Testing Permutation Avoidance in the Bounded Range Setting

Nathan S. Sheffield, Alek Westover, Zoe Xi

October 2024

Abstract

Given $f : [n] \to \mathbb{R}$[1], and a permutation $\pi : [k] \to [k]$, we say that f contains an instance of π if there are indices $1 \leq i_1 < i_2 < \cdots < i_k \leq n$ such that $f(i_j) < f(i_\ell)$ whenever $\pi(j) < \pi(\ell)$. We say that f is π-***avoiding*** if f contains no instances of π, and that f is ε-***far*** from π-avoiding if any π-avoiding $g : [n] \to \mathbb{R}$ has $\Pr_x[f(x) \neq g(x)] \geq \varepsilon$.

Recently, Newman and Varma [NV24] gave a property tester for π-avoidance, for any constant length permutation π, with running time $n^{O(1/\log\log\log n)}$ (assuming $\varepsilon \geq \Omega(1)$). Newman and Varma left as an open question improving on this result, and suggested that considering the ***bounded range*** setting — where we require the output of f to lie in some small set rather than all of $\mathbb{R}$ — could be a particularly tractable easier problem to work on first. In this note we give an exposition of Newman and Varma's property tester, and then present novel property testers for permutation avoidance in the bounded range setting.

1 Introduction

In ***property testing*** (introduced by [BLR90]) we have an object f — which will be a function $f : [n] \to \mathbb{R}$ for us — and would like to tell whether f satisfies a certain predicate P which we call a ***property***. For instance, we could consider the predicate P which checks whether $f(i) < f(i+1)$ for all $i \in [n-1]$. We are interested in algorithms that query f at a sublinear number of locations, and then output whether f has property P. As stated, our goal is too ambitious: it could be the case that f is equal to a function with property P in all but one location, in which case we would need to query a constant fraction of the domain of f to be confident that f doesn't have property P. To eliminate this issue, we consider an easier problem: ***accept*** f that have property P, and ***reject*** with constant probability f which are ε-***far*** from all functions with property P, where $\mathsf{dist}(f, g) = \Pr_x[f(x) \neq g(x)]$.

When building a property tester, we are generally interested in minimizing the number of queries made by the algorithm, rather than the running-time of an algorithm. We will consider ***adaptive*** testers by default, but some of our results also hold for ***non-adaptive*** property testers which make all of their queries up front instead of getting to choose their queries based on the outcomes of earlier queries.

Related Work

There is a long line of work studying property testing algorithms for permutation avoidance, for example: [Ben19] [BC18] [New+19] [Ras14]. The prior work can be summarized as follows, using S_k to denote the set of permutations on $[k]$, and taking ε to be constant while $n \to \infty$:

- For every $k \leq 3, \pi \in S_k$, there is an adaptive π-avoidance tester using polylog(n) queries. [New+19]
- For constant k and the identity permutation $\pi(i) = i$ (or the reverse $\pi(i) = k - i + 1$), there is an adaptive π-avoidance tester using $O(\log(n))$ queries. [BLW19].
- For $k \in O(1), \pi \in S_k$, there is an adaptive π-avoidance tester using $n^{O(1/\log\log\log n)}$ queries. [NV24]
- For $k \in O(1), \pi \in S_k$, there is a non-adaptive π-avoidance tester using $O(n^{1-1/k})$ queries. [New+19]

[1] Throughout the paper we will use $[x]$ to denote the set $\{1, 2, \ldots, \lfloor x \rfloor\}$.

- For constant k there are some $\pi \in S_k$ for which a non-adaptive π-avoidance tester requires $\Omega(n^{1-1/k})$ queries. [BC18]
- An adaptive 12-avoidance tester requires $\Omega(\log n)$ queries. [Fis04]

Some people conjecture that for any constant k and $\pi \in S_k$ there is a polylog(n)-query π-avoidance tester, but this remains an open question.

Paper Outline

In this note we will begin by summarizing Newman and Varma's property tester, for educational purposes. We will try to give a short intuitive explanation of their algorithm, but recommend the interested reader to also take a look at Newman and Varma's excellent exposition of their algorithm.

Afterwords, we will present some modest novel results on testing π-avoidance in the ***bounded range setting***. Specifically, in the bounded range setting we restrict to considering functions $f : [n] \to [R]$ for some small $R < n^{o(1)}$, rather than considering all functions $f : [n] \to \mathbb{R}$. In the bounded range setting our main results are:

Theorem 1.1. There is a 3412-avoidance tester for $f : [n] \to [R]$ with query complexity

$$O\Big(\varepsilon^{-1} R \log n \left(\log^3 R + \log^2 R \log \varepsilon^{-1}\right) \log\log n\Big).$$

Theorem 1.2. If π is a permutation of length k, there is a non-adaptive π-avoidance tester for $f : [n] \to [R]$ which uses $O(\varepsilon^{-1} R^k)$ queries.

2 Preliminaries

Before continuing, we establish a few tools that will be useful throughout the paper. One extremely important fact that will be used constantly is that if f is ε-far from being π-avoiding, then f contains a large number of disjoint copies of π. Specifically,

Fact 2.1. If f is ε-far from π-avoiding, then f contains $n\varepsilon/|\pi|$ disjoint copies of π.

Proof. Find a maximal set of disjoint copies of π. Let S be the set of indices involved in all of those copies. We would like to show that $|S| \geq \varepsilon n$. To do so, it suffices to show that we can make f π-avoiding by modifying only the indices in S. And indeed, we can do so! Let i be some arbitrary index in S, and j be the index in $[n] \setminus S$ closest to i. We will set $f(i) = f(j)$, remove i from S, and then repeat this process until S is empty. Observe that each step of this process maintains that f is π-free over domain $[n] \setminus S$: this initially holds by maximality of S, and at each step we only add an identical element adjacent to an existing element and so cannot introduce a new π. So, by the end f will be π-free. □

When discussing how a copy of a permutation π is laid out in f, it will often be useful to refer to the values $x \in [n]$ corresponding to each element $\pi(1), \pi(2), \ldots$ in the permutation. We'll call these the ***legs*** of a permutation.

One problem that we will face in our algorithms is that we will want to split the problem into sub-problems that correspond to a certain "rectangle" in the input-output grid, but we can't make a rectangle, we can only restrict to vertical slices. However, for some problems, restricting to a vertical column is sufficient. Specifically, previous work has investigated ***erasure resilient*** property testers for various problems (of particular relevance to us is erasure resilient testing of permutation avoidance). We say that a tester is erasure resilient (***ER***) if it can tolerate some fraction of the input being erased. More formally an erasure resilient tester is defined as follows:

Definition 2.2. Let $D \subseteq [n]$, $f : D \to \mathbb{R}$, and $f' : [n] \to \mathbb{R} \cup \perp$, where f' restricted to D equals f', and for $x \notin D, f'(x) = \perp$. We say that an algorithm $\mathcal{A}$ is an ER tester if, given such an f', $\mathcal{A}$ can make some

queries, and distinguish between the cases that f has a property, and that f is ε-far from having a property (where here we define ε-far to mean we must modify at least εn values, as opposed to the other natural definition of $\varepsilon|D|$). The query complexity of $\mathcal{A}$ is the number of queries made to f'.

An ER tester is parameterized not just by the ***distance parameter*** ε, but also by the ***sparsity parameter*** $\alpha = |D|/n$ of the actual function.

One specific powerful result established in previous work is:

Fact 2.3. Fix $\pi \in S_2 \cup S_3$. There is an ER tester for α-sparse π-avoidance with query complexity $O((\varepsilon\alpha)^{-1}\operatorname{polylog}(n))$.

3 Overview of Newman's and Varma's Tester

In this section, we describe Newman's and Varma's $2^{O(\sqrt{\log n})}$-query tester for π-freeness of functions $f : [n] \to \mathbb{R}$, where $\pi \in S_4$ [NV24]. This tester captures most of the ideas behind Newman's and Varma's $n^{o(1)}$-query tester for arbitrary constant-length permutations and is much simpler to describe.

Before giving the proof, we introduce some terminology and give intuition for the proof. Given a function $f : [n] \to \mathbb{R}$ and an error parameter ε, we view f as an $n \times |\mathrm{Im}(f)|$ grid G_0, whose horizontal axis is labeled with values in $[n]$ and vertical axis with values in $\mathrm{Im}(f)$ (in increasing order). The grid G_0 contains the points $(i, f(i))$ for $i \in [n]$. Note that the tester does not have access to G_0; the key first step of the tester will be to build a coarse representation of G_0 called the ***coarse grid***. The coarse grid is defined by a partition of $\mathrm{Im}(f)$ into horizontal ***layers*** $I_1, \ldots, I_m$ and a partition of $[n]$ into vertical ***stripes*** $S_1, \ldots, S_m$. The intersection of a stripe and a layer is called a ***box***. See Figure 1 for a visual definition of these terms.

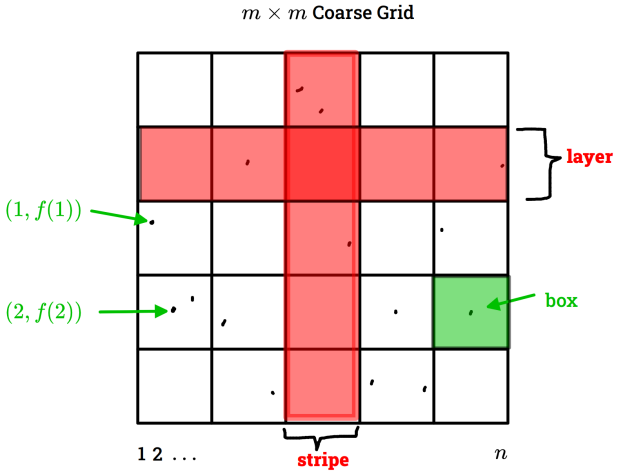

Figure 1: Coarse Grid

The testing algorithm will work by sampling points to form a coarse grid, and then handling each of the following cases (roughly speaking):

1. There is an occurrence of π that is "visible in the coarse grid" — i.e., the non-empty cells of the coarse grid contain an occurrence of π.
2. We can find a box which contains many disjoint π's, and recursively apply our algorithm to the box.
3. It's possible to find a π by finding part of a π within a box, and combining this with some structure between boxes.

We'll now give a more precise description of our algorithm, and an analysis of the algorithm.

Theorem 3.1. Fix $\varepsilon \geq \Omega(1)$ and $\pi = (3, 2, 1, 4)$. There is a $2^{O(\sqrt{\log n})}$-query π-avoidance tester.

Proof Sketch. Let $m = 2^{\sqrt{\log n}}$. Our tester begins by constructing an $m \times m$ coarse grid G as follows:
• Sample $t = \Theta(m \log n)$ values $x \in [n]$.

- For $i \in [m-1]$, let x_i be the $i \cdot t/m$-th largest sampled x value and let y_i be the $i \cdot t/m$-th largest y_i value among the $f(x)$'s for the sampled points.
- As corner cases, define $x_0 = 1, y_0 = -\infty$ and $x_m = n, y_n = \infty$.
- For $i \in [m]$, define the i-th layer I_i as $[x_{i-1}, x_i)$.
- For $i \in [m]$, define the i-th stripe S_i as $[y_{i-1}, y_i)$.

We say that a box is ***vdense*** if it contains more than $\Omega(n/m)$ points of G (we'll specify the constant later). We now will establish that if more than $n\varepsilon/32$ points lie outside of vdense boxes, then we will find (with good probability) a copy of π between the boxes of the coarse grid. To prove this we use a powerful theorem of Marcos and Tardos:

Fact 3.2 (Marcos, Tardos [MT04])**.** There exists a constant C_π such that if A is an $n \times n$ $\{0,1\}$-valued matrix with more than $C_\pi \cdot n$ 1's, then there are a set of 1's in A that form a π pattern.

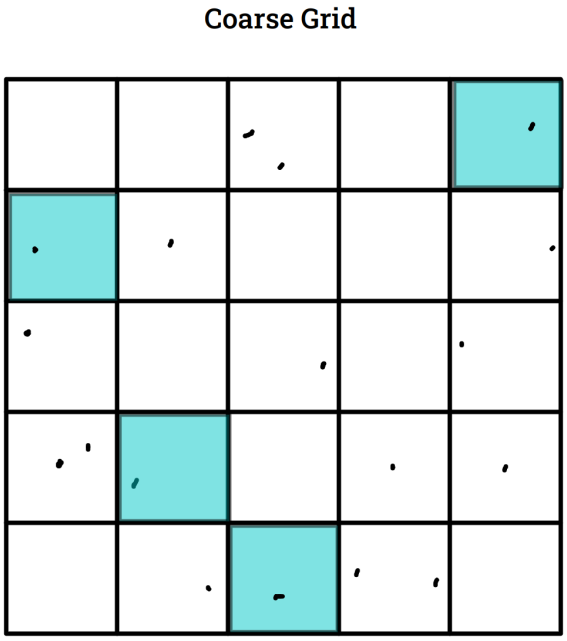

Figure 2: Finding π in the Coarse Grid

Lemma 3.3. Call a box in the coarse grid ***vdense*** if it contains more than $(n/m)\varepsilon/(100C_\pi)$ points of G (where C_π is the constant from Fact 3.2). Let X be the set of points in G that lie in non-vdense boxes. If $|X| \geq n\varepsilon/32$, then with high probability when we sample points to construct the coarse grid we will find a copy of π between the non-empty boxes in the coarse grid.

Proof. We claim that over the course of choosing $t = \Theta(m \log n)$ random points (where we now can specify the constant needed in the expression for t, setting $t = 10000C_\pi\varepsilon^{-1}m \log n$ works), we will obtain points in $1 + C_\pi m$ distinct boxes of the coarse grid, at which point Fact 3.2 allows us to find an occurrence of π in the coarse grid, which implies that our function is not π-avoiding.

To see why we get this many points in distinct boxes, we consider taking the t points incrementally. Suppose we have taken some points, and have hit $x \leq C_\pi m$ distinct non-vdense boxes so far. The number of points in the union of these x non-vdense boxes is at most $x(n/m)\varepsilon/(100C_\pi) \leq n\varepsilon/100$. Thus, there are still at least $n\varepsilon/64$ points in X whose boxes we haven't hit, so sampling another random point yields a new box with probability at least $\varepsilon/64$.

Now standard probability bounds show that our choice of t is large enough that we will hit $C_\pi m+1$ disjoint boxes with very good probability. By the Marcus Tardos theorem Fact 3.2, this gives us an occurrence of π in the coarse grid, and hence a π in the function. □

By Lemma 3.3 we may assume for the rest of the proof that there are at most $n\varepsilon/8$ points of G in non-vdense boxes (as defined in Lemma 3.3). By Fact 2.1 this implies that there are $n\varepsilon/8$ disjoint π's with all legs lying in vdense boxes; call such π's ***good***.

Call a box ***dense*** it it contains at least $1/4$ the number of points required to be vdense.

Lemma 3.4. Using our t sampled points, we can identify a set of boxes $\mathcal{B}$ which includes all vdense boxes, and doesn't include any non-dense boxes.

Proof. A standard Chernoff bound implies that we can get an additive $\Theta(n/m)$ estimate of the number of points in each box (note that n/m is the maximum possible number of points in a box). We set $\mathcal{B}$ to be the set of boxes for which our estimate of the number of points in the box is at least $1/2$ the number of points required to be vdense. Our Chernoff bound plus a union bound implies that this strategy places all vdense boxes in $\mathcal{B}$, and doesn't place any non-dense boxes in $\mathcal{B}$. □

At this point we (logically) ***erase*** all points outside of $\mathcal{B}$, and focus on testing the erased function for π-avoidance; call the erased function f'. Note that if f is π-avoiding, the erased function f' is also π-avoiding, and by Lemma 3.4 combined with Lemma 3.3 if f is ε-far from π-avoiding then f' contains $n\varepsilon/8$ disjoint copies of π. In future discussions we will treat erased points as completely not existing. So, e.g., all non-empty boxes are now dense. This also means that we know all the dense boxes, because these are exactly $\mathcal{B}$.

Now, we show that the dense boxes are nicely spread out.

Lemma 3.5. There are at most $O(1)$ dense box in each stripe and each layer.

Proof. A straightforward Chernoff bound plus union bound implies that every stripe and every layer contains $\Theta(n/m)$ points. But there are $\Omega(n/m)$ points in a single dense box, so there are at most $O(1)$ dense boxes per layer/stripe. □

Now, the obstacle to finding π's using the coarse grid is that a π might have multiple legs in the same stripe or layer, making the coarse grid insufficiently granular to find an instance of π. We want to look for π's by finding part of the structure of the π in the coarse grid and part of it within layers / stripes. See Figure 3 for a visualization of this. To facilitate this, we define a graph on the dense boxes. We say that

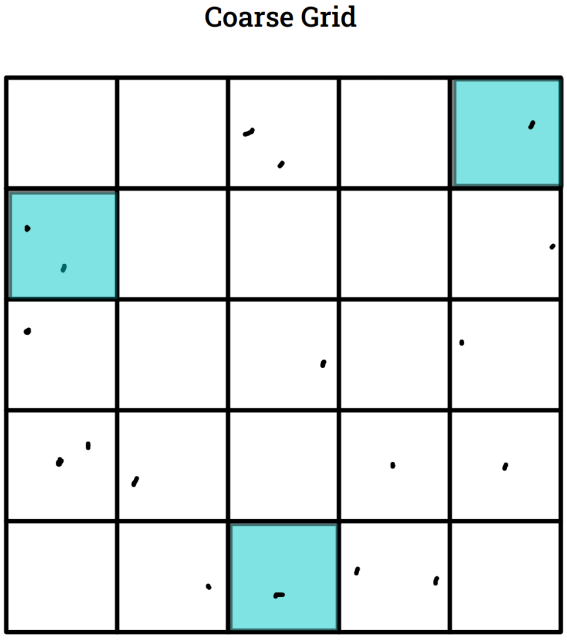

Figure 3: A copy of π with the first two legs in a single box, and the last two legs in different boxes.

two dense boxes are ***directly connected*** if they lie in the same layer or stripe, and ***connected***, if there is a path of direct connections that joins the boxes. For a dense box B, define $N_\ell(B)$ to be the set of dense boxes which are within distance ℓ of B (note that B is distance 0 from B so $B \in N_\ell(B)$ for any $\ell \geq 0$). Now, define M_i to be the set of π's whose legs are split across exactly i connected components of this graph. Note that for some $i \in [4]$, there will be $\Omega(n)$ disjoint copies of π in M_i; our algorithm will exploit this fact by performing queries that would uncover a π in each of the 4 cases. We are now ready to finish the description of our tester:

- Choose a random dense box B and run our algorithm recursively on $N_3(B)$. If the recursive application of our algorithm finds an occurrence of π, report this.
- For each dense box B, test $N_3(B)$ for all patterns of length at most 3. If there is some way to piece together patterns that you find to form a π, report this.
- If no π's were found, report that there are probably no π's in the array.

Description of details that we're glossing over in this proof sketch There are a number of subtle details that we're missing in this high level description, and that we will not handle in our proof sketch. We'll now give a description of the places where we're being imprecise, and give some intuition for how to fix the algorithm.

We've defined our algorithm recursively, but the smaller instances of the problem aren't quite the same as the big instance. In particular, we'll assume that the smaller instances receive the original n as an input in addition to their domain size n'. The "recursive calls" of the algorithm will actually use $m = 2^{\sqrt{\log n}}$ still, as opposed to $2^{\sqrt{\log n'}}$. We'll terminate the recursion once $n' < 2^{\sqrt{\log n}}$, and handle this base case by querying all n' points in the domain of the recursive call. Note that this strategy ensures that the number of levels of recursion in the algorithm is $O(\sqrt{\log n})$.

Another detail that we're not handling is that we'd like to perform a union bound over the levels of recursion to say that all of our assertions "this holds with good probability" simultaneously hold across all levels of recursion with good probability. This is not a big deal because there are only $O(\sqrt{\log n})$ levels of recursion. However, this necessitates repeating any steps that only succeed with constant probability a sufficient number of times so that we can perform this union bound; we won't spell this out because this is a proof sketch.

Another detail that we're glossing over in the above description of the algorithm is that we can't actually restrict to a box and do queries on that box. Fortunately, because the boxes in question are all dense, and because each stripe has at most $O(1)$ dense boxes, we can treat points that lie in the stripe's x-range but outside of the box in question as "erasures" and use erasure resilient testers as described in Fact 2.3. Because the recursive call to our algorithm will also need to handle erasures, we'd need to handle those as well in our algorithm.

There's an additional subtlety in the part of the algorithm where we "test all dense boxes for avoiding smaller permutations". Namely, in addition to testing for occurrences of a pattern we also need to test for occurrences of these smaller patterns with certain ***restrictions***. Specifically, the extra restriction is that we might require legs of the permutation to lie in certain parts of the grid. This is important to ensure that we can combine the partial pattern with things from other dense cells to form an occurrence of π. We'll discuss this detail a bit more when arguing for the correctness of the algorithm, but refer the reader to [NV24] for the full details. An illustration of this last issue is provided in Figure 4.

Query Complexity

Claim 3.6. The query complexity is $2^{O(\sqrt{\log n})}$.

Proof. Let $T(n, \varepsilon, \alpha)$ be the running time of our tester with domain size n, error parameter ε, and erasure parameter α. We have the following recurrence:

$$T(n, \varepsilon, \alpha) = \widetilde{\mathcal{O}}(m) + O(T(n/m, \Omega(\varepsilon), \Omega(\alpha))).$$

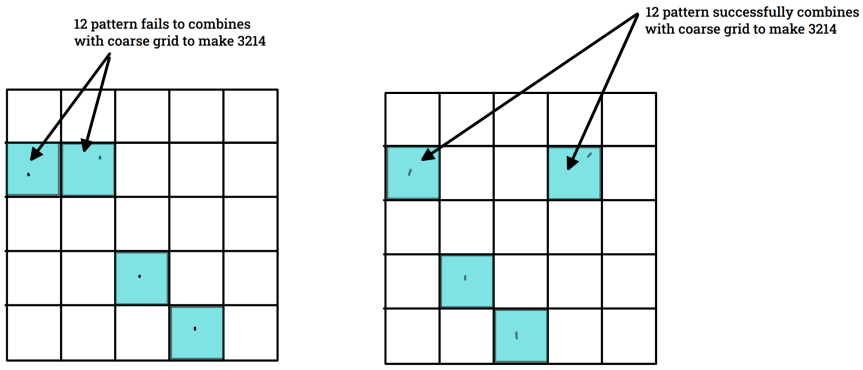

Figure 4: Illustration of sub-pattern successfully and unsuccessfully combining with the coarse grid to form a 3214-occurrence.

Setting $m = 2^{\sqrt{\log n}}$ makes the recursion tree depth $O(\sqrt{\log n})$. Thus, our query complexity is at most

$$2^{\sqrt{\log n}} \cdot O(1)^{\sqrt{\log n}} \leq 2^{O(\sqrt{\log n})}.$$

□

Correctness To conclude we must show that if there are $n\varepsilon/8$ disjoint π's, then this procedure will find one with good probability. We break into cases based on which of the M_i's is large (this is for analysis purposes, the tester doesn't need to know which case it's in).

Case 1: $|M_4| \geq \Omega(n)$. Then the pattern appears in the coarse grid, and we will already have found it.

Case 2: $|M_1| \geq \Omega(n)$. Then there must be $\Omega(m)$ dense boxes with $\Omega(n/m)$ disjoint π's in each of them. If not, then the number of M_1-type π's would be at most $O(m)o(n/m) + o(m)O(n/m) = o(n)$, because there are only $O(m)$ dense boxes and each has at most $O(n/m)$ π's. Thus, sampling a random box B and recursing on $N_3(B)$ will find a π with good probability.

Case 3: $|M_3| \geq \Omega(n)$. A type M_3 π has 2 legs in $N_3(B)$ for some B, and then has the other 2 legs in different connected components. Say that π ***double hits*** a neighborhood $N_3(B)$ if π has two legs in this neighborhood. We claim that $\Omega(m)$ many dense boxes are double hit by $\Omega(n/m)$ many π's. Indeed, if this were not the case, then the number of disjoint M_3 π's would be at most $o(m)O(n/m)+O(m)o(n/m) = o(n)$. Thus, there is some box B (in fact many, but we only need one) such that B is far from avoiding the 2 leg pattern that would contribute to π, and a π in B can combine with elements from two other coarse boxes to form an instance of a π. Thus, our procedure of testing all the dense boxes for each pattern of length 2 will find that box B has this pattern, and realize that this can be combined with boxes in the coarse grid to make the entire pattern.

Actually, we are glossing over an important detail here: when testing $N_3(B)$ for smaller patterns, we actually aren't quite just checking for patterns, we are checking for patterns where the legs are restricted to lie in specific places. This is important to make sure that we can actually combine the pattern with the dense boxes.

Case 4: $|M_2| \geq \Omega(n)$. There are actually two quite distinct types of M_2 π's.

Case 4.1: there are $\Omega(n)$ many π's with 3 legs in one connected component and 1 leg in a different connected component . We handle this case using exactly the same technique as in Case 3.

Namely, we assert that there must be some box B such that $N_3(B)$ contains $\Omega(n/m)$ many copies of the length 3 sub-permutation that we need, and so our tester will find some such copy and we will be able to combine it with a box from the coarse grid to complete the pattern.

Case 4.2: There are $\Omega(n)$ many π's with 2 legs in one connected component, and the other two legs in their own different connected component . This is the most challenging case. We split into cases based on which of the legs are grouped together in the same component. For simplicity, let's assume that the first two legs are grouped together, and so are the second. Then we can write $\pi = \phi\psi$ where ϕ, ψ are the first and second two legs of the permutation respectively. Call a box B ϕ-good if $\Omega(n/m)$ π's have their first two legs (i.e., the ϕ part of π) in B. Similarly, say a box is ψ-good if the last two legs are in B.

We'll now show that there are $\Omega(n)$ many disjoint π's that start in a ϕ-good box. Indeed, if this were not the case, then the total number of π's would be at most $o(n) + o(n/m)O(m) = o(n)$, because there are at most $O(m)$ dense boxes and each non-ϕ-good box contributes at most $o(n/m)$ copies of π. Hence, there are actually $\Omega(n)$ many π's that start in a ϕ-good box; call these ***awesome* π*'s***. By averaging there is a box B such that $\Omega(n/m)$ of these awesome π's end in B. This makes B a ψ-good box, and by construction there's a ϕ-good box which can be paired with B' to get a π. Thus, our procedure of testing all the boxes for small patterns will result in finding two size two patterns that can be combined to our size 4 π pattern. □

4 Testers in the Bounded Range Setting

We now give our simple novel testers for the bounded range setting. We'll start with a very specific example, and then discuss what can be done in general.

Theorem 1.1. There is a 3412-avoidance tester for $f : [n] \to [R]$ with query complexity

$$O\Big(\varepsilon^{-1} R \log n \left(\log^3 R + \log^2 R \log \varepsilon^{-1}\right) \log\log n\Big).$$

Proof. Our algorithm will only reject if it finds an occurrence of 3412 in f — so, it will always accept when f is indeed 3412-avoiding. For the remainder of the proof we'll assume that f is ε-far from 3412-avoiding, and show that the algorithm is likely to find a copy of 3412. The basic idea will be to guess a division into 4 "quadrants", as depicted Figure 5, and then search for an increasing pair in both the top left and the bottom right quadrant.

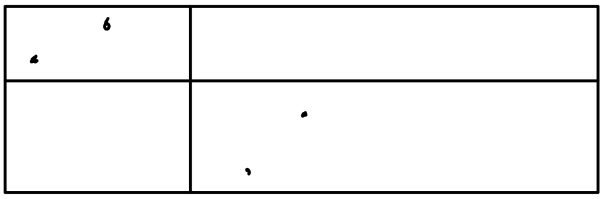

Figure 5: A division of the grid into 4 quadrants, with increasing pairs in both the top left and bottom right quadrant together forming a copy of the pattern 3412.

That is, we'll choose thresholds $x^* \in [n]$, $y^* \in [R]$, and search for pairs $x_1 < x_2$ with $f(x_1) < f(x_2)$ and either $x_2 \le x^*$, $f(x_1) \ge y^*$ (i.e. both points are in the top left quadrant) or $x_1 > x^*$, $f(x_2) < y^*$ (i.e. both points are in the bottom right quadrant).

In order to search for increasing pairs in a particular quadrant, we'll make use of Dixit, Raskhodnikova, Thakurta and Varma's $O(\varepsilon^{-1} \log n)$-query erasure resilient monotonicity tester [Dix+18], which we'll refer to as MonoTester. The reason we need an erasure-resilience property is that, to test the upper left quadrant, we'll want to run a tester on the first x^* elements of the domain, but to consider the output of f as "erased"

whenever $f(x) < y^*$. (Similarly for the bottom right quadrant.)

Our overall algorithm will be as follows:

Algorithm 1 3412-avoidance tester

for $i \in \{1, \dots, \lceil \log R \rceil\}$ **do**
  for 2^i iterations **do**
    $y^* \leftarrow$ random element of $[R]$ ▷ choose a y threshold randomly
    $x^* \leftarrow \lceil \frac{n}{2} \rceil$ ▷ do a binary search for the x threshold
    binSearchJump $\leftarrow \lceil \frac{n}{4} \rceil$
    while binSearchJump $> \lceil \frac{\varepsilon n}{8R\lceil \log R \rceil} \rceil$ **do**
      run MonoTester on the upper left quadrant $\log \log n$ times with distance parameter $2^i \cdot \frac{\varepsilon}{8R\lceil \log R \rceil}$
      if an increasing pair is found **then**
        $x^* \leftarrow x^* -$ binSearchJump
      else
        $x^* \leftarrow x^* +$ binSearchJump
      binSearchJump $\leftarrow \frac{\text{binSearchJump}}{2}$
    $x^* \leftarrow x^* + \lceil \frac{\varepsilon n}{8R\lceil \log R \rceil} \rceil$
    run MonoTester with parameter $2^i \cdot \frac{\varepsilon}{9R\lceil \log R \rceil}$ on the bottom right quadrant
output "not 3412-free" if the algorithm has queried 4 points forming a 3412; output "3412-free" otherwise

To show that this algorithm works, we first state the following simple lemma:

Lemma 4.1. For some $i \in [\lceil \log R \rceil]$, there are at least $R \cdot 2^{-i}$ many heights y with at least $2^i \cdot \frac{n\varepsilon}{4R\lceil \log R \rceil}$ disjoint 3412's whose first leg is at height y.

Proof. For each $i \in [\lceil \log R \rceil]$, let x_i be the number of heights y such that the number of 3412's starting at height h lies in $\left[2^{i-1} \cdot \frac{n\varepsilon}{4R\lceil \log R \rceil}, 2^i \cdot \frac{n\varepsilon}{4R\lceil \log R \rceil}\right)$ (except, if $i = \log R$, then include the right endpoint). Then, because there are $n\varepsilon/4$ disjoint copies of 3412 in f by Fact 2.1, we have:

$$\frac{n\varepsilon}{4} \leq \sum_{i=1}^{\lceil \log R \rceil} 2^i \cdot \frac{n\varepsilon}{4R\lceil \log R \rceil} \cdot x_i.$$

Thus, there exists $i \in [\lceil \log R \rceil]$ such that $x_i \geq R \cdot 2^{-i}$ □

Since for each $i \in [\lceil \log R \rceil]$ we run 2^i iterations of our procedure, Lemma 4.1 ensures that, for some i, with constant probability one of our iterations with that i will choose y^* such that at least $2^i \cdot \frac{\varepsilon n}{4R\lceil \log R \rceil}$ disjoint 3412's have first leg at height y^*. Let us now restrict our attention to such an iteration. Our next lemma will help us show that our binary search procedure over x^* lets us find a 3412 with constant probability.

Lemma 4.2. For such a y^*, there exists an x' such that there are at least $2^i \cdot \frac{\varepsilon n}{8R\lceil \log R \rceil}$ disjoint increasing pairs in each of the top left and bottom right quadrants.

Proof. We know that at least $2^i \cdot \frac{\varepsilon n}{4R\lceil \log R \rceil}$ disjoint 3412's have first leg at height y^*. Let x' be median x-coordinates of all the second legs of those 3412; we will use this x' to define our quadrants. Observe that all those 3412s whose second leg lies before x' contribute an increasing pair to the upper left quadrant, and that all those 3412s whose third leg lies after x' contribute an increasing pair to the bottom right quadrant. So, our choice guarantees at least $\frac{1}{2} \cdot 2^i \cdot \frac{\varepsilon n}{4R\lceil \log R \rceil}$ increasing pairs in each of these quadrants. □

We now claim that, with at least constant probability, the final value of x^* that our binary search ends up on is at most $x' + \lceil \frac{\varepsilon n}{100R\lceil \log R \rceil} \rceil$. Suppose this were not the case: then, at some $x \geq x'$, the binary search must have chosen to step to the right, meaning that all $\log \log n$ runs of MonoTester must have failed to find an increasing pair in the upper left quadrant. But note that at such an x, since there are at least

$2^i \cdot \frac{\varepsilon n}{8R\lceil \log R\rceil}$ disjoint increasing pairs in the upper left quadrant, the distance from monotone in the upper left quadrant actually is at least $2^i \cdot \frac{\varepsilon}{8R\lceil \log R\rceil}$, so each run of MonoTester will find an increasing pair with constant probability. The chance of all of these runs failing is at most $\frac{1}{2^{\log\log n}} = \frac{1}{\log n}$. So, since our binary search takes $\log n$ steps, with at least constant probability such a failure never happens.

Observe that, letting x^* be the final value of the x threshold, in the process of the binary search we must have found an increasing pair in the upper right quadrant for x^*. This is because the last time the binary search steps left, it must have been because an increasing pair was found in the upper left quadrant at that stage, and x^* will end up greater than or equal to the value of the x threshold at the last step left. So it suffices to show that we have at least constant probability of finding an increasing pair in the bottom right quadrant when we run MonoTester on it. But this follows because $x^* \le x' + \lceil \frac{\varepsilon n}{8R\lceil \log R\rceil}\rceil$, so by Lemma 4.2 there are at least $2^i \cdot \frac{\varepsilon n}{8\mathbb{R}ceil\log R} - \lceil \frac{\varepsilon n}{100R\lceil \log R\rceil}\rceil \ge 2^i \cdot \frac{\varepsilon n}{9R\lceil \log R\rceil}$ disjoint increasing pairs in the bottom right quadrant, so the distance to monotone is at least $2^i \cdot \frac{\varepsilon}{9R\lceil \log R\rceil}$.

Thus, if f is ε-far from 3412-free, this algorithm will find a 3412 with constant probability. Repeating the algorithm a constant number of times will increase this probability to at least 99%. To calculate query complexity, recall that MonoTester requires $O(\delta^{-1}\log n)$ queries when run with distance parameter δ. For a given value of i, we perform 2^i iterations of our binary search. Each iteration of binary search requires order

$$\log\left(\frac{n}{\lceil \frac{\varepsilon n}{8R\lceil \log R\rceil}\rceil}\right)\cdot \log\log n \le O\left(\log\left(\varepsilon^{-1}R\right)\log\log n\right)$$

runs of MonoTester at distance parameter $2^i \cdot \frac{\varepsilon}{8R\lceil \log R\rceil}$, so in total every given value of i requires

$$2^i \cdot O\left(\log\left(\varepsilon^{-1}R\right)\log\log n\right)\cdot O\left(\frac{\log n}{2^i \cdot \frac{\varepsilon}{8R\lceil \log R\rceil}}\right) \le O\Big(\varepsilon^{-1}R\left(\log^2 R + \log R\log\varepsilon^{-1}\right)\log n\log\log n\Big)$$

queries. Since we run this for $\lceil \log R\rceil$ values of i, the total query complexity is therefore

$$O\Big(\varepsilon^{-1}R\left(\log^3 R + \log^2 R\log\varepsilon^{-1}\right)\log n\log\log n\Big).$$

□

4.1 More General Bounded Range Algorithms

In the previous algorithm, we exploited some particular structure of 3412. Here, we'll give a very simple algorithm showing that one can still get a runtime polynomial in R for any permutation π of a constant length k.

Theorem 4.3. If π is a permutation of length k, there is a non-adaptive $O(\varepsilon^{-1}R^k)$-query tester for range-R π-avoidance.

Proof. The algorithm will simply sample $100\varepsilon^{-1}k^3R^k$ random inputs, and claim the permutation is π-free if f does not have a π using only those inputs. To show that this algorithm succeeds, note that if f is ε-far from π-free, by Fact 2.1 f contains $\frac{n\varepsilon}{k}$ disjoint copies of π. So, by the pigeonhole principle, there is some tuple $(y_1,\dots,y_k)\in[R]^k$ such that f contains $\frac{n\varepsilon}{kR^k}$ disjoint copies of π whose ith leg has height y_i for all i.

Let S_1 consist of the smallest $\left(\frac{1}{k}\cdot\frac{n\varepsilon}{kR^k}\right)$ many indices of first legs among those copies of π. Similarly, for $i\in\{2,\dots,k\}$, we'll let S_i consist of the $\left(\frac{i-1}{k}\cdot\frac{n\varepsilon}{kR^k}\right)$th through $\left(\frac{i}{k}\cdot\frac{n\varepsilon}{kR^k}\right)$th smallest indices of ith legs among those copies of π. Observe that any index in S_i is smaller than any index in S_{i+1}, and that every $x\in S_i$ has $f(x)=y_i$. So, if our random samples find at least one element from each of these S_i, then we will have found a copy of π. Since each S_i has size $\frac{n\varepsilon}{k^2R^k}$, a given sample will hit it with probability $\frac{\varepsilon}{k^2R^k}$. Since we're making $100\varepsilon^{-1}k^3R^k$ random samples, with high probability we will find an element from each of them. □

5 Conclusion

In this note we have given an exposition of Newman and Varma's sublinear permutation avoidance tester. We have also studied permutation avoidance in the bounded range setting, and obtained extremely simple testers with polynomial dependence on the range. We leave improving our results in the bounded range setting, or showing a lower bound, as open problem.

References

[BC18] Omri Ben-Eliezer and Clément L Canonne. "Improved bounds for testing forbidden order patterns". In: *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM. 2018, pp. 2093–2112.

[Ben19] Omri Ben-Eliezer. "Testing Local Properties of Arrays". In: *10th Innovations in Theoretical Computer Science* (2019).

[BLR90] Manuel Blum, Michael Luby, and Ronitt Rubinfeld. "Self-testing/correcting with applications to numerical problems". In: *Proceedings of the twenty-second annual ACM symposium on Theory of computing*. 1990, pp. 73–83.

[BLW19] Omri Ben-Eliezer, Shoham Letzter, and Erik Waingarten. "Optimal adaptive detection of monotone patterns". In: *arXiv preprint arXiv:1911.01169* (2019).

[Dix+18] Kashyap Dixit et al. "Erasure-Resilient Property Testing". In: *SIAM Journal on Computing* 47.2 (2018), pp. 295–329. DOI: 10.1137/16M1075661. eprint: https://doi.org/10.1137/16M1075661. URL: https://doi.org/10.1137/16M1075661.

[Fis04] Eldar Fischer. "On the strength of comparisons in property testing". In: *Information and Computation* 189.1 (2004), pp. 107–116.

[MT04] Adam Marcus and Gábor Tardos. "Excluded permutation matrices and the Stanley–Wilf conjecture". In: *Journal of Combinatorial Theory, Series A* 107.1 (2004), pp. 153–160.

[New+19] Ilan Newman et al. "Testing for forbidden order patterns in an array". In: *Random Structures & Algorithms* 55.2 (2019), pp. 402–426.

[NV24] Ilan Newman and Nithin Varma. "Strongly sublinear algorithms for testing pattern freeness". In: *TheoretiCS* 3 (2024).

[Ras14] Sofya Raskhodnikova. "Testing if an array is sorted". In: *Encyclopedia of Algorithms*. Ed. by Ming-Yang Kao. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 1–5.