, in which, the arrangements of the boxes in terms of the relative order of layers and stripes are identical. So, a configuration is just a set of boxes (or points) in the $k \times k$ grid. An actual set of boxes in $G_{m,m}$ forming a specific type of configuration is referred to as a *copy* of that configuration.

Let $c(k)$ be the number of all possible configurations that are consistent with a π-appearance, for $\pi \in \mathcal{S}_k$. For any fixed π, the number $c(k)$ of distinct types of configurations is constant as shown in the following observation.

▶ **Observation 2.4.** $c(k) \leq 2^{O(k \log k)}$

A configuration $\mathcal{C}$ does not fully specify the way in which a π-appearance can be present. It is necessary to also specify the way the k legs of the π-appearance are partitioned among the boxes in a copy of $\mathcal{C}$. Let $\mathcal{B}$ denote a set of boxes forming the configuration $\mathcal{C}$. Let $\phi : [k] \to \mathcal{B}$ denote the mapping of the legs of the π-appearance to boxes in $\mathcal{B}$, where $\phi(j), j \in [k]$ denotes the box in $\mathcal{B}$ containing the j-th leg of the π-appearance. We say that the copy of $\mathcal{C}$ formed by the boxes in $\mathcal{B}$ contains a ϕ-legged π-appearance.

A configuration $\mathcal{C}$ in which the boxes form $p \geq 2$ components, and that is consistent with a π-appearance, defines $\nu_1, \ldots, \nu_p$ -appearances, respectively, in the p components of $\mathcal{C}$, where ν_j for $j \in [p]$ is the subpermutation of π that is defined by the restriction of π to the j-th component. In addition, $\mathcal{C}$ defines the corresponding mappings $\phi_j, j = 1, \ldots p$, of the corresponding legs of each ν_j to the corresponding boxes in the jth component. For example, consider $\pi = (3, 2, 1, 4)$ and the box arrangement shown in Figure 1(F). That arrangement has two connected components: one that contains B_1, B_4 and the other that contains B_2, B_3, where we number the boxes from left to right (by increasing stripe order). Further, the (only) consistent partition of the legs of π into these boxes is $\pi(i) \in B_i,\ i \in [4]$. In particular, it means that the component formed by B_1, B_4 contains the $3, 4$ legs of π and the component formed by B_2, B_3 contains the $2, 1$ legs of π. Thus, in terms of the discussion above, the component formed by B_1, B_4 has a $\nu_1 = (1, 2)$-appearance (corresponding to the $3, 4$ legs of π), with leg mapping ϕ_1 mapping the 1-leg into B_1 and the 2-leg into B_4. Similarly, the component formed by B_2, B_3 has a $\nu_2 = (2, 1)$-appearance (corresponding to the $2, 1$ legs of π) with corresponding leg mapping ϕ_2 that maps the 2-leg into B_2 and the 1-leg into B_3. Note that the converse is also true: every ν_1 appearance in the component $B_1 \cup B_4$, with a leg-mapping ϕ_1 (that is, in which the $1, 2$ legs are in B_1, B_4 respectively), in addition to a ν_2 appearance in $B_2 \cup B_3$ with the leg-mapping ϕ_2, results in a π-appearance in $G_{m,m}$.

This latter comment leads to the crucial observation that if π defines the corresponding $\nu_1, \ldots, \nu_p$ appearances in the p components of the configuration $\mathcal{C}$, then, *any* $\nu_1, \ldots, \nu_p$-appearances in the p components of any copy of $\mathcal{C}$ with consistent leg-mappings is a π-appearance in $\mathcal{C}$. This is formally stated below.

▶ **Definition 2.5.** *Let $\nu \in \mathcal{S}_r$. Let $B_1, \ldots, B_p$ be a set of boxes forming* one *component C and $\phi : [r] \mapsto \{B_1, \ldots, B_p\}$ be an arbitrary mapping of the legs of a ν-appearance to boxes. We say that C has a ϕ-legged ν-appearance if there is a ν-appearance in $\cup_{j=1}^{p} B_j$ in which for each $i \in [r]$, the i-th leg of ν appears in the box $B_{\phi(i)}$.*

▶ **Observation 2.6.** *Let $\pi \in \mathcal{S}_k$ and assume that there exists a π-appearance in G_n that in the grid of boxes $G_{m,m}$ forms a configuration $\mathcal{C}$ that contains t components $C_1, \ldots, C_t$, with C_j having r_j boxes, $j = 1, \ldots, t$ respectively. Let the restriction of this π-appearance to $C_1, \ldots, C_t$ define the permutation patterns $\nu_1, \ldots \nu_t$ in $C_1, \ldots, C_t$, with leg mappings $\phi_1, \ldots \phi_t$, respectively.*

Then, any collection of configuration copies of C_j, $j = 1, \ldots t$, for which $\cup_1^t C_j$ is a copy of $\mathcal{C}$, and ϕ_j-legged ν_j-appearances in C_i, $i = 1, \ldots, t$, defines a π-appearance in $\cup_1^t C_j$. ⌟

2.3 Erasure-resilient testing

Erasure-resilient (ER) testing, introduced by Dixit, Raskhodnikova, Thakurta and Varma [16], is a generalization of property testing. In this model, algorithms get oracle access to functions for which the values of at most α fraction of the points in the domain are erased by an adversary, for $\alpha \in [0, 1)$. As part of our algorithm for testing π-freeness for $\pi \in \mathcal{S}_k$ for $k \geq 4$, we call testers for smaller subpatterns on sub-regions of the grid G_n which may be defined by, say, $\mathsf{box}(S, I)$ for some $S \subseteq [n], I \subseteq R(f)$. In this case, the only access to points in $\mathsf{box}(S, I)$ is by sampling indices from S and checking whether their values fall in I. If the values do not fall in I, we can treat them as erasures. Given the assurance that the number of points falling in $\mathsf{box}(S, I)$ is a constant fraction of $|S|$, we can simply run ER testers on $f|_S$ to test for these smaller subpatterns.

3 High level description of the basic algorithm for $\pi \in \mathcal{S}_4$

Many of the high level ideas in the design of our π-freeness tester of complexity $\tilde{O}(n^{o(1)})$ are described in this section. For simplicity, we describe first the ideas behind a $\tilde{O}(\sqrt{n})$-query tester for π-freeness of $\pi \in \mathcal{S}_4$. Towards the end of this section, we briefly describe how to generalize these ideas to obtain the query complexity of $\tilde{O}(n^{o(1)})$ and for longer constant-length permutations. For simplicity, we assume in what follows that the input function $f : [n] \to \mathbb{R}$ is one-to-one. The algorithm for functions that are not one-to-one differs in a few places and these are explained in Section 4.1.

For the purposes of this high level description, we fix the forbidden permutation $\pi = (3, 2, 1, 4)$. The same algorithm works for any $\pi \in \mathcal{S}_4$. We view f as an (implicitly given) $n \times |R(f)|$ grid G_n consisting of points $(i, f(i))$ for $i \in [n]$, where, in particular, $R(f)$ is neither known nor bounded. Our first goal is to approximate G_n by a coarse grid of boxes $G_{m,m}$, as described above, for $m = \sqrt{n}$. This is done by querying f on $\tilde{\Theta}(m)$ random indices, after which we obtain a partition $\mathcal{L}$ of $R(f)$ into $m' = \Theta(m)$ horizontal layers (value intervals). Then we partition the index set $[n]$ into m' contiguous intervals $\{S_i\}_{i=1}^{m'}$ of equal length. This results in a grid $G_{m',m'}$ in which we estimate the density of each box as the number of sampled points falling in that box, normalized by n/m'. A box $\mathsf{box}(S_i, I_j), i, j \in [m']$ will be tagged as *dense* if it contains $\Omega(1)$ fraction of sampled points. All of the above takes $\tilde{O}(m) = \tilde{O}(\sqrt{n})$ queries, for the above choice of m. It satisfies the following properties with high probability:

- Each layer, that is $\mathsf{box}([n], I_j)$, $j \in [m']$, has approximately the same number of points.
- It is either the case that the dense boxes contain all but an insignificant fraction of the points in G_n, or the total number of marked boxes is larger than $m' \log n$.

Next, we use the following lemma of Marcus and Tardos.

▶ **Lemma 3.1** ([26]). *For any $\pi \in \mathcal{S}_k$, $k \in \mathbb{N}$, there is a constant $\kappa(k) \in \mathbb{N}$ such that for any $r \in \mathbb{N}$, if a grid $G_{r,r}$ contains at least $\kappa(k) \cdot r$ marked points, then it contains a π-appearance among the marked points.*

Let $\kappa = \kappa(4)$. Using Lemma 3.1, we may assume that there are at most $\kappa \cdot m'$ non-empty boxes in $G_{m',m'}$, as otherwise, we already would have found a π-appearance in $G_{m',m'}$, which by Observation 2.3, implies a π-appearance in G_n and in f as well. Hence, as a result of the gridding, if we do not see a π-appearance among the sampled points, the second item above implies that there are $\Theta(m')$ dense boxes in $G_{m',m'}$ and that these boxes cover a large fraction of the points of G_n.

An averaging argument implies that, for an appropriate constant $d = d(\varepsilon)$, only a small constant fraction of layers (or stripes) contain more than d nonempty boxes. Therefore, since the grid G_n is ε-far from being π-free, the restriction of G_n to the layers and stripes that contain at most d boxes each, is also ε'-far from π-free for a large enough constant $\varepsilon' < \varepsilon$. This implies that G_n restricted to the points in dense boxes that belong to layers and stripes containing at most d dense boxes each, has a matching M of π-appearances of size at least $\varepsilon' n/4$. We assume in what follows that this is indeed the situation.

An important note at this point, is that every dense box B is contained in $O(d^3)$ (that is, constantly many) 1-component configurations with at most 4 dense boxes. This implies that there are $O(m)$ such copies of 1-component configurations in $G_{m',m'}$.

Recall that every π-appearance in M defines a configuration of at most 4 components in $G_{m',m'}$. Hence, the matching M of size $|M| = \Omega(n)$ can be partitioned into 4 sub-matchings $M = M_1 \cup M_2 \cup M_3 \cup M_4$, where M_i, $i = 1, \ldots, 4$ consists of the π-appearances participating in configurations having exactly i components. Since $|M| = \Omega(n)$ it follows that at least one of M_i, $i = 1, 2, 3, 4$ is of linear size. Now, any π-appearance in M_4 is an appearance in 4 distinct dense boxes in $G_{m',m'}$, where no two share a layer or a stripe. In that case, such an appearance can be directly detected from the tagged $G_{m',m'}$ with no further queries.

The description of the rest of the algorithm can be viewed as a treatment of several independent cases regarding which of the constantly many configuration types contributes the larger mass out of the $\Omega(n)$ π-appearances in $M_1 \cup M_2 \cup M_3$. There are only two significant cases, but to ease the reader, we split these two cases into the more natural larger number of cases, and observe at the end that most cases can be treated conceptually in the same way.

Case 1: Assume that $|M_1| \geq \varepsilon' n/3$, and further, for simplicity, that a constant fraction of the π-appearances in M_1 are in a single-box component. Then, on average, a dense box, out of the $\Theta(m')$ dense boxes, is expected to contain at least $\Theta(n/m') = \Theta(m') = \Theta(\sqrt{n})$ many π-appearances. Thus a random dense box B is likely to have $\Theta(\sqrt{n})$ many π-appearances, and hence, making queries to all points of such a box will enable us to find one such π-appearance. This takes an additional $n/m' = \Theta(\sqrt{n})$ queries, which is within the query budget.

Next, consider the case that a constant fraction of the π-appearances in M_1 belong to a configuration $\mathcal{C}$ that has more than one dense box (but only one connected component). By a similar argument, a random dense box is expected to participate in at least $\Theta(n/m')$ many π-appearances of copies of configuration-type $\mathcal{C}$. Since each dense box is part of at most $O(d^3)$ (constantly many) connected components of at most 4 dense boxes, sampling a random dense box B and querying all the indices in each of the components that contain at most 4 dense boxes and involve B, is likely to find a π-appearance with high probability. Each connected component is over at most $4n/m'$ indices, resulting in $O(n/m')$ queries.

Case 2: $|M_3| \geq \varepsilon' n/3$, and assume first that a constant fraction of the members in M_3 belong to copies of a configuration $\mathcal{C}$ of 3 components B_1, B_2, B_3, where each one is a single box. For our current working example, $\pi = (3,2,1,4)$, assume further that B_1 contains the $3,2$ legs of a π-appearance and B_2, B_3 contain its 1 and 4 legs, respectively (see Figure 1(G) for an example). In this case B_1 is not $(2,1)$-free (as B_1 contains the $(3,2)$-subpattern of π).

By an averaging argument, it follows that there is a dense box B for which: (a) B is far from $(2,1)$-free, and (b) there are corresponding dense boxes B_2, B_3 that, together with B, form a copy of the configuration $\mathcal{C}$. Now, a test follows easily. We test every dense box for $(2,1)$-freeness, which can be done in $O(\log n)$ queries per box, and hence in $\tilde{O}(m)$ in total. Then, by the guarantee above we will find the corresponding B and B_2, B_3 and a π-appearance in it (by Observation 2.6 with the trivial mapping).

A similar argument holds for a 3-component configuration $\mathcal{C}'$ in which one component contains more than one box, and for any configuration of 3 components.

Case 3: Assume now that $|M_2| \geq \varepsilon' n/3$, and that the corresponding configurations of the π-appearances in M_2 contain two single-box components B_1, B_2, where B_1 holds the first 3 legs of π and B_2 holds the 4-th leg. E.g., For $\pi = (3,2,1,4)$, the configuration $\mathcal{C}$ contains two boxes B_1, B_2 where B_1 contains the subpattern $(3,2,1)$ and B_2 is any nonempty box such that $B_1 < B_2$, (see Figure 1(H) for an illustration). An averaging argument, as made in Case 2, shows that there is a dense box B_1 for which (a) B_1 is far from $(3,2,1)$-free, and (b) there is a corresponding dense box B_2 that, together with B_1, forms a copy of the configuration $\mathcal{C}$. This suggests a test that is conceptually similar to the test in Cases 1 and 2. We test each box for being $(3,2,1)$-free. This can be done in $\operatorname{polylog} n$ queries (e.g., [8]). Then once finding a $(3,2,1)$ in B_1 for which (a) and (b) hold, $B_1 \cup B_2$ contains a π-appearance.

We note here that for the example above, we ended by testing for $(3,2,1)$-freeness which is relatively easy. For a different configuration or π, we might need to test B_1 for a different $\nu \in \mathcal{S}_3$, but this can be done for any $\nu \in \mathcal{S}_3$ using $O(\operatorname{polylog} n)$ queries [28]. Hence the same argument and complexity guarantee hold for any 2-component configuration $\mathcal{C}$ as above.

Case 4: A more complicated situation arises when $|M_2| \geq \varepsilon' n/3$, and the corresponding configurations of the π-appearances in M_2 are formed of two components D, B, with D holding 3 legs of π in 2 or 3 boxes (rather than in one box as in Case 3). E.g., $\pi = (4,2,1,3)$, and the configuration $\mathcal{C}$ as illustrated in Figure 1(E).

By a similar averaging argument to that made in Case 2, it follows that there is a dense box B_1 for which (a) there are dense boxes B_2, B_3 forming a copy D' of D with B_1, and a dense box B such that the configuration formed by D', B is a copy of $\mathcal{C}$, and (b) there are $\Omega(n/m) = \Omega(\sqrt{n})$ ϕ-legged $(3,2,1)$-appearances in D', where ϕ is consistent with the leg mapping that is induced by the configuration $\mathcal{C}$. This implies a conceptually similar test to that of the simpler Case 3 above - we test each of the $O(m)$ components D for $(3,2,1)$-freeness, and then with the existence of the corresponding box B we find a π-appearance. However, this is not perfectly accurate: the algorithm for finding $\nu = (3,2,1)$ in D', although efficient, might find a $(3,2,1)$-appearance where the 3 legs appear in B_1 or in $B_1 \cup B_2$. But this does not extend with B to form a π-appearance, as the leg mapping is not consistent with the one that is induced by $\mathcal{C}$. Namely, unlike before, we do not only need to find a ν-appearance in D but rather a ϕ-legged ν-appearance with respect to a fixed mapping ϕ (that in this case maps each leg to a different box in the component D').

To resolve the problem we need to efficiently test ϕ-legged ν-appearances in multi-boxed components. This, however, we currently do not know how to do. Instead, we design a test that either finds a ϕ-legged ν-appearance, or finds the original π-appearance. This is done using the procedure $\mathsf{AlgTest}_\pi(\nu, \phi, D, m, \varepsilon)$ that will be described in Section 4.

Case 5: The last case that we did not consider yet is when most of the π-appearances are in a configuration containing more than one component, with at least two components containing two (or more) legs each. For $\pi \in \mathcal{S}_4$ the only such case is when the configuration $\mathcal{C}$ contains exactly two components, each containing exactly two legs of π. Returning to our working example with $\pi = (3, 2, 1, 4)$, such examples are depicted e.g., in Figure 1(B) or 1(F). For the explanation below, we will discuss the case that the configuration $\mathcal{C}$ is as in Figure 1(F). Namely, it contains components D_1 that is above D_2, with two boxes each $D_1 = \{B_1, B_4\}$ and $D_2 = \{B_2, B_3\}$, and so that every box contains exactly one leg of π (boxes are numbered by order from left to right in $G_{m',m'}$). Our goal is to find two copies D'_1, D'_2 of the components D_1, D_2 respectively, that form a copy of $\mathcal{C}$, and to find a ϕ_1-legged appearance of $(1, 2)$ in D'_1, and a ϕ_2-legged appearance of $(2, 1)$ in D'_2, so that Observation 2.6 will implies that these two appearances form together a π appearance. Indeed, an averaging argument shows that there are D'_1, D'_2 as above, with D'_i containing $\Omega(n/m)$ ϕ_i-legged appearances of ν_i for $i = 1, 2$. However, we do not know that sampling a pair D'_1, D'_2 in some way, will result in such a good pair. Rather, we are only assured of the existence of only one such pair! Hence, in this case we need to test *every* component D', and for every $\nu \in \mathcal{S}_2$, and for every leg mapping ϕ, for a ϕ-legged ν-appearance in D' in order to find such an asserted pair of components. Such restricted ν-appearances can be tested in $O(\log n)$ queries per component. Therefore, this takes $\tilde{O}(m)$ queries in total. The same argument holds for any $\pi \in \mathcal{S}_4$, and for every configuration that is consistent with Case 5.

Concluding remarks

- At some places in the algorithm above, we had to test for ν-appearances (or restricted ν-appearances) in "dense" subgrids of G_n. For this, we need that all our algorithms are ER, which will be implicitly clear from the description. We also need to take care of reducing the total error when we do non-constant number of by tests, or want to guarantee a large success probability for a large number of events - this is done by trivial amplification that results in a multiplicative polylog n factor.
- In Case 1, we reduced the problem of finding a π-appearance in G_n, that is assumed to be ε-far from π-free, to the same problem on a subrange of the indices (formed by a small component) of size $\Theta(n/m)$ (with a smaller but constant distance parameter $\varepsilon' < \varepsilon$). For the setting of $m = \sqrt{n}$, solving the problem on the reduced domain was trivially done by querying all indices in the subrange. In the general algorithm, where our goal is a query complexity of $n^{o(1)}$, we set $m = n^\delta$ for an appropriately small δ and apply self-recursion in Case 1.
- In Cases 2, 3, 4 we end up testing ν-freeness for $\nu \in \mathcal{S}_2 \cup \mathcal{S}_3$ in dense boxes, or ϕ-legged ν-freeness of such ν in components of multiple dense boxes. An average argument shows that this can simply be done by sampling one box or component, and making queries to all indices therein. This however, is true only for $\pi \in \mathcal{S}_4$.
  Case 5 is different: here sampling a small number of components does not guarantee an expected large number of the corresponding appearances. This is the reason that we need to test *all* components with at most 2 dense boxes, for ϕ-legged ν-freeness, and for every $\nu \in \mathcal{S}_2$ and leg mapping ϕ. Algorithm $\mathsf{AlgTest}_\pi(\nu, \phi, D, m, \varepsilon)$ can do this for any

$\nu \in \mathcal{S}_2 \cup \mathcal{S}_3$ in n^δ queries for an arbitrary small constant δ. Since we have to do it in Case 5, we may do the same in cases 2, 3, 4 as well! As a result, the algorithm above will contain only two cases: Case 1 where we reduce the problem to the same problem but on a smaller domain, and the new Case 2 where we test *every* small component for ϕ-legged ν-appearance for every $\nu \in \mathcal{S}_2 \cup \mathcal{S}_3$ and every leg mapping ϕ – namely a case in which we reduce the problem to testing (restricted appearances) for smaller patterns.

- In view of the comment above, the idea behind improving the complexity to n^δ for constant $0 < \delta < 1$ is obvious: Choosing $m = n^{\delta/2}$ will result in an $m \times m$ grid, where Layering can be done in $\tilde{O}(n^{\delta/2})$ queries. Then, Case 2 will be done in an additional n^δ queries by setting a query complexity for $\mathsf{AlgTest}_\pi(\nu, \phi, D, m, \varepsilon)$ to be $n^{\delta/2}$ per component. The self-recursion in Case 1 will result in the same problem over a range of n/m. For the fixed $m = n^{\delta/2}$, this will result in a recursion depth of $2/\delta$, after which the domain size will drop down to m and allow making queries to all corresponding indices. This results in a total of $\tilde{O}(n^\delta)$ queries, including the amplification needed to account for the accumulation of errors and deterioration of the distance parameter at lower recursion levels.
- **Generalized testing and testing beyond $k = 4$.** Applying the same ideas to $\pi \in \mathcal{S}_k$, $k \geq 5$ works essentially the same way, provided we can test for ϕ-legged ν-freeness of $\nu \in \mathcal{S}_r$ for $r < k$. This we know how to do for $\nu \in \mathcal{S}_3$ but not beyond. In particular, one difficulty is that after gridding, a superlinear number of non-empty boxes do not guarantee such appearance, as Lemma 3.1 does not apply. However, for our goal of testing π-freeness for $\pi \in \mathcal{S}_k$, we can relax the task of finding ϕ-legged ν-freeness of $\nu \in \mathcal{S}_r, r \leq k$ to the following problem which we call "generalized-testing ν w.r.t. π", denoted $\mathsf{GeneralizedTesting}_\pi(\nu, D, \phi)$: The inputs are a permutation $\nu \in \mathcal{S}_r$, a component D, and a leg mapping ϕ. Our goal is to find either a ϕ-legged ν-appearance **OR** a π-appearance in D. The way we solve this generalized problem is very similar, conceptually, to the way we solve the unrestricted problem. This will be defined formally in the next section.

4 Generalized testing of forbidden patterns

In this section, we formally define the problem of testing (or deciding) freeness from ν-appearances with a certain leg-mapping. We then provide an algorithm for a relaxation of this testing problem, which we call $\mathsf{GeneralizedTesting}_\pi(\nu, \phi, D)$. This will imply, in turn, our algorithm for testing π-freeness.

Recall that G_n denotes the $n \times n$ grid that represents the input function $f : [n] \to \mathbb{R}$. Let $G_{\ell,\ell}$ be a partition of G_n into a grid of boxes for arbitrary $\ell \geq 1$, and D be a connected component in $G_{\ell,\ell}$ containing t boxes $B_1, \ldots, B_t$. Let $\nu \in \mathcal{S}_r$, and let $\phi : [r] \mapsto [t]$ be an arbitrary mapping of the legs of ν into the boxes of D, where $t \leq r$. We say that $1 \leq i_1 \leq \ldots \leq i_r \leq n$ is a ϕ-legged ν-appearance if $(i_1, \ldots, i_r)$ forms a ν-appearance in G_n such that $(i_j, f(i_j)) \in B_x$ for $x = \phi(j)$, $j = 1, \ldots, r$. That is, the corresponding legs of the ν-appearance are mapped into the boxes given by ϕ. For example, consider Figure 1(B), $\nu = (3, 2, 1, 4)$, and D the component formed by the two boxes in the same layer. The function ϕ maps the 3 and 2-legs of the appearance to the left box and the 1 and 4-legs to the right box.

For ϕ, ν as above, the property of containing a ϕ-legged ν-appearance is a generalization of the problem of pattern-freeness; taking $\ell = 1$, $G_{\ell,\ell}$ is just G_n itself viewed as one single box D. Any ν-appearance in G_n is a ϕ-legged ν-appearance for the constant function $\phi \equiv D$.

The complexity of testing ϕ-legged ν-freeness is not clear and was not previously explicitly studied. For permutations of length 2 as well as for longer monotone permutations, its complexity is identical to unrestricted testing and can be done in $\text{polylog}\, n$ queries. For length 3 non-monotone permutations it can be done in $\text{polylog}\, n$ queries using similar ideas as in [28]. For larger $k \geq 4$ the complexity is open.

While testing ϕ-legged ν-freeness is interesting in its own, we encounter it only as a sub-problem in the testing of standard π-freeness. This motivates the following definition.

Let $\pi \in \mathcal{S}_k$, G_n be fixed. Further let $G_{\ell,\ell}$ be a decomposition of G_n into boxes and D be a t-boxed *one component* over boxes $(B_1, \ldots, B_t)$, $t \leq r$, in $G_{\ell,\ell}$. For inputs $\nu \in \mathcal{S}_r$, and $\phi : [r] \mapsto \{B_1, \ldots, B_t\}$, the problem $\mathsf{GeneralizedTesting}_\pi(\nu, \phi, D)$, is to find a ϕ-legged ν-appearance in D **OR** to find any (unrestricted) π-appearance. The distance of D from being ϕ-legged ν-free is defined naturally. The distance for the generalized testing w.r.t π, $\mathsf{GeneralizedTesting}_\pi(\nu, \phi, D)$, is defined as the distance from being free of ϕ-legged ν-appearance regardless of the π-appearances.

Our algorithm for $\mathsf{GeneralizedTesting}_\pi(\nu, \phi, D)$ is called $\mathsf{AlgTest}_\pi(\nu, \phi, D, m, \varepsilon)$ (presented in Algorithm 1), where $\pi \in \mathcal{S}_k$ is fixed. The algorithm has inputs: $\nu \in \mathcal{S}_r$, $r \leq k$, a component D in a decomposition $G_{\ell,\ell}$ of G_n that contain boxes $B_1, \ldots, B_t$, $t \leq r$, and a function $\phi : [r] \to \{B_1, \ldots B_t\}$. In addition, it gets a distance parameter ε, and a free parameter m that that is used to control the query complexity which will be $\tilde{O}(m^r)$ for $m \geq n^{\Omega(1/\log\log\log n)}$. With high probability, the algorithm either finds a π-appearance or a ϕ-legged ν-appearance in D, if D is ε-far from being free of ϕ-legged ν-appearances.

The algorithm is recursive. A recursion is done by reducing ν to smaller length patterns, and/or self-reduction to the same ν but on a smaller length box D'. The base cases are when the length of D is small enough to allow queries to all indices in D, or when $\nu \in \mathcal{S}_2$, in which case the algorithm is reduced to testing monotonicity.

The permutation $\pi \in \mathcal{S}_k$ is fixed and hardwired into the algorithm. G_n is assumed to fixed and not part of the recursion. The actual input is the t-boxed component D in a grid of boxes $G_{\ell,\ell}$ over G_n (that is, a sub-function of the original function f).

In general, the complexity of testing ϕ-legged ν-appearances may depend on ϕ and ν. Our algorithm does not use any structure of π. The only role of π in the algorithm is to ensure that after gridding D, the resulting grid $D_{m,m}$ contains only a linear in m number of marked boxes (as otherwise, by Lemma 3.1, a π-appearance is guaranteed).

Finally, and as we already pointed out, the algorithm for $\mathsf{GeneralizedTesting}_\pi(\nu, \phi, D)$ will allow us to test for π-freeness in G_n by calling $\mathsf{AlgTest}_\pi(\pi, \phi, G_n, m, \varepsilon)$, where ϕ is the constant function that maps each of the k legs to the single box G_n.

The following theorem asserts the correctness of $\mathsf{AlgTest}_\pi(\pi, \phi, G_n, m, \varepsilon)$ and the corresponding query complexity. We assume that ℓ and a corresponding component D in the grid of boxes $G_{\ell,\ell}$, inside G_n, is given.

▶ **Theorem 4.1.** *Let $\varepsilon \in (0,1)$ and $\nu \in \mathcal{S}_r$, $r \leq k$. Let $D = \mathsf{box}(S, I)$ be a connected component in $G_{\ell,\ell}$, composed of boxes $B_1, \ldots, B_t$, $t \leq r$ and $\phi : [r] \to \{B_1, \ldots B_t\}$. If D is ε-far from being free of ϕ-legged ν-appearances, then $\mathsf{AlgTest}_\pi(\nu, \phi, D, m, \varepsilon)$ finds, either a ϕ-legged ν-appearance or a π-appearance, with probability at least $1 - o(1)$. Its query complexity is $\tilde{O}(n^{\eta r})$, for $\eta \in (0,1)$ and $m = n^\eta$.*

We note that since $\mathsf{AlgTest}_\pi(\nu, \phi, D, m, \varepsilon)$ either finds π or a ϕ-legged ν-appearance in D, then if D is free of ϕ-legged ν-appearance, the algorithm will never return such an appearance. As a result, the corollary below follows by calling $\mathsf{AlgTest}_\pi(\pi, \phi, G_n, m, \varepsilon)$.

Algorithm 1 $\mathsf{AlgTest}_{\pi}(\nu, \phi, D, m, \varepsilon)$.

Input: pattern $\nu \in \mathcal{S}_r$; D is a component in $G_{\ell,\ell}$ containing boxes $B_1, \ldots, B_t$ for $1 \leq t \leq r$; the function $\phi : [r] \mapsto \{B_1, \ldots, B_t\}$ is a leg-mapping of ν into the boxes of D.

Goal: Find a ϕ-legged ν-appearance **or** an unrestricted π-appearance, in D.

Let $S = \cup_{i\in[r]} \mathrm{St}(B_{\phi(i)})$ and $I = \cup_{i\in[r]} L(B_{\phi(i)})$ define $\mathsf{box}(S, I)$ in $G_{\ell,\ell}$ that contains D (or the subcomponent of D where a ν-appearance should be found).

Base cases:

1. If $|S| \leq m$ query all indices in S. **Output** "π-appearance is found" or "ϕ-legged ν-appearance is found" if one of these is found, otherwise **output** "not found".
2. If $r \leq 2$, use the test for restricted appearance of 2-patterns. This is easy and will not be described in this manuscript. If $r = 1$, **output** "ϕ-legged ν-appearance is found" if $\phi(1)$ contains a point. If $k = 1$, **output** "π is found" if D contains a point.
3. If the sampled points in D already contains a ϕ-legged ν-appearance, or contains a π-appearance then **output**: "found ϕ-legged ν-appearance" or "found π-appearance" respectively.

Gridding D: We set $\beta = \frac{\varepsilon}{200k\kappa(k)}$. Call Gridding on $\mathsf{box}(S, I)$ with parameters: (S, I, m, β). The details on the procedure Gridding can be found in the full version [30]. As a result, we obtain a decomposition of $\mathsf{box}(S, I)$ (and D) into an $m' \times m'$ grid of sub-boxes $D_{m',m'}$, $m \leq m' \leq 2m$, where a subset of boxes are *marked* and a subset of the marked boxes are *dense*.

Simple case: If $D_{m',m'}$ contains more than $\kappa(k) \cdot m'$ marked sub-boxes then **output** "found π-appearance".

Sparsification: Delete every stripe and layer in $D_{m',m'}$ that contains more than $d = 100k\kappa(k)/\varepsilon$ marked sub-boxes. Delete all non-dense sub-boxes.

Multi-component configurations: For every possible configuration $\mathcal{C}$ of sub-boxes that is consistent with ϕ such that $\mathcal{C}$ forms components $\mathcal{C}_1, \ldots \mathcal{C}_p$, $p > 1$, the pair (ν, ϕ) define sub-permutations of ν : $\nu_1, \ldots, \nu_p$ and subfunctions of ϕ : $\phi_1, \ldots, \phi_p$ on $\mathcal{C}_1, \ldots \mathcal{C}_p$ respectively.

Let c denote the number of distinct configurations with at most r components.

Recursively **call** $\mathsf{AlgTest}_{\pi}(\nu_i, \phi_i, D_i, m, \varepsilon')$ with distance parameter $\varepsilon' = \frac{9\varepsilon}{10kcr^2 \cdot r! \cdot (2d)^r}$ for *every* component D_i, where D_i is a copy of $\mathcal{C}_i$ in $D_{m',m'}$, and is contained in D. Note that the recursive call is done for smaller length patterns ν_is.

Output "found ϕ-legged ν-appearance" if for a copy $(D_1, \ldots, D_p)$ of $(\mathcal{C}_1, \ldots, \mathcal{C}_p)$, the region D_i contains a ϕ_i-legged ν_i-appearance for each $i = 1, \ldots, p$. Or **output** "found π-appearance" if a π-appearance is found among the sampled points.

One component configurations: Let $\mathcal{A}$ be the set of all possible copies of configurations $\mathcal{C}$ in $D_{m',m'}$ for which $\mathcal{C}$ forms *one* component, and that are contained in D. Note that $\mathcal{A}$ contains $O(m)$ such copies.

loop $\frac{\log^3 n}{\varepsilon^r}$ times:

Sample a member D' from $\mathcal{A}$. For each ϕ-consistent mapping ϕ', recursively **call** $\mathsf{AlgTest}_{\pi}(\nu, \phi', D', m, \varepsilon'')$ with $\varepsilon'' = \frac{9\varepsilon}{20k \cdot (2d)^r \cdot (r-1)! \cdot r^r}$.

▷ A mapping ϕ' from the legs of a ν-appearance to the sub-boxes in $D_{m',m'}$, is ϕ-consistent if for each i-leg, $i = 1, \ldots, r$, the sub-box $\phi'(i)$ is contained in the box $\phi(i)$.

end loop

If no output is declared in any of the previous steps, **output** "not-found".

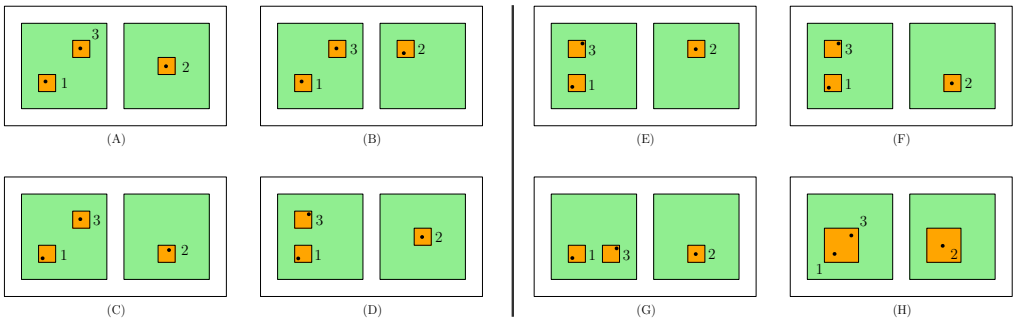

Figure 2 Different configurations for (1,3,2)-appearances.

▶ **Corollary 4.2.** *There is a 1-sided error test for π-freeness, for every $\pi \in \mathcal{S}_k$ of query-complexity $\tilde{O}(n^{\delta k})$ for arbitrary $\delta > 1/\log\log\log n$.*

4.1 Proof of Correctness

In this section, we give a description of the algorithm and the proof sketch for the first non-base case of testing ϕ-legged ν-freeness of $\nu \in \mathcal{S}_r$, $r = 3$, with respect to an arbitrary $\pi \in \mathcal{S}_k$ and for some fixed $k \geq 4$. The full proof of Theorem 4.1 can be found in the full version [30].

4.1.1 An example for $\nu \in \mathcal{S}_3$

For this exposition, we fix $\nu = (1, 3, 2)$, and D being composed of 2 boxes B_1 B_2, in the same layer, where B_1 is to the left of B_2, and ϕ maps the $1, 3$ legs of ν to B_1, and the 2-leg to B_2. See Figure 2(D) for illustration of one such case. In the figure, the green boxes represent B_1 and B_2. The orange boxes indicate the subboxes in the finer grid formed when gridding is called on the green boxes.

We note that Figure 2(D) illustrates the hardest case for $\nu \in \mathcal{S}_3$. There are additional one-component configurations in which the boxes are in the same stripe or layer, but these turn out to be much easier. We will set $m = m(n)$ to be defined later and express the complexity as a function of m. We do not specify π since, as explained above, π is only needed at Step 4 when the number of marked boxes is superlinear in m' in some recursive call. The same argument holds for any $\pi \in \mathcal{S}_k,\ k \geq 4$.

Algorithm for $\nu = (1, 3, 2)$ and B_1, B_2 as above:

1. We assume that B_1, B_2 have at most $s \leq n$ indices each, and that the distance of $B_1 \cup B_2$ from being free from ϕ-legged ν-appearance is at least $\varepsilon = \Omega(1)$. In particular B_1, B_2 are dense. We start in Step 3 of Algorithm 1 where we do gridding of the union of B_1 and B_2 into a $m' \times m'$ grid, $D_{m',m'}$, of sub-boxes (each having roughly at most s/m' indices), where $m \leq m' \leq 2m$. We either find a π-appearance among the sampled points or we may assume, after Steps 4, 5 that there are $O(m')$ dense sub-boxes in $B_1 \cup B_2$ and that each layer and each stripe contains $O(1)$ dense boxes. This is obtained by an averaging argument and is described in the formal proof in the full version. It shows that if $B_1 \cup B_2$ contains a large matching of ϕ-legged ν-appearances, then so does the restricted domain after deleting points from non-dense boxes, and deleting layers and stripes that contain too many dense boxes. This steps takes $\tilde{O}(m)$ queries (the complexity of gridding).
2. A ϕ-legged ν-appearance in $B_1 \cup B_2$ can be in 8 possible configurations in the grid $D_{m',m'}$, as depicted in Figure 2. Consider first $\mathcal{C}_1, \ldots, \mathcal{C}_4$ as in Figure 2(A)-(D), that form 2 or 3 components each. For these, a ϕ-legged ν-appearance in $B_1 \cup B_2$ decomposes into two

or three subpatterns, and for which any restricted appearances in the corresponding components results in a ϕ-legged ν-appearance. E.g., in Figure 2(B) the configuration $\mathcal{C}_2$ contains one component $D_1 = (B_{1,3}, B_{2,2})$, where $B_{1,3} \in B_1, B_{2,2} \in B_2$, and another single boxed component $B_{1,1} \in B_1$, where $B_{i,j}$ is the orange subbox contained within the green box B_i and such that the j-th leg belongs to $B_{i,j}$ for $i \in [2], j \in [3]$.

In Step 6, we test each of the $O(m)$ many copies of D_1 for a ϕ'-legged $(2,1)$-appearance for which $\phi'(2) = B_{1,3}$ and $\phi'(1) = B_{2,2}$. Then for any such D_1-copy in which a such ϕ'-legged $(2,1)$-appearance is found, any nonempty dense box $B_{1,1}$ forming with D_1 a copy of $\mathcal{C}_2$ results in a ϕ-legged ν-appearance.

Since this is a reduction to generalized 2-pattern appearance, the recursion stops here with $O(\log n)$-complexity per copy of D_1. Hence, altogether this will contribute a total of $\tilde{O}(m)$ queries. The same argument holds for any of $\mathcal{C}_i,\ i = 1,2,3,4$.

If a desired appearance is found, then clearly a correct output is produced.

Finally, if indeed (B_1, B_2) contains $\Omega(s)$ (that is, linear in the length of $B_1 \cup B_2$) many ϕ-legged ν-appearances that are consistent with one of the configurations $\mathcal{C}_i, i = 1,2,3,4$, then there will be such a D_1 and corresponding $B_{1,1}$ that together contribute $\Omega(s/m)$ (that is – linear in the domain size of D_1) such subpattern appearances by an averaging argument.

We note that for the more general case of $r > 3$, the reduction will be done in higher complexity per component (that is dependent on m rather than just $O(\log n)$).

3. Consider now $\mathcal{C}_i, i = 5,6,7,8$ that form 1-component each (with 2 or 3 orange subboxes). In these cases, if such appearances contribute ε' to the total distance, then a simple averaging argument shows that for a uniformly sampled component, its distance will be linear from being free from ϕ-legged ν-appearances. Hence in Step 9, sampling such a component will enable us to recursively find a ϕ-legged ν-appearance with high probability. Since the length of a component on which the recursive call is made is $\Theta(s/m)$, the complexity of this step is $\tilde{O}(q(s/m))$, where $q(*)$ is the complexity of the algorithm, for the case of $\nu \in \mathcal{S}_3$, in terms of the length of D.

The correctness of the algorithm follows from the fact that if D is indeed far from being ϕ-legged ν-free, then it must be that there are linearly many ϕ-legged ν-appearances in at least one of the 8 configurations discussed above, and for each case, either a π-appearance or a ϕ-legged ν-appearance is found, by induction.

The base case is for $m = n$ for which the trivial algorithm that queries all indices is obviously correct, and has complexity n. To understand the query complexity for general m, let a be the smallest integer for which $m^a \geq n$. We express the query complexity for functions over a domain of length n, and parameter m as $q(m,a)$ for a as defined above. We get that, omitting polylogarithmic factors, $q(m,1) = m = s$ (that is the base case above), and $q(m,a) = m + m + q(m, a-1)$ where the first summand is the number of queries made by the gridding, the second is the number of queries made by Step 2 above (corresponding to Step 6 in AlgTest), and the last is the query complexity of the recursive call on subbox of length s/m with the same m, for which the corresponding $a' = a - 1$.

The recursion equation implies that $q(a,m) = \tilde{O}(am)$, which implies a query complexity n^δ by choosing $m = n^\delta$. We note that this is true as long as m (and hence δ) is large enough, as the recursion depth is a and there is an exponential deterioration of the distance parameter at each recursive call in Steps 2 and 3 above (corresponding to Steps 6 and 9 in AlgTest). However, for arbitrary constant $\delta < 1$ we can achieve complexity n^δ and this is true even for $\delta = 1/\log\log\log n$ for which the complexity becomes $n^{o(1)}$.

A final note that is due here, is that for $\nu \in \mathcal{S}_r$ with $r \geq 4$, the complexity of Step 2 above is not $O(\log n)$, and thus the complexity dependence on m becomes important.

References

1. Shlomo Ahal and Yuri Rabinovich. On complexity of the subpattern problem. *SIAM J. Discret. Math.*, 22(2):629–649, 2008. doi:10.1137/S0895480104444776.
2. Michael H. Albert, Robert E. L. Aldred, Mike D. Atkinson, and Derek A. Holton. Algorithms for pattern involvement in permutations. In Peter Eades and Tadao Takaoka, editors, *Algorithms and Computation, 12th International Symposium, ISAAC 2001, Proceedings*, volume 2223 of *Lecture Notes in Computer Science*, pages 355–366. Springer, 2001.
3. Noga Alon and Ehud Friedgut. On the number of permutations avoiding a given pattern. *J. Comb. Theory, Ser. A*, 89(1):133–140, 2000. doi:10.1006/jcta.1999.3002.
4. Noga Alon, Michael Krivelevich, Ilan Newman, and Mario Szegedy. Regular languages are testable with a constant number of queries. *SIAM J. Comput.*, 30(6):1842–1862, 2000. doi:10.1137/S0097539700366528.
5. Richard Arratia. On the Stanley-Wilf conjecture for the number of permutations avoiding a given pattern. *The Electronic Journal of Combinatorics*, 6:1–4, 1999.
6. Aleksandrs Belovs. Adaptive lower bound for testing monotonicity on the line. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2018*, pages 31:1–31:10, 2018.
7. Omri Ben-Eliezer and Clément L. Canonne. Improved bounds for testing forbidden order patterns. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018*, pages 2093–2112. SIAM, 2018.
8. Omri Ben-Eliezer, Clément L. Canonne, Shoham Letzter, and Erik Waingarten. Finding monotone patterns in sublinear time. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019*, pages 1469–1494. IEEE Computer Society, 2019.
9. Omri Ben-Eliezer, Shoham Letzter, and Erik Waingarten. Optimal adaptive detection of monotone patterns. *CoRR*, abs/1911.01169, 2019. arXiv:1911.01169.
10. Benjamin Aram Berendsohn, László Kozma, and Dániel Marx. Finding and counting permutations via CSPs. *Algorithmica*, pages 1–26, 2021.
11. Donald J. Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In Usama M. Fayyad and Ramasamy Uthurusamy, editors, *Knowledge Discovery in Databases: Papers from the 1994 AAAI Workshop, 1994. Technical Report WS-94-03*, pages 359–370. AAAI Press, 1994.
12. Arnab Bhattacharyya, Elena Grigorescu, Kyomin Jung, Sofya Raskhodnikova, and David P. Woodruff. Transitive-closure spanners. *SIAM Journal on Computing*, 41(6):1380–1425, 2012. doi:10.1137/110826655.
13. Miklós Bóna. *Exact and asymptotic enumeration of permutations with subsequence conditions*. PhD thesis, Massachusetts Institute of Technology, 1997.
14. Miklós Bóna. The solution of a conjecture of Stanley and Wilf for all layered patterns. *J. Comb. Theory, Ser. A*, 85(1):96–104, 1999. doi:10.1006/jcta.1998.2908.
15. Deeparnab Chakrabarty and C. Seshadhri. Optimal bounds for monotonicity and Lipschitz testing over hypercubes and hypergrids. In *Proceedings of the ACM Symposium on Theory of Computing (STOC) 2013*, pages 419–428, 2013.
16. Kashyap Dixit, Sofya Raskhodnikova, Abhradeep Thakurta, and Nithin Varma. Erasure-resilient property testing. *SIAM J. Comput.*, 47(2):295–329, 2018. doi:10.1137/16M1075661.
17. Yevgeniy Dodis, Oded Goldreich, Eric Lehman, Sofya Raskhodnikova, Dana Ron, and Alex Samorodnitsky. Improved testing algorithms for monotonicity. In *RANDOM-APPROX, 1999, Proceedings*, pages 97–108, 1999.

18 Funda Ergün, Sampath Kannan, Ravi Kumar, Ronitt Rubinfeld, and Mahesh Viswanathan. Spot-checkers. *Journal of Computer and System Sciences*, 60(3):717–751, 2000. doi:10.1006/jcss.1999.1692.

19 Eldar Fischer. On the strength of comparisons in property testing. *Inf. Comput.*, 189(1):107–116, 2004. doi:10.1016/j.ic.2003.09.003.

20 Jacob Fox. Stanley-Wilf limits are typically exponential. *CoRR*, abs/1310.8378, 2013. arXiv:1310.8378.

21 Jacob Fox and Fan Wei. Fast property testing and metrics for permutations. *Comb. Probab. Comput.*, 27(4):539–579, 2018. doi:10.1017/S096354831800024X.

22 Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *J. ACM*, 45(4):653–750, 1998. doi:10.1145/285055.285060.

23 Sylvain Guillemot and Dániel Marx. Finding small patterns in permutations in linear time. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014*, pages 82–101. SIAM, 2014.

24 Eamonn J. Keogh, Stefano Lonardi, and Bill Yuan-chi Chiu. Finding surprising patterns in a time series database in linear time and space. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2002*, pages 550–556. ACM, 2002.

25 Martin Klazar. The Füredi-Hajnal conjecture implies the Stanley-Wilf conjecture. In *Formal power series and algebraic combinatorics*, pages 250–255. Springer, 2000.

26 Adam Marcus and Gábor Tardos. Excluded permutation matrices and the Stanley-Wilf conjecture. *J. Comb. Theory, Ser. A*, 107(1):153–160, 2004. doi:10.1016/j.jcta.2004.04.002.

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

28 Ilan Newman, Yuri Rabinovich, Deepak Rajendraprasad, and Christian Sohler. Testing for forbidden order patterns in an array. *Random Struct. Algorithms*, 55(2):402–426, 2019. doi:10.1002/rsa.20840.

29 Ilan Newman and Nithin Varma. New sublinear algorithms and lower bounds for LIS estimation. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021*, volume 198 of *LIPIcs*, pages 100:1–100:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

30 Ilan Newman and Nithin Varma. Strongly sublinear algorithms for testing pattern freeness. *CoRR*, abs/2106.04856, 2021. arXiv:2106.04856.

31 Ramesh Krishnan S. Pallavoor, Sofya Raskhodnikova, and Nithin Varma. Parameterized property testing of functions. *ACM Trans. Comput. Theory*, 9(4):17:1–17:19, 2018. doi:10.1145/3155296.

32 Pranav Patel, Eamonn J. Keogh, Jessica Lin, and Stefano Lonardi. Mining motifs in massive time series databases. In *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002)*, pages 370–377. IEEE Computer Society, 2002.

33 Ronitt Rubinfeld and Madhu Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM J. Comput.*, 25(2):252–271, 1996. doi:10.1137/S0097539793255151.

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

35 Michael E. Saks and C. Seshadhri. Estimating the longest increasing sequence in polylogarithmic time. *SIAM Journal on Computing*, 46(2):774–823, 2017.