Testing Properties of Sparse Images

Dana Ron*
School of EE
Tel-Aviv University
Ramat Aviv, Israel
danar@eng.tau.ac.il

Gilad Tsur
School of EE
Tel-Aviv University
Ramat Aviv, Israel
giladt@post.tau.ac.il

Abstract

We initiate the study of testing properties of images that correspond to *sparse* 0/1-valued matrices of size $n \times n$. Our study is related to but different from the study initiated by Raskhodnikova (*Proceedings of RANDOM, 2003*), where the images correspond to *dense* 0/1-valued matrices. Specifically, while distance between images in the model studied by Raskhodnikova is the fraction of entries on which the images differ taken with respect to all n^2 entries, the distance measure in our model is defined by the fraction of such entries taken with respect to the actual number of 1's in the matrix. We study several natural properties: connectivity, convexity, monotonicity, and being a line. In all cases we give testing algorithms with sublinear complexity, and in some of the cases we also provide corresponding lower bounds.

*This work was supported by the Israel Science Foundation (grant number 246/08).

1 Introduction

Suppose we are given access to an image that is defined by a 0/1-valued $n \times n$ matrix M, and would like to know whether it has a particular property (e.g., the image it contains corresponds to a convex shape). We may read all pixels (bits) in the matrix and run an appropriate algorithm on this data, thus obtaining an exact answer in at least linear time. However, assume we are interested in a much more efficient algorithm, and are willing to sacrifice some precision. Namely, we seek a randomized, sublinear-time algorithm that can distinguish with high success probability between a matrix that has the specified property, and a matrix that is relatively far from having the property. In other words, we seek a *property testing* algorithm [RS96, GGR98].

In order to formalize the above question, we first need to define what it means to be *far* from having the property, and what access we have to the matrix. One natural definition of distance between matrices is the Hamming weight of their symmetric difference, normalized by the size of the matrices, which is n^2, and the most straightforward form of accessing the matrix is probing its entries. Indeed, this model of testing properties of images was introduced and studied by Raskhodnikova [Ras03], and we later discuss in more details the results that she obtained as well as their relation to our results.

The abovementioned model is most suitable for relatively dense images, that is, images in which the number of 1-pixels (i.e., entries (i, j) for which $M[i, j] = 1$) is $\Omega(n^2)$. However, if the image is relatively sparse, e.g., the number of 1-pixels is $O(n)$, then a natural alternative is to normalize the distance with respect to the Hamming weight of the matrix, which we denote by $w(M)$, rather than to normalize by n^2. We believe that this type of measurement is appropriate in many contexts. For instance, an image of a single line of constant width is not generally viewed as very similar to an empty image, while it is considered so in the dense-images model. Essentially, while the dense-images model is suitable for testing images composed of areas, the sparse-images model works just as well with images composed of lines (or outlines).

An additional difference between the dense-images model and the sparse-images model is that in the latter model we also give the algorithm access to uniformly selected 1-pixels (in addition to query access to entries of its choice). Observe that if an image is sparse, then it actually makes sense to store it in a data-structure of size $O(w(M) \log n)$ rather than in an $n \times n$ matrix. Such space-efficient data structures may be easily devised to support uniform sampling of 1-pixels as well as answering queries to particular entries of M (possibly with an overhead of $O(\log w(M))$). In the dense image model (as in many property testing scenarios) the algorithm complexity is measured in terms of the number of queries it performs, where a query is checking whether the value of a pixel is 0 or 1. As we also allow our algorithms access to uniformly selected 1-pixels, we will wish to know how many of these were sampled, as well. Thus we consider two measures of complexity – the number of locations in the image that the algorithm queries, which we call *query complexity*, and the number of uniformly selected 1-pixels that the algorithm requests, which we call *sample complexity*.

We note that the relation between the dense-images model and the sparse-images model is reminiscent of the relation between the dense-graphs model [GGR98] and the sparse-graphs model [PR02, KKR04] (which extends the bounded-degree model [GR02]). We return to this relation subsequently, but first we state our results and the techniques we apply.

1.1 Our results

In what follows, when we use the term "complexity" we mean the sample and query complexity of the algorithm. For all the properties we study, except for line imprints, the running time of the algorithm is at most a logarithmic factor larger, and for line imprints it is polynomial in the number of queries (which is independent of n). The parameter ϵ is the *distance* parameter. Namely, the algorithm should accept with high constant probability[1] every matrix that has the property and should reject with high constant probability every matrix M that is ϵ-far from having the property (that is, more than $\epsilon \cdot w(M)$ pixels in M should be modified so that it obtains the property). We study the following properties.

- **Connectivity.** We say that an image M is *connected* if the underlying graph induced by the neighborhood relation between 1-pixels is connected. We give a testing algorithm for connectivity whose complexity is $\tilde{O}\left(\min\left\{w(M)^{1/2}, n^2/w(M)\right\} \cdot \epsilon^{-2}\right)$. Thus, as long as $w(M) \leq n^{4/3}$, the complexity of the algorithm increases like the square-root of $w(M)$, and once $w(M) > n^{4/3}$ it starts decreasing as $w(M)$ increases. We also prove a lower bound of $\Omega\left(\min\left\{w(M)^{1/3}, n^2/w(M)\right\}\right)$ for $\epsilon = \Theta(1)$. For one-sided error algorithms (that is, algorithms which accept an image with probability 1 if the image has the property) we show a simple lower bound of $\Omega(\min\{w(M), n^2/w(M)\})$ (which is $\Omega(w(M))$ for $w(M) = O(n)$).

- **A line (imprint).** We say that an image is a *line imprint* (or simply a *line*) if there exists a line such that all the pixels that the line intersects are 1-pixels, and there are no other 1-pixels in the image. We give a (one-sided error) algorithm for testing this property whose complexity is $O(\log(1/\epsilon)/\epsilon)$. This algorithm and its analysis are presented with a more general result concerning testing sparse images that have a small VC-dimension (and a corresponding result about learning when the distance measure is defined with respect to $w(M)$ rather than the size of the domain, which is n^2). While this result is fairly simple, it does not follow from the known transformation of (proper) learning results to testing results [GGR98].

- **Convexity.** We say that an image M is *convex* if there exists a convex shape that is connected, closed and such that all the pixels that the shape intersects in M are 1-pixels, and there are no other 1-pixels in the image.[2] We assume without loss of generality that the convex shape is a polygon, and we consider a certain variant of this property where we require the gradient of the lines defining the convex shape to be of the form $1/r$ for an integer r.[3] For this property we give an algorithm whose complexity is $\tilde{O}(w(M)^{1/4} \cdot \epsilon^{-2})$.

- **Monotonicity.** We say that an image M is *monotone* if for every two 1-pixels (i_1, j_1) and (i_2, j_2), if $i_1 < i_2$, then $j_1 \leq j_2$. We give a one-sided error algorithm for testing monotonicity whose complexity is $\tilde{O}\left((n^{2/3}/w(M)^{1/3})\epsilon^{-2}\right)$. This algorithm improves on a simple sampling algorithm whose complexity is $O((w(M)/\epsilon)^{1/2})$, whenever $w(M) = \Omega(n^{4/5})$. (This simple algorithm only takes uniform samples and rejects if and only if it detects a violation of monotonicity.) For example, when $w(M) = \Theta(n)$, the dependence on n is reduced from $n^{1/2}$ to $n^{1/3}$.

[1] Whenever we refer to an event that occurs "with high constant probability", we mean with probability at least $1 - \delta$ for any small constant δ of our choice.

[2] We assume that the shape is contained within the image area.

[3] The requirement for a gradient of the form $1/r$ is imposed to avoid a host of issues relating to the irregular shape of pixelated lines in polygons with other gradients. It is possible that our results can be extended to such cases almost unchanged or that the difference is of essence.

We also give an almost matching lower bound. Namely, we show that any (two-sided error) testing algorithm for monotonicity must have complexity $\Omega(\min\{w(M)^{1/2}, n^{2/3}/w(M)^{1/3}\})$ (for constant ϵ).

For illustrations of the different properties, see Figure 1. Our algorithms (with the exception of the line imprint algorithm) are assumed to be given a constant factor estimate of $w(M)$. The lower bounds hold when the algorithm has such knowledge as well. In the case of connectivity and convexity we show how such an estimate can be obtained without increasing the complexity of the algorithm.

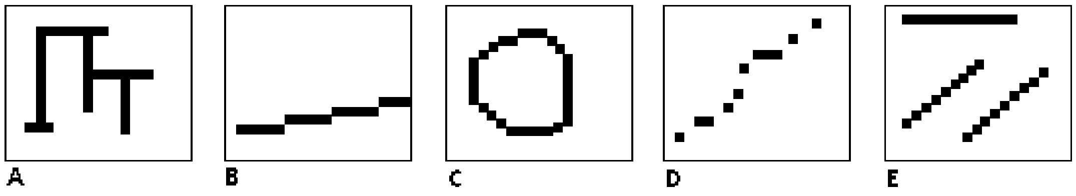

Figure 1: Examples of the tested properties in images: (A) is a connected image, (B) is a line imprint, (C) is convex and (D) is monotone. The image (E) is relatively far from having any of the properties.

1.2 Techniques

As noted in the previous subsection, one of our results, concerning testing the basic property of being a line, is part of a more general technique that exploits the small VC-dimension of the property. While using bounds on the VC-dimension is far from being new, there is a small "twist" in our application. The other results differ from this one, and though each has its own particularities, they can be viewed as sharing a common "theme".

This common theme is that the image is considered in two "scales": a coarser one and a finer one. The coarser scale is determined by uniform samples of 1-pixels, and the finer scale by queries. For example, in the case of testing connectivity, the algorithm considers a partition of the input matrix into submatrices (of size roughly $\sqrt{w(M)}$ in each dimension). Given a sample of 1-pixels, the algorithm first checks whether the submatrices that contain samples are connected. If they are not connected, then the algorithm rejects.[4] Otherwise these submatrices are considered the *backbone* of the image and the algorithm now tests (with the use of queries), whether all but a small fraction of the submatrices in the backbone are "internally connected", and that all but a small fraction of the points are "well connected" to them.

In the case of monotonicity there is no predetermined partition, but rather the sample determines such a partition (or causes rejection since a violation of monotonicity is observed). The partition is such that if the image is far from being monotone, then (almost all) the violations are within the submatrices defined by the sample. We then show that by performing queries within these submatrices (with an appropriate distribution over the queries), we will detect a violation

[4]Indeed, this causes the algorithm to have two-sided error. As noted previously, if we require that the algorithm have one-sided error, then there is no sublinear algorithm when the matrix is relatively sparse.)

with high probability. The convexity testing algorithm also does not have a predetermined partition, but its use of queries is more similar to the connectivity testing algorithm (though it is able to exploit certain useful features of convex images, and is hence more efficient).

1.3 The work of Raskhodnikova [Ras03] and its relation to our work

Raskhodnikova studies three properties in the dense-images model: *connectivity*, *convexity*, and *being a half-plane*. All the algorithms in [Ras03] have complexity that is at most quadratic in $1/\epsilon$, and have no dependence on the size (n^2) of the matrix.

We also consider the (same) property of connectivity and give two algorithms. The first is more efficient when the matrix is below a certain threshold of the density (i.e., $w(M) \leq n^{4/3}$) and the second is more efficient when the density goes above this threshold. The second algorithm is essentially the same as the connectivity testing algorithm in [Ras03] (with the appropriate setting of certain parameters), and its analysis is similar (with certain subtleties). This algorithm is also similar to the connectivity testing algorithm in bounded-degree graphs [GR02] (though the analysis is naturally different).

We also have an algorithm for testing convexity, however, the notion we study of convexity (appropriate for sparse images where $w(M) = O(n)$) and the one studied in [Ras03] (which considers the convex hull of 1-pixels and hence is appropriate for dense images) are different, and so the results are incomparable.

Finally, our testing algorithm for a line imprint can be seen as the "sparse analog" of being a half-plane. As noted by Raskhodnikova [Ras03], it is possible to test whether an image corresponds to a half-plane by attempting to learn the half-plane. She suggests a direct approach that is more efficient (the complexity is $O(1/\epsilon)$ rather than $O(\log(1/\epsilon)/\epsilon)$). For the line imprint we do take what can be seen as a "learning-based" approach (for an appropriate notion of learning), and it is possible that in our case an improvement is possible as well by a direct approach.

1.4 The relation to models for testing graph properties

In the adjacency-matrix (dense-graphs) model [GGR98], the distance between two n-vertex graphs is the fraction of entries on which their adjacency matrices differ (where the fraction is with respect to n^2). In this model the algorithm is allowed to probe the matrix (that is, query whether there is an edge between any pair of vertices of its choice). In the sparse/general graphs model [PR02, KKR04], distance is measured with respect to the number of edges, m, in the graph (or a given upper bound on this number). The algorithm may query any vertex of its choice on its i^{th} neighbor (for any i), and it may also query whether there is an edge between any two vertices (the latter is useful when the graph is sufficiently dense).

Thus there is a similarity in the way the sparse/general graphs model relates to the dense-graphs model and the way the sparse-images model relates to the dense-images model. We also note that for both types of objects (graphs and images) while some properties are meaningful only in one model (dense or sparse), there are properties that are of interest in both models. For example, in the case of graphs, the property of *bipartiteness* (studied in [GGR98, AK02, GR02, KKR04]) exhibits an interesting behavior when considering the whole spectrum of graph densities. In particular, as long as the number of edges, m in the graph is below $n^{3/2}$, the complexity grows like $n^{1/2}$, and once the edge-density increases, the complexity behaves like n^2/m (and there is an almost tight corresponding lower bound). In the case of images, the property of connectivity exhibits a somewhat similar behavior (as discussed previously).

1.5 Other related work

In addition to the work of Raskhodnikova [Ras03] (which has been discussed above), Kleiner et al. [KKN10] study testing partitioning properties of dense images, and there have been quite a few works on testing geometric properties, and in particular convexity, in various models [EKK$^+$00, RV04, CLM06, CSZ00, CS01]. The different definitions in different models lead to a variety of results. The property of monotonicity has been studied quite extensively in the context of functions [EKK$^+$00, BRW05, GGL$^+$00, DGL$^+$99, AC06, HK03, FLN$^+$02, Fis04, HK04, BGJ$^+$09, MBCGS10]. Monotonicity in functions is not a direct equivalent of monotonicity in sparse images, just as connectivity in graphs is not the direct equivalent of connectivity in images - the queries we are allowed differ substantially, and the way local changes effecct the distance from the property differ as well.

1.6 Open problems

Our work suggest several open problems. First, for the property of convexity we have no lower bound, and hence a natural question is whether there is a more efficient algorithm (or a matching lower bound). It is also interesting to study the variant of the property that is defined by imprints of general lines (that is, when the gradient is not necessarily $1/r$). For connectivity our upper and lower bound are (almost) tight for $w(M) > n^{3/2}$ and the question is what is the exact complexity of the problem when $w(M) \leq n^{3/2}$. One might also want to consider a variant of the property of monotonicity in which the image is also required to be connected. It is not hard to verify that by combining a testing algorithm for connectivity and a testing algorithm for monotonicity, we get a testing algorithm for the combined property (this is not necessarily the case in general for disjunctions of properties). The question is whether there is a more efficient algorithm for the combined property. Finally, one may of course consider other natural properties of sparse images that were not studied in this work.

2 Preliminaries

For an integer k let $[k] = \{1, \ldots, k\}$. For a $\{0, 1\}$ matrix M of size $n \times n$, let $w(M)$ denote its *Hamming weight* (number of 1's). For a fixed property $\mathcal{P}$ of $\{0, 1\}$-valued matrices, we say that M is *ϵ-far* from having $\mathcal{P}$, if more than $\epsilon \cdot w(M)$ of the entries of $w(M)$ must be modified so as to obtain the property.

A *testing algorithm* for $\mathcal{P}$ may query the value of $M[i][j]$ for any $1 \leq i, j, \leq n$ of its choice, and may also obtain uniformly selected entries (i, j) such that $M[i][j] = 1$ (which we'll refer to as *samples of* 1*-pixels*). If the matrix M has the property $\mathcal{P}$, then the algorithm should accept with probability at least $2/3$, and if M is ϵ-far from having $\mathcal{P}$, then the algorithm should reject with probability at least $2/3$. If the algorithm accepts every M that has property $\mathcal{P}$ with probability 1, then we say that it is a *one-sided error* algorithm (otherwise it is a *two-sided error* algorithm).

In what follows, when we refer to an event that occurs "with high constant probability", we mean with probability at least $1 - \delta$ for any small constant δ of our choice. For the sake of simplicity, unless it affects the analysis (by more than introducing constant factors in the complexity), we ignore floors and ceilings.

3 Testing Connectivity

For a $\{0,1\}$ matrix M, consider the underlying undirected graph $G(M) = (V(M), E(M))$, where $V(M) = \{(i,j) : M[i,j] = 1\}$ (so that $|V(M)| = w(M)$) and $E(M)$ consists of all pairs $(i_1, j_1) \neq (i_2, j_2)$ in $V(M)$ such that $|i_1 - i_2| \leq 1$ and $|j_1 - j_2| \leq 1$. We say that M is *connected* if the underlying graph $G(M)$ is connected. Given the correspondence between M and $G(M)$ we shall interchangeably refer to 1-pixels in M and vertices of $G(M)$. We shall assume that $w(M) \geq 1$, since we can detect that $w(M) = 0$ by asking for a single sample 1-pixel (and getting none), in which case we can accept.

In this section we describe an algorithm for testing connectivity whose sample complexity, query complexity and running time are

$$\tilde{O}\left(\min\left\{w(M)^{1/2}, n^2/w(M)\right\}\right) \cdot \mathrm{poly}(1/\epsilon) \; .$$

Thus, as long as $w(M) \leq n^{4/3}$, the complexity increases with $w(M)^{1/2}$, and once $w(M)$ goes above $n^{4/3}$, the complexity starts decreasing. We later prove a lower bound of $\Omega(\min\{w(M)^{1/3}, n^2/w(M)\})$ queries (for a constant ϵ) on the complexity of any two-sided error algorithm. We also show that if one requires that the algorithm have one-sided error, then there is no sublinear-time algorithm (that is, there is a lower bound of $\Omega(w(M))$).

3.1 The algorithm

We start by assuming that we are given a constant factor estimate, $\widehat{w}$ of $w(M)$. In Subsection 3.1.2 we remove this assumption (by showing how to estimate $w(M)$ using a procedure that has sample and query complexity $\tilde{O}\left(\min\left\{w(M)^{1/2}, n^2/w(M)\right\}\right)$).

We describe an algorithm such that given $w(M)/c \leq \widehat{w} \leq c \cdot w(M)$ (for some fixed and known constant c), the algorithm has query and sample complexities, as well as running time, $\tilde{O}(\sqrt{w(M)}\epsilon^{-2})$. We may thus assume that $\epsilon = \omega(1/\sqrt{w(M)})$ or else, we can take a single sample 1-pixel, run a Breadth First Search (BFS) to find its connected component in $G(M)$ by performing $O(1/\epsilon^2) = O(w(M))$ queries, and then take an additional sample of $\Theta(1/\epsilon)$ 1-pixels to verify that are are no (few) 1-pixels that do not belong to this component. In Subsection 3.1.1 we show that if $w(M)$ is relatively large (the threshold is roughly around $w(M) = n^{4/3}$), then an alternative (and simpler) algorithm, which generalizes the algorithm in [Ras03], has better performance (and in particular improves as $w(M)$ increases).

The high level idea of the algorithm is as follows: the algorithm tries to find evidence that the tested matrix M is not connected, where the evidence comes in one of the following two forms. (1) "Hard" evidence, in the form of a small connected component in $G(M)$; (2) "Soft" ("statistical") evidence in the form of more than one connected component when viewing the matrix at a "coarser" resolution. Namely, if we partition the matrix into (equal-size) submatrices, and take a sample of

1-pixels, then we can define a graph over those submatrices that contain at least one sample 1-pixel similarly to the way it was defined for single 1-pixels (i.e., $G(M)$). The algorithm checks whether this "backbone' graph is connected. Evidence against connectivity of this type is "soft", or "statistical" since it is possible that the matrix is connected but the sample missed some submatrix, causing the backbone graph to be disconnected. Basing the decision of the algorithm on the second type of evidence and not only on the first, is what makes the algorithm have two-sided error. Evidence of the "hard" form is obtained by performing several BFS's on $G(M)$ (note that the neighbors of a vertex in $G(M)$ that corresponds to an entry (i, j) in M can be obtained by performing 8 queries to M).

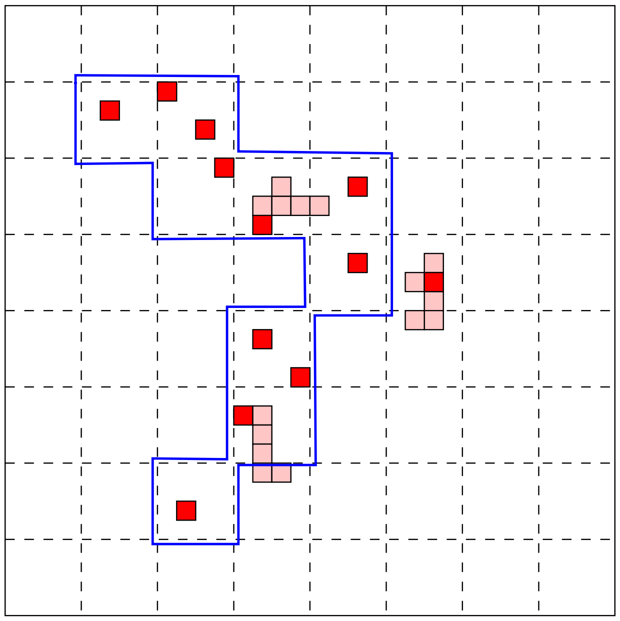

Figure 2: An illustration for the execution of Algorithm 1. The partition into submatrices is marked by a grid of dashed lines. The sampled 1-pixels (in either Step 3 or in Step 4) are marked by dark filled pixels, and the queried entries that are answered by 1 in the course of the BFS's are marked by lighter filled pixels. The backbone is outlined by a bold line. Note that the marked pixels outside the backbone correspond to a BFS performed in Step 4.

Algorithm 1: **Testing connectivity (Version I)**

1. *Consider a fixed partition of M into equal-size submatrices of dimensions $s \times s$ where $s = \sqrt{\widehat{w}/c}$ (recall that $\widehat{w} \leq c \cdot w(M)$ and that the constant c is known to the algorithm).*

2. *Take a sample S_1 of $t_1 = \Theta\left(\sqrt{\widehat{w}} \cdot \log(\widehat{w})\right)$ uniformly distributed 1-pixels in M and consider all non-empty submatrices in the abovementioned partition (that is, all submatrices that contain a sample 1-pixel). Let $B(S_1)$ be the ("backbone") graph whose vertices are the non-empty submatrices, and where there is an edge between two submatrices if they are adjacent (horizontally, vertically, or diagonally). If $B(S_1)$ is not connected, then* REJECT *(otherwise, continue).*

3. *Select, uniformly at random, $t_2 = \Theta(\log(\widehat{w})/\epsilon)$ non-empty submatrices (vertices in $B(S_1)$). For each submatrix selected, consider the first sample 1-pixel that fell into the submatrix, and perform a BFS in $G(M)$ starting from the vertex that correspond to this 1-pixel. Stop once the BFS reaches at least $8\sqrt{c \cdot \widehat{w}}/\epsilon$ vertices in $G(M)$ or the BFS gets "stuck" (a small connected component in $G(M)$ is detected). In the latter case* REJECT *(otherwise, continue).*
4. *Take an additional sample, S_3, of $t_3 = \Theta(1/\epsilon)$ 1-pixels. If any selected 1-pixel belongs to a submatrix that does not neighbor a submatrix in the backbone (a vertex of $B(S_1)$), then* REJECT. *Otherwise, perform a BFS starting from each sample 1-pixel in S_3 as described in the previous step. If a small connected component is found then* REJECT.
5. *If no step caused rejection, then* ACCEPT.

For an illustration of a (successful) execution of the algorithm, see Figure 2.

Theorem 3.1 *Algorithm 1 is a (two-sided error) testing algorithm for connectivity. Its sample and query complexities as well as its running time are $\tilde{O}\left(\sqrt{\widehat{w}} \cdot \epsilon^{-2}\right) = \tilde{O}\left(\sqrt{w(M)} \cdot \epsilon^{-2}\right)$.*

Proof: The sample complexity of the algorithm is $t_1 + t_3 = O\left(\sqrt{\widehat{w}} \cdot \log(\widehat{w}) \cdot \epsilon^{-1}\right)$. Its query complexity is $O(\log(\widehat{w}) \cdot \epsilon^{-1}) \cdot O(\sqrt{\widehat{w}} \cdot \epsilon^{-1}) = O\left(\sqrt{\widehat{w}} \cdot \log(\widehat{w}) \cdot \epsilon^{-2}\right)$, since it performs $t_2 + t_3 = O(\log(\widehat{w}) \cdot \epsilon^{-1})$ searches (in Steps 3 and 4), in each search it reaches $O(\sqrt{\widehat{w}} \cdot \epsilon^{-1})$ vertices in $G(M)$, and determining all neighbors of a vertex in $G(M)$ can be done by performing 8 queries to M. The algorithm can be implemented so that it run in time $\tilde{O}\left(\sqrt{\widehat{w}} \cdot \epsilon^{-2}\right)$.

We turn to analyzing the correctness of the algorithm. In all that follows, when we refer to submatrices, we mean one of the $(n/s)^2 = c \cdot n^2/\widehat{w}$ submatrices in the partition defined by the algorithm. For each $s \times s$ submatrix, we say that the submatrix is *heavy* if the number of 1-pixels in it is at least $s/2$ (recall that $s = \sqrt{\widehat{w}/c}$). Otherwise it is *light*. By our choice of the sample size $t_1 = \Theta(\sqrt{\widehat{w}}\log(\widehat{w}))$, with high constant probability, in Step 2 we'll get at least one sample 1-pixel from each heavy submatrix. This is true because for each heavy submatrix, the probability that a single sample 1-pixel falls into it is at least $\sqrt{\widehat{w}/c}/w(M) = \Omega(1/\sqrt{\widehat{w}})$, and so the probability that we don't get even one sample 1-pixel from the submatrix is at most $(1 - \Omega(1/\sqrt{\widehat{w}}))^{t_1} = 1/\text{poly}(\widehat{w})$. Since the number of heavy submatrices is at most $w(M)/\sqrt{\widehat{w}/c} = O(\sqrt{\widehat{w}})$, by taking a union bound over all heavy submatrices we have that with high constant probability the sample "hits" each heavy submatrix. We shall say in such a case that the sample S_1 is *typical*.

Consider first the case that M is connected. We claim that the probability that it is rejected is at most a small constant. First we observe that M cannot be rejected due to a small connected component of $G(M)$ being found in Step 3 or Step 4, since $G(M)$ consists of a single connected component of size $w(M)$ (and we assume that $\epsilon = \omega(1/\sqrt{\widehat{w}}) = \omega(1/\sqrt{w(M)})$. Next we note that if S_1 is typical, then the backbone graph $B(S_1)$ must be connected. This is true since each submatrix is of size $s \times s$ for $s = \sqrt{\widehat{w}/c}$, so the existence of more than one connected component in $B(S_1)$ implies that some heavy submatrix was not hit (under the premise that M is connected).[5] By the same reasoning, if S_1 is typical, then all 1-pixels in the sample selected in Step 4 belong

[5]To see this, consider the 1-pixels in the area of two connected components in $B(S_1)$, C_1 and C_2. As $G(M)$ is connected, there must be at least one path between these pixels in $G(M)$, and as $B(S_1)$ isn't connected, the length

to submatrices that belong to or neighbor the backbone submatrices. Since the probability that S_1 is not typical is upper bounded by a small constant, M will be accepted with high constant probability.

We now turn to deal with the case that M is ϵ-far from being connected. If the backbone graph $B(S_1)$ is not connected, then M is rejected in Step 2 of the algorithm, so we consider the case that $B(S_1)$ is connected. We say that a submatrix in the backbone is "reliable" if it would pass the BFS test performed by the algorithm in Step 3 (starting from the first sample 1-pixel that fell into it). Otherwise it is "unreliable". Similarly, we say that a 1-pixel is "well connected" to the backbone if it would pass the test performed in Step 4 of the algorithm (that is, it belongs to a submatrix that neighbors one of the backbone submatrices, and a BFS that starts from it does not detect a small connected component).

Suppose that the number of submatrices in the backbone that are unreliable is greater than $(\epsilon/8)\sqrt{w(M)} \geq (\epsilon/8) \cdot \sqrt{\widehat{w}/c}$. Recall that the total number of submatrices in the backbone is at most $t_1 = \Theta\left(\sqrt{\widehat{w}} \cdot \log(\widehat{w})\right)$. Therefore, one of these submatrices is selected with high constant probability in Step 3 of the algorithm (where $\Theta\left(\log(\widehat{w}/\epsilon)\right)$ submatrices of the backbone are selected), causing the algorithm to reject. Similarly, if the fraction of 1-pixels that are not well-connected to the backbone is greater than $\epsilon/8$, then with high constant probability we'll obtain such a 1-pixel in Step 4 of the algorithm and reject.

We next show that if both the number of unreliable submatrices in the backbone is at most $(\epsilon/8)\sqrt{w(M)}$, and the fraction of 1-pixels that are not well-connected to the backbone is at most $\epsilon/8$, then M is ϵ-close to being connected. We show this by describing how M can be made connected with relatively few modifications, building on the backbone. This implies that if M is ϵ-far from being connected then it will be rejected with high constant probability. Details follow.

For each of the reliable submatrices of the backbone, consider the BFS performed starting from the first sample 1-pixel in S_1 that belongs to the submatrix. Since the submatrix is reliable, at least $8\sqrt{c\widehat{w}}/\epsilon$ vertices in $G(M)$ are reached by the BFS. Similarly, for each well-connected 1-pixel, consider the BFS that starts from this 1-pixel and reaches at least $8\sqrt{c\widehat{w}}/\epsilon$ vertices in $G(M)$. Since the total number of vertices in $G(M)$ is $w(M)$, the number of connected components in the subgraph of $G(M)$ that is induced by the union of all these BFS's is at most $\frac{w(M)}{8\sqrt{c\widehat{w}}/\epsilon} \leq (\epsilon/8)\sqrt{w(M)}$. We note that, by their definition, well-connected 1-pixels may belong to submatrices in the backbone. That is, there may be more than one BFS that starts in the same submatrix.

Next we deal with the unreliable submatrices in the backbone (where there are at most $(\epsilon/8)\sqrt{w(M)} \leq (\epsilon/8)w(M)/s$ such submatrices). For each unreliable submatrix in the backbone, we change at most s of the entries in it from 0 to 1, so as to obtain a connected component that corresponds to some arbitrary row in the submatrix (say, the middle row). Let M' be the resulting matrix (where M' and M differ in at most $(\epsilon/8)w(M)$ entries).

At this point we have at most $(\epsilon/4)\sqrt{w(M)}$ connected components in the subgraph of $G(M')$ that is induced by the aforementioned BFS's and the modified entries in the unreliable submatrices. These components intersect all submatrices in the backbone and possibly additional neighboring submatrices (due the the BFS's that start from well-connected 1-pixels). If we consider an auxiliary graph whose vertices are these components and where there is an edge between two components

of this path must be at least s. Let us trace this path until it leaves the submatrices neighboring C_1. In particular, consider the last s 1-pixels in this subpath. These pixels pass through no more than 2 submatrices and so one of these is heavy. Such a heavy submatrix must be hit by S_1 in a typical sample.

if they intersect neighboring submatrices or the same submatrix, then this auxiliary graph is connected. Let T be some (arbitrary) spanning tree of this auxiliary graph. For each edge in the spanning tree (a pair of neighboring (or identical) submatrices that are intersected by different connected components), we can modify at most $2^{3/2}s = 2^{3/2}\sqrt{\widehat{w}/c} \leq 2^{3/2}\sqrt{w(M)}$ entries in the neighboring (or identical) submatrices from 0 to 1 so as to connect the two corresponding connected components, and get a single connected component. Let M'' be the resulting matrix (so that M'' and M' differ on less than $(3\epsilon/4)w(M)$ entries).

Finally, we observe that all vertices in $G(M'')$ that do not belong to the abovementioned single connected component in $G(M'')$ necessarily correspond to 1-pixels in M that are not well-connected (though it is possible that some 1-pixels that are not defined as well-connected belong to the single connected component). Therefore, we can change all these (at most $(\epsilon/8)\cdot w(M)$) entries in M'' from 1 to 0 and remain with a single connected component in the resulting matrix. The total number of modifications made is at most $(\epsilon/8)w(M) + (3\epsilon/4)w(M) + (\epsilon/8)\cdot w(M) = \epsilon w(M)$, implying that M is ϵ-close to being connected, as claimed. ■

3.1.1 An alternative algorithm for dense submatrices

In this subsection we describe an algorithm whose query and sample complexities, as well as its running time are $O((n^2/w(M))\cdot\epsilon^{-3})$, when given an estimate $\widehat{w}$ such that $w(M)/c \leq \widehat{w} \leq c\cdot w(M)$. The algorithm is essentially a generalization of the algorithm of Raskhodnikova [Ras03] for testing connectivity in the dense-images model (which is appropriate when $w(M) = \Theta(n^2)$), and we also apply a lemma proved in [Ras03]. We note that the complexity can be reduced to $O\left((n^2/w(M))\cdot\epsilon^{-2}\log^2(1/\epsilon)\right)$ similarly to what is done in [Ras03] and [GR02]. We discuss this further following the proof of Theorem 3.2.

Thus, we get an improved performance when M is relatively dense (ignoring the dependence on $1/\epsilon$, and logarithmic factors, this occurs around the threshold of $w(M) = n^{4/3}$). Here we assume that $\epsilon = \omega(n^2/\widehat{w}^2) = \omega(n^2/w(M)^2)$, or else, by taking a sample of size $\tilde{\Theta}((n^2/w(M))\cdot\epsilon^{-2})$, we can obtain with high constant probability all the 1-pixels in M.[6]

Algorithm 2: Testing connectivity (Version II)

1. *Take a sample of* $\Theta(1/\epsilon)$ 1-*pixels.*
2. *From each sampled* 1-*pixel perform a BFS in* $G(M)$ *until* $16cn^2/(\epsilon^2\widehat{w})$ *vertices in* $G(M)$ *are reached (recall that* $w(M)/c \leq \widehat{w} \leq c\cdot w(M)$ *and the constant* c *is known to the algorithm) or the BFS gets "stuck" before reaching this number of vertices (a small connected component is found).*
3. *If a small connected component is found, then* REJECT*, otherwise,* ACCEPT.

Theorem 3.2 *Algorithm 2 is a (one-sided error) testing algorithm for connectivity. Its sample complexity is* $O(1/\epsilon)$ *and its query complexity and running time are* $O\left((n^2/w(M))\cdot\epsilon^{-3}\right)$.

We shall apply the next lemma from [Ras03].

[6]Alternatively, we can slightly modify the algorithm so that if it finds a single connected component, then it takes an additional sample of $\Theta(1/\epsilon)$ 1-pixels and rejects only if one of them falls in another connected component.

Lemma 3.1 ([Ras03]) *If $G(M)$ contains at most k connected components then it can be made connected by changing at most $n(\sqrt{2k}+O(1))$ pixels in M from 0 to 1.*

The lemma follows by considering a partition of M into submatrices of size $s \times s$, for $s = n\sqrt{2/k}$, creating a path between the bottom-right corners of all these submatrices, and adding a path from each connected component to the closest corner.

Proof of Theorem 3.2: The bounds on the sample complexity, query complexity and running time of the algorithm follow directly from its description (where again we note that in order to obtain all the neighbors of a vertex in $G(M)$ it suffices to perform 8 queries to M).

Clearly, if M is connected, then the algorithm accepts (with probability 1). Therefore, suppose that M is ϵ-far from being connected. We shall show that in such a case, necessarily the number of vertices in $G(M)$ that belong to connected components of size smaller than $16cn^2/(\epsilon^2\widehat{w})$, is greater than $(\epsilon/2)w(M)$. But in such a case, with high constant probability, the algorithm rejects (since it takes a sample of $\Theta(1/\epsilon)$ vertices in $G(M)$ and from each it performs a BFS until it reaches $16cn^2/(\epsilon^2\widehat{w})$ vertices).

Assume, contrary to the claim, that M is ϵ-far from being connected but in $G(M)$ there are at most $(\epsilon/4)w(M)$ vertices that belong to connected components of size smaller than $16cn^2/(\epsilon^2\widehat{w})$. We next show that in such a case, by modifying at most $\epsilon w(M)$ entries in M we can obtain a connected matrix, which contradicts the premise that M is ϵ-far from being connected.

We can first change from 1 to 0 all the at most $(\epsilon/2)w(M)$ entries in M that correspond to vertices that belong to connected components of size smaller than $16cn^2/(\epsilon^2\widehat{w})$. Consider the remaining big connected components (that is, the components of size at least $16cn^2/(\epsilon^2\widehat{w}) \geq 16n^2/(\epsilon^2 w(M))$). Since there are at most $\frac{w(M)}{16n^2/(\epsilon^2 w(M))} = \frac{\epsilon^2 w(M)^2}{16n^2}$, such components, by Lemma 3.1, we can connect them by modifying at most $n \cdot \left(\frac{\epsilon w(M)}{2\sqrt{2}n} + O(1)\right)$ which is at most $(\epsilon/2)w(M)$. ■

In order to reduce the dependence on ϵ to $\tilde{O}(1/\epsilon^2)$, we do the following. We will consider the possibility of connected components of different sizes in separate iterations. If connected components of small size cause our image to be far from connected, a relatively small number of queries will discover such a component if we land in it. Likewise, if a number of relatively large components is responsible for the image being far from connected, we have a reasonable probability of landing in each such component in a sample, but we must query more pixels to cover it. Obviously a combination of these two possibilities exists, and each iteration i in the following deals with components of a different order of magnitude. The algorithm works in $\log(1/\epsilon)$ iterations. In iteration i it takes a sample of $\Theta(\log(1/\epsilon)/(2^i\epsilon))$ 1-pixels and for each it performs a BFS in $G(M)$ until it reaches $(32cn^2 2^i)/(\epsilon\widehat{w})$ vertices (or it detects a small connected component). The dependence on ϵ is hence reduced to $\tilde{O}(1/\epsilon^2)$. The correctness of the algorithm is implied by the following argument. If for some $1 \leq i \leq \log(1/\epsilon)$, the number of vertices that reside in connected components of size at most $(32cn^2 2^i)/(\epsilon\widehat{w})$ is at least $2^i\epsilon w(M)/(4\log(1/\epsilon))$, then, with high constant probability, the algorithm rejects in iteration i.

But otherwise, we have that there are at most $\epsilon w(M)/(2\log(1/\epsilon))$ vertices in components of size at most $(32cn^2)/(\epsilon\widehat{w})$, and for each $1 \leq i \leq \log(1/\epsilon)$ there are at most $2^i\epsilon w(M)/(4\log(1/\epsilon))$ vertices in components of size at most $(32cn^2 2^i)/(\epsilon\widehat{w})$ and at least $(32cn^2 2^{i-1})/(\epsilon\widehat{w})$. We next show that after changing from 1 to 0 all (at most $(\epsilon/2)w(M)$) pixels that belong to components of size at most $(32cn^2)/(\epsilon\widehat{w})$, we have at most $\epsilon^2 w(M)^2/(16n^2)$ connected components. For each $1 \leq i \leq \log(1/\epsilon)$, the number of components of size $(32cn^2 2^i)/(\epsilon\widehat{w})$ and at least $(32cn^2 2^{i-1})/(\epsilon\widehat{w})$

is upper bounded by $\frac{2^i\epsilon w(M)/(4\log(1/\epsilon))}{(32cn^2 2^{i-1})/(\epsilon\widehat{w})} \leq \epsilon^2 w(M)^2/(64\log(1/\epsilon)$, and the number of components of size greater than $(32cn^2)/(\epsilon^2\widehat{w})$ is upper bounded by $\epsilon^2 w(M)^2/(32\log(1/\epsilon))$. Adding them all up, we get the desired bound on the number of connected components and we can apply Lemma 3.1.

3.1.2 Obtaining an estimate of $w(M)$

In this subsection we show how to obtain an estimate $\widehat{w}$ such that with high constant probability, $w(M)/c \leq \widehat{w} \leq c \cdot w(M)$ for some constant c. Given the estimate we get, we decide whether to run Algorithm 1 or Algorithm 2 (where the algorithm is provided with the estimate $\widehat{w}$).

We next show how to obtain such an estimate by taking a sample of size $O(\min\{\sqrt{w(M)}, n^2/w(M)\})$ and performing at most these many queries. The idea behind the algorithm is the following. It is possible to obtain an estimate of $w(M)$ in two different ways. Roughly speaking, the first achieves a better performance when $w(M)$ is "large", and the second when $w(M)$ is "small".

Specifically, the first approach is to simply query uniformly selected entries in M and take the fraction of entries that are 1 (among those queried) to be an estimate for $p(M) \stackrel{\text{def}}{=} w(M)/n^2$. If we want to obtain an estimate $\hat{p}$ such that with high constant probability, $p(M)/c \leq \hat{p} \leq cp(M)$, then by the multiplicative Chernoff bound, it suffices to query $\Theta(1/p(M)) = \Theta(n^2/w(M))$ uniformly selected entries in M. The difficulty of course is that we do not know $w(M)$ so that it is not clear how many queries to perform so as to get such an estimate. We can however perform queries (on uniformly selected 1-pixels), until we get a certain constant number of "hits" (i.e., queries (i, j) for which $M[i, j] = 1$) and use the number of queries performed to derive an estimate for $1/p(M) = n^2/w(M)$. (In fact, for a rough estimate, it suffices to obtain a single hit.) The drawback of this approach is that if $w(M)$ is very small (in particular, $n^2/w(M)$ is significantly greater than $\sqrt{w(M)}$), then the query complexity is large.

The second approach is based on collision probabilities (where in the term *collision* we mean getting the same 1-pixel twice when uniformly sampling 1-pixels). Namely, if there are $w(M)$ entries in M that are 1, then if we take a sample of significantly less than $\sqrt{w(M)}$ (uniformly selected) 1-pixels then we do not expect to see any collision, while if we take a sample of, say, $4\sqrt{w(M)}$ 1-pixels, then we expect to get a collision with high constant probability.

In order to "enjoy both worlds", we run two iterative processes "in parallel". One process is the "query process" and the other is the "sampling process". Namely, in each iteration, we both ask a new query (uniformly selected among all entries $(i, j) \in [n] \times [n]$, where $n = \{1, \ldots, n\}$), and we ask for a new sample 1-pixel (which is uniformly selected among all entries (i, j) such that $M[i, j] = 1$). We stop once we either get a positive answer to a query (i.e., we "hit" an entry (i, j) such that $M[i, j] = 1$ in a uniformly selected query), or we get a collision in our sampled 1-pixels (i.e., we get the same sample 1-pixel (i, j) a second time). Let t be the number of queries performed (samples obtained) until one of these events occurs. If the first event occurs, then we output $\widehat{w} = n^2/t$, and if the second event occurs, then we output t^2.

It remains to bound the probability that $\widehat{w} > c \cdot w(M)$ or that $\widehat{w} < w(M)/c$. The first event may occur either because the query process causes the algorithm to stop when $t < t_1$ for $t_1 = (1/c)(n^2/w(M))$ or because the sampling process causes the algorithm to stop only when $t > t_2$ for $t_2 = \sqrt{c \cdot w(M)}$. The probability that the query process stops prematurely is upper bounded by $t_1 \cdot w(M)/n^2 = 1/c$, and the probability that the sampling process doesn't stop before

t reaches t_2 is upper bounded by $(1-(t_2/2)\cdot(1/w(M)))^{t_2/2} = (1-\sqrt{c/w(M)}/2)^{\sqrt{c\cdot w(M)}/2} < e^{-c/4}$. The latter bound can be derived by considering a partition of the t_2 samples into two equal parts, p_1 and p_2. The expression $(1-(t_2/2)\cdot(1/w(M)))$ is the probability for each sample in p_2 *not* to hit a sample in p_1 given that all the samples in p_1 are distinct (which they are, or we would have stopped earlier). This process is repeated $t_2/2$ times once t_2 samples have been taken, and thus at least one sample will be hit twice with probability greater than $(1-(t_2/2)\cdot(1/w(M)))^{t_2/2}$.

Similarly, the second event ($\widehat{w} < w(M)/c$) may occur because the sampling process causes the algorithm to stop when $t < t_2'$ for $t_2' \leq \sqrt{w(M)/c}$. The probability of this is upper bounded by $\binom{t_2'}{2}\cdot(1/w(M)) < 1/c$, the union bound on the probability of any two such samples returning the same location. The other possibility for underestimation is that the query process stops only when $t > t_1'$ for $t_1' = c\cdot(n^2/w(M))$, which is upper bounded by $(1-w(M)/n^2)^{t_1'} < e^{-c}$.

The above discussion also implies that the probability that the estimation algorithm performs more than $\min\{\sqrt{c\cdot w(M)}, c\cdot(n^2/w(M))\}$ queries (takes more than these many samples), decreases exponentially with c.

3.2 Lower bounds on testing connectivity

We start with a simple lower bounded for one-sided error algorithms, and then turn to two-sided error algorithms.

Theorem 3.3 *Any one-sided error testing algorithm for connectivity must perform $\Omega(\min\{w(M), n^2/w(M)\})$ queries (for a constant ϵ). The lower bound holds when the algorithm is given an estimate $\widehat{w}$ such that $w(M)/2 \leq \widehat{w} \leq 2w(M)$.*

Proof: By the definition of one-sided error testing algorithm, a one-sided error testing algorithm for connectivity can reject only if the sample and queries that it observes are not consistent with *any* connected matrix M for which $w(M)/2 \leq \widehat{w} \leq 2w(M)$. That is, the entries of the matrix that are viewed by the algorithm imply that $G(M)$ must contain more than one connected component.

Consider first the case that $\widehat{w} \leq 2n$ (so that the lower bound is $\Omega(w(M))$). Let M be the matrix in which $M[1,1] = 1, \ldots, M[1, \widehat{w}/2] = 1$, $M[\widehat{w}/2, 1] = 1, \ldots, M[\widehat{w}/2, \widehat{w}/2] = 1$ and $M[i,j] = 0$ for every other entry (i,j). The matrix M is $\Omega(1)$-far from being connected (and $w(M) = \widehat{w}$). However even if we give the algorithm sample 1-pixels "for free" (and the estimate $\widehat{w}$ so that it knows that $w(M) \leq 2\widehat{w}$), then the algorithm cannot reject before it performs $\Omega(w(M))$ queries, since it won't observe a "vertex cut" between the two sub-rows.

If $\widehat{w} > 2n$ then we generalize the above construction as follows (where we assume for simplicity that $\widehat{w}$ is divisible by $2n$ and that n^2 is divisible by $2\widehat{w}$ or else some rounding is required). We take $2\widehat{w}/n$ equally spaced rows in the matrix (so that the distance between every two rows is $n^2/(2\widehat{w})$), and in each row we put 1's in $\widehat{w}/(2n)$ subrows, each of length $n^2/\widehat{w}$, where each two consecutive subrows are at distance $n^2/\widehat{w}$ from each other. Here too, the matrix is $\Omega(1)$-far from being connected (and $w(M) = \widehat{w}$), but every one-sided error algorithm must perform $\Omega(n^2/w(M))$ queries. ■

Theorem 3.4 *Any (two-sided) error testing algorithm for connectivity has sample complexity and query complexity $\Omega(\min\{w(M)^{1/3}, n^2/w(M)\})$ (for a constant ϵ). The lower bound holds when the algorithm is given an estimate $\widehat{w}$ such that $w(M)/2 \leq \widehat{w} \leq 2w(M)$.*

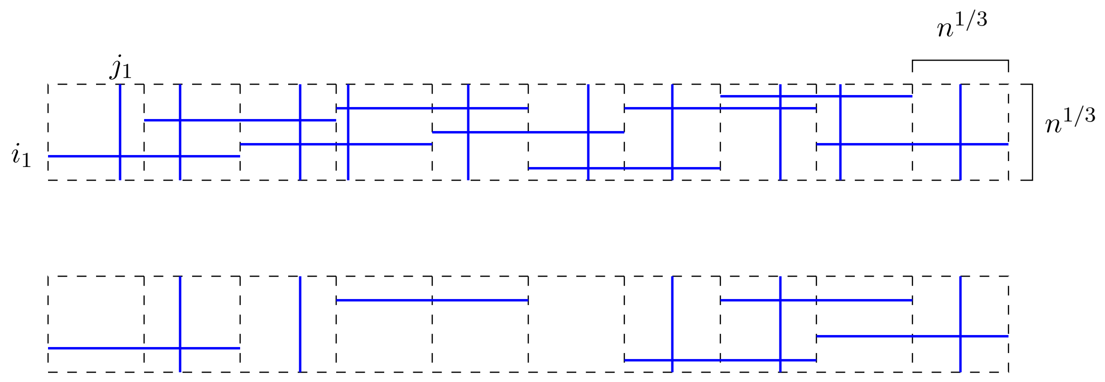

Figure 3: An illustration for the proof of Theorem 3.4 for $\widehat{w} = n$. On the top is an example of the first row of submatrices of a matrix in $\mathcal{F}_1$, and on the bottom is an example of the first row of submatrices of a matrix in $\mathcal{F}_2$. The outline of the submatrices is marked in dashed lines, and the rows and columns of 1-pixels are marked by bold lines.

Proof: We first establish the claim for $\widehat{w} = \Theta(n)$ (where the bound is $\Omega(n^{1/3})$), and later explain how to modify it to smaller and larger values of $\widehat{w}$ (and hence $w(M)$). In order to prove the lower bound we define two families of matrices. In the first family, denoted $\mathcal{F}_1$, all matrices are connected, and in the second family, denoted $\mathcal{F}_2$, with very high probability over the choice of a random matrix in $\mathcal{F}_2$, the matrix is $\Omega(1)$-far from being connected. We shall show that any algorithm that takes a sample of size $o(n^{1/3})$ and performs at most these many queries, cannot distinguish with constant probability between a matrix selected uniformly at random from $\mathcal{F}_1$ and a matrix selected uniformly at random from $\mathcal{F}_2$.

Defining the two families. Consider a partition of the entries of an $n \times n$ matrix into submatrices of dimensions $n^{1/3} \times n^{1/3}$. For both families there will actually be 1-pixels only in the first "row" of these submatrices, where we number this sequence of submatrices from 1 to $n^{2/3}$ (from left to right).

Each matrix in $\mathcal{F}_1$ is determined by $2n^{2/3}$ integers, $i_1, \ldots, i_{n^{2/3}}$ and $j_1, \ldots, j_{n^{2/3}}$, where $1 \leq i_k, j_k \leq n^{1/3}$ for every $1 \leq k \leq n^{1/3}$. These integers determine the locations of the 1-pixels in the matrix in the following way: For each k, there are 1-pixels in row i_k of submatrix number k and submatrix number $k+1$ (if such exists), and there are 1-pixels in column j_k of submatrix number k. All other entries are 0.

Each matrix in $\mathcal{F}_2$ is determined by two subsets $T_r, T_c \subset [n^{2/3}]$ each of size $n^{2/3}/2$ and by $n^{2/3}$ indices $\{i_k\}_{k \in T_r} \cup \{j_\ell\}_{\ell \in T_c}$ where $1 \leq i_k, j_\ell \leq n^{1/3}$ for every $k \in T_r$ and $\ell \in T_c$. These integers determine the locations of the 1-pixels in the matrix in the same way that was defined for matrices in $\mathcal{F}_1$. That is, For each $k \in T_r$, there are 1-pixels in row i_k of submatrix number k and submatrix number $k+1$ (if such exists), and for each $\ell \in T_c$ there are 1-pixels in column j_ℓ of submatrix number ℓ. All other entries are 0. Thus, the difference between matrices in $\mathcal{F}_1$ and matrices in $\mathcal{F}_2$, is that in the latter family, there may be submatrices (in the first row of submatrices) that are "empty" (contain only 0's) or contain only a row of 1-pixels and no column of 1-pixels.

Properties of the two families. By the above description, every matrix in $\mathcal{F}_1$ is connected. On the other hand, as we explain next, with high probability a uniformly selected matrix in $\mathcal{F}_2$ will be

$\Omega(1)$-far from being connected. The reason is that with high probability over the choice of M the following three events will occur:

1. The columns $1, \ldots, n/3$ will contain at least $n/12$ different 1-pixels.
2. The columns $2n/3 + 1, \ldots, n$ will contain at least $n/12$ different 1-pixels.
3. At least $n/24$ of the columns $n/3 + 1, \ldots, 2n/3$ will contain no 1-pixels.

Thus M will be $\Omega(1)$-far from connected, as either all 1-pixels must be removed from one of the sides of the image, or each column in the middle of the image must contain at least a single 1-pixel.

The difficulty of distinguishing between the two families. Consider any (two-sided error) algorithm for testing connectivity of matrices that takes a sample of $o(n^{1/3})$ 1-pixels and asks $o(n^{1/3})$ queries. We may assume without loss of generality that it first takes the sample and then performs all queries (possibly adaptively). We first show that for both families, the distributions on sampled 1-pixels are very similar. More precisely, we show that unless a certain low probability event occurs, the distributions on samples are identical. We later deal with the answers to queries.

Since once the algorithm is given a sample 1-pixel it can determine in an additional constant number of queries whether the 1-pixel belongs to a row or to a column (and in the former case whether the row extends to the next submatrix or to the previous submatrix), we assume that the algorithm is actually given a sample of rows/columns. That is, each sample is either of the form (k, i_k) or (ℓ, j_ℓ) for $k, \ell \in [n^{2/3}]$ and $i_k, j_k \in [n^{1/3}]$. Rather than first selecting a matrix uniformly from $\mathcal{F}_1$ (similarly, uniformly from $\mathcal{F}_2$) and then generating a sample, we may think of the sample being generated in the process of determining the uniformly selected matrix. Specifically, for each family we define a "process" ($\mathcal{P}_1$ for $\mathcal{F}_1$ and $\mathcal{P}_2$ for $\mathcal{F}_2$) that generates samples that are distributed according to a uniformly selected matrix in the family, while constructing the matrix. This is done as follows.

For each new sample, both of the processes first flip a coin with bias $2/3$ to decide whether to generate a row or a column (as in both families the number of 1-pixels that belong to rows is twice as large as the number of 1-pixels that belong to columns). Suppose that a row is to be generated (the generation of a column is analogous). Let t be the number of different rows already generated (where by our assumption on the sample complexity of the algorithm, $t = o(n^{1/3})$). Then the process $\mathcal{P}_1$ flips a coin with bias $t/n^{2/3}$ to determine whether the new row will be identical to a row that already appeared in the sample. If the coin turns out "heads", then one of the previously generated rows is selected to be the next sample row, while if the coin turns out "tails", then the process uniformly selects a submatrix k that is not yet associated with a row (that is, there is no row starting at this submatrix and ending in the next). It then uniformly selects $i_k \in [n^{1/3}]$ to determine the position of the row in submatrix k (and $k + 1$). The process $\mathcal{P}_2$ does the same *except* that it flips a coin with bias $t/(n^{2/3}/2) = 2t/n^{2/3}$. The important observation is that for both processes, for any choice of a prefix of the sample, conditioned on the coin coming up "tails" (which occurs with probability at least $1 - 2t/n^{2/3}$ in both cases), the distribution over the new row is *identical*.

The above discussion implies, that, since the algorithm takes a sample of size $o(n^{1/3})$, with probability at least $1 - o(n^{1/3}) \cdot o(n^{1/3})/n^{2/3} = 1 - o(1)$, the distributions over the samples that the algorithm observes are identical for both families (processes).

It remains to deal with the queries. Here the argument is even simpler, where we now let the two processes answer queries while continuing to construct a matrix in their respective families. We may assume, without loss of generality, that the algorithm does not ask queries about 1-pixels that belong to rows/columns that it observed in the sample (as we already gave the algorithm the complete row/column "for free"). Thus the algorithm only asks queries about entries that do not belong to sample rows/columns. We claim that given that the algorithm asks $o(n^{1/3})$ queries, for both processes (families), with probability $1 - o(1)$, all queries are answered by 0.

To verify this, consider any fixed submatrix k. If the algorithm asked already t queries in the submatrix, so that the queries belong to at most t rows and at most t columns, and they were all answered by 0, for both processes, the probability that the next query in the submatrix is answered by 1 is upper bounded by $O(1/(n^{1/3} - t))$. Since the algorithm performs $o(n^{1/3})$ queries, the probability that it gets an answer of 1 to any of its queries, is $o(1)$, as claimed.

Thus, if the algorithm takes a sample of size $o(n^{1/3})$ and performs $o(n^{1/3})$ queries, then with probability at least $1-o(1)$ the distributions on samples and the answers to its queries are identical when the matrix is uniformly selected in $\mathcal{F}_1$ and when it is uniformly selected in $\mathcal{F}_2$. This implies that there is no testing algorithm with this complexity for the property of connectivity when $w(M) = \Theta(n)$.

Dealing with $w(M)$ that is not $\Theta(n)$. To extend the lower bound to $w(M) = \Theta(\widehat{w})$ for $\widehat{w} < n$, we can easily scale-down the lower bound construction as follows. We take a partition of M into submatrices of size $\widehat{w}^{1/3} \times \widehat{w}^{1/3}$, consider the first $\widehat{w}^{2/3}$ such submatrices in the first row of submatrices, and replace each occurrence of n in the above construction (and corresponding analysis) by $\widehat{w}$.

To extend the bound to $w(M) = \Theta(\widehat{w})$ for $\widehat{w} > n$, we further consider two cases. If $\widehat{w} = O(n^{3/2})$ (so that the lower bound should still be $\Omega(\widehat{w}^{1/3})$), then we consider a partition into submatrices of size $\widehat{w}^{1/3} \times \widehat{w}^{1/3}$ (as in the case that $\widehat{w} \le n$), but the construction uses not only the first row of submatrices but rather $\widehat{w}/n$ (which is at most $n/\widehat{w}^{1/3}$) such rows. Here we have that the total number of submatrices considered is of the order of $\widehat{w}^{2/3}$. To ensure connectivity between different rows of submatrices we can add (in both families) the last column of the matrix, and in order to simplify the argument that a random matrix in $\mathcal{F}_2$ is $\Omega(1)$-far from being connected, we can use only half of the rows of submatrices (alternating between an "occupied" row and an "empty" row). Other than that, the argument remains essentially as is (with n replaced by $\widehat{w}$).

Finally, if $\widehat{w} = \Omega(n^{3/2})$ (so that the lower bound should be $\Omega(n^2/\widehat{w})$), then we consider a partition into submatrices of size $(n^2/\widehat{w}) \times (n^2/\widehat{w})$, where we use all $\widehat{w}/(2n)$ odd numbered "rows" of submatrices (and add the rightmost column as in the previous case). Since the number of submatrices is of the order of $(\widehat{w}/n)^2$, the probability of getting a collision (that is, that a sample row/column hits the same submatrix twice) using $o(n^2/\widehat{w})$ samples, is even smaller than in the other cases (since $n^2/\widehat{w} < \widehat{w}/n$ for the current setting of $\widehat{w}$). Turning to the queries, if the algorithm performs $o(n^2/\widehat{w})$ queries, then for both distributions it will with high probability see only 0's, as in the other cases. ■

4 Testing and Learning in the Sparse Image Model

In the standard PAC learning model [Val84], a learning algorithm is given access to examples that are distributed according to a fixed underlying distribution D and labeled by an unknown target function f from a known class $\mathcal{C}$ of Boolean functions (also known in the learning theory literature as *concepts*). Given an error (or distance) parameter ϵ and a confidence parameter δ, the algorithm is required, based on the labeled samples it has seen, to output a hypothesis h for which the following holds. With probability at least $1-\delta$, taken over the selection of the examples and possibly the internal coin-flips of the algorithm, the distance between h and f with respect to the underlying distribution D, is at most ϵ (i.e., $\Pr_{x\sim D}[h(x) \neq f(x)] \leq \epsilon$). If the algorithm always outputs a hypothesis $h \in \mathcal{C}$, then it is a *proper* learning algorithm. Variants of this model may allow the algorithm to have query access to the target function, and it may also be assumed that the underlying distribution D is known, and in particular that it is the uniform distribution over the domain.

It was observed in [GGR98], that given a proper learning algorithm for a class of Boolean functions $\mathcal{C}$ (which is allowed queries and works under the uniform distribution), we can easily transform it into a testing algorithm for the property of membership in this class, with the same complexity. In particular, this implies that we can get testing algorithms for membership in classes of functions where the complexity of the algorithm depends linearly on the VC-dimension [VC71] of the class (we recall the notion of the VC-dimension momentarily).

In our context, an image M can be seen as a Boolean function $f_M : [n] \times [n] \to \{0, 1\}$. Therefore, the existence of a proper learning algorithm for a certain class of images under the uniform distribution over $[n] \times [n]$ and with queries, implies testing the class in the dense-images model (with essentially the same complexity). However, in our sparse-images model, the distance measure is different, and furthermore, we are given access to uniform samples of 1-pixels. Thus, the corresponding notion of learning needs to be modified.

Specifically, a learning algorithm for a class $\mathcal{C}$ of sparse images is given query access to an unknown image M in the class as well as access to uniformly selected 1-pixels in the image. The algorithm is also given a distance parameter ϵ, and is required to output a (representation of) an image $\widehat{M}$ such that with probability at least[7] $2/3$, the hypothesis image $\widehat{M}$ and the target image M differ on at most $\epsilon \cdot w(M)$ pixels.

While our focus is on *testing* algorithms for properties of sparse images (membership in classes of sparse images), and we state our results for such algorithms, in this section we actually perform "testing through learning", even when this is not stated explicitly. We first describe two general results, and then consider a particular basic class of sparse images – line imprints, and present a special purpose algorithm (that does not directly apply the general results but rather uses ideas from them). We also note that the results presented here, though stated in the context of sparse images, are not restricted to this domain. Rather they apply in general to learning and testing membership for classes of sparse functions.

4.1 Testing sparse images and the VC-dimension

In what follows we may view an image M as the set of its 1-pixels. Recall that the VC-dimension [VC71] of a class of images (sets) $\mathcal{C}$ is the size of the largest subset of pixels in $[n] \times [n]$ that is *shattered* by $\mathcal{C}$. A subset S is said to be shattered by $\mathcal{C}$ if for every $T \subseteq S$ there exists an image $M \in \mathcal{C}$ such that $M \cap S = T$. The relation between small VC-dimension and PAC learnability is well known. Here we adapt this knowledge to our context of testing sparse images (where, as

[7]For the sake of simplicity, and in order to be consistent with our definition of testing, we set $\delta = 1/3$. As usual, it is possible to increase the success probability of the algorithm to $1 - \delta$ at a multiplicative cost of $\log(1/\delta)$.

noted previously, we actually perform testing through learning, where learning here is in the sense discussed above).

Lemma 4.1 *Let $\mathcal{C}$ be a class of images with VC-dimension d. There exists a* one-sided error *property testing algorithm for testing membership in $\mathcal{C}$ (in the sparse-images model) that given $w = w(M)$ for the tested image M has sample complexity $O(d \log(1/\epsilon)/\epsilon)$ (and performs no queries).*

Proof: Denote by $\mathcal{C}_w$ the subclass of $\mathcal{C}$ where all hypotheses have Hamming weight w. Now consider any hypothesis image H in this class that differs from M on more than $\epsilon \cdot w$ pixels (i.e., viewing the two images as sets of 1-pixels, $|M \setminus H| + |H \setminus M| > \epsilon \cdot w$). Since $w(H) = w$ (as $H \in \mathcal{C}_w$), we have that $|M \setminus H| > \epsilon/2$, and so the hypothesis H is at least $(\epsilon/2)$-far from M with respect to the uniform distribution on 1-pixels in M. We claim that in order to test for membership in $\mathcal{C}$ it suffices to sample $\Theta\left(\frac{d \log(1/\epsilon)}{\epsilon}\right)$ 1-pixels uniformly from M, accept if there is a member of $\mathcal{C}_w$ that is consistent with this sample, and reject otherwise.

Clearly, if $M \in \mathcal{C}$ (so that $M \in \mathcal{C}_w$ since $w = w(M)$) then we accept (with probability 1). To prove that if M differs from every image in $\mathcal{C}$ on more than $\epsilon \cdot w$ pixels, then it is rejected with high constant probability, we consider the class $\mathcal{C}' = \mathcal{C}_w \cup \{M\}$ (that has a VC-dimension at most $d+1$, and by its definition contains M). By [VC71] (see also [KV94, Sec. 3.5]) we know that there exists a constant c_1 such that for any fixed distribution D, with high constant probability, a sample of $c_1 \frac{d \log(1/\epsilon)}{\epsilon}$ examples distributed according to a distribution D, forms an $(\epsilon/2)$-*net* for the concept class with respect to D (or, more precisely, for the class of symmetric differences defined by the concept class, which has the same VC-dimension as the original class). Namely, with high constant probability, the sample is such that for every two functions (images) in the class that have distance greater than $\epsilon/2$ according to D, the sample contains at least one point (pixel) on which they disagree.

It follows that if M differs from every image in $\mathcal{C}$ on more than $\epsilon \cdot w$ pixels, so that it has distance greater than $\epsilon/2$ from every image in $\mathcal{C}_w$ with respect to the distribution D that is uniform on the 1-pixels of M, then with high constant probability, no image in $\mathcal{C}_w$ will be consistent with a sample of 1-pixels of M that has size $\Theta\left(\frac{d \log(1/\epsilon)}{\epsilon}\right)$. ■

A difficulty with applying Lemma 4.1 is that it requires knowing $w = w(M)$ exactly. However, the lemma and proof above can be adapted to the case where we have an estimate $\widehat{w}$ such that $w(M)/(1+\epsilon/4) \le \widehat{w} \le (1+\epsilon/4) \cdot w(M)$. In such a case we will simply test consistency with a subset of $\mathcal{C}$ containing all images whose Hamming weight is in the interval $[\widehat{w}/(1+\epsilon/4), (1+\epsilon/4) \cdot \widehat{w}]$, and take a sample that is a constant factor larger. If $M \in \mathcal{C}$ then it is still accepted with probability 1 (since $w = w(M) \in [\widehat{w}/(1+\epsilon/4), (1+\epsilon/4) \cdot \widehat{w}]$). On the other hand, for every H that differs from M on more than $\epsilon w(M)$ pixels and for which $w(H) \in [\widehat{w}/(1+\epsilon/4), (1+\epsilon/4) \cdot \widehat{w}] \subseteq [w(M)/(1+3\epsilon/4), (1+3\epsilon/4) \cdot w(M)]$, we have that $|M \setminus H| > \epsilon/8$. Therefore, we can apply the same argument as in the proof of Lemma 4.1 (using a slightly bigger sample) to infer that if M is ϵ-far from membership in $\mathcal{C}$, then with high constant probability, it will be rejected.

We next extend our approach to deal with the case where we don't have a good approximation of w. This comes at a cost of increasing the complexity of the algorithm by a factor of $\log \log n$, and obtaining a two-sided error algorithm rather than a one-sided error algorithm.

Lemma 4.2 *Let $\mathcal{C}$ be a class of images with VC-dimension d. There exists a* two-sided error *property testing algorithm for membership in $\mathcal{C}$ (in the sparse-images model) whose sample complexity is $O(\log\log n \cdot d\log(1/\epsilon)/\epsilon)$ and whose query complexity is $O(1/\epsilon)$.*

Proof: Denote by $\mathcal{C}_{\leq w}$ the subset of $\mathcal{C}$ containing images with Hamming weight *at most* w. The two-sided error algorithm will perform a binary search to try and find a hypothesis image H that is consistent with a sample of 1-pixels from M (whose size is as stated in the lemma) and such that H belongs to $\mathcal{C}_{\leq w}$ for as small a value of w as possible.

We first check whether $w(M) = 0$ by requesting a single sample 1-pixel from M. If no pixel is returned, then we accept or reject depending on whether the empty image belongs to $\mathcal{C}$ (i.e., $\mathcal{C}_{\leq 0}$ is non-empty). Assuming M is not the empty image, we start the search from $\mathcal{C}_{\leq n^2} = \mathcal{C}$. Namely, we take a sample of size $\Theta(\log\log n \cdot d\log(1/\epsilon)/\epsilon)$ (which will be "re-used" in the different stages of the search), and we first check whether $\mathcal{C}_{\leq n^2}$ contains a consistent hypothesis. If no consistent hypothesis is found in $\mathcal{C}_{\leq n^2} = \mathcal{C}$, then we may reject. Otherwise, we continue the search in an iterative manner. At the start of each iteration we have two values, w_1 and w_2 such that $\mathcal{C}_{\leq w_2}$ contains a hypothesis image that is consistent with the sample, while $\mathcal{C}_{\leq w_1}$ does not. We then set $w = \lfloor \frac{w_1+w_2}{2} \rfloor$ an check whether $\mathcal{C}_{\leq w}$ contains a hypothesis that is consistent with the sample.

The search terminates after $O(\log n)$ iterations, once we find the *smallest* value w such that $\mathcal{C}_{\leq w}$ contains a consistent hypothesis H. Given H, we uniformly select $\Theta(1/\epsilon)$ pixels among the 1-pixels of H, and we query M on these pixels. If more than an $(\epsilon/2)$-fraction of the queries are answered by 0, then we reject, otherwise we accept.

We next prove the correctness of this procedure. The size of the sample was selected so that for each fixed choice of w, with probability at most $1/(c\log n)$ (for a sufficiently large constant c), there exists a hypothesis in $\mathcal{C}_{\leq w}$ that differs from M on more than $(\epsilon/4)w(M)$ of the 1-pixels in M, but is consistent with the sample. By a union bound over the $2\log n$ iterations of the algorithm, such a hypothesis exists for some class $\mathcal{C}_{\leq w}$ considered with probability at most $1/c$. Assume from this point on that no such hypothesis exists. In particular, for the final hypothesis H we have that $|M \setminus H| \leq (\epsilon/4) \cdot w(M)$.

If M is ϵ-far from $\mathcal{C}$, then this implies that $|H \setminus M| > (3\epsilon/4) \cdot w(M) \geq (3\epsilon/5) \cdot w(H)$ (where we assume that $\epsilon \leq 1/3$ or else we run the procedure with $\epsilon = 1/3$). By a multiplicative Chernoff bound we get that with high constant probability, a sample of $\Theta(1/\epsilon)$ pixels among the 1-pixels of H will contain at least an $(\epsilon/2)$-fraction of pixels that do not belong to M. On the other hand, assume $M \in \mathcal{C}$. By definition of $w = w(H)$ we have that $w - 1 < w(M)$, so that $w(H) \leq w(M)$. Since $|M \setminus H| \leq (\epsilon/4) \cdot w(M)$, we also have that $|H \setminus M| \leq (\epsilon/4) \cdot w(M) \leq (3\epsilon/11) \cdot w(H)$ (again assuming that $\epsilon \leq 1/3$), and so with high constant probability a sample of $\Theta(1/\epsilon)$ pixels among the 1-pixels of H will contain less than an $(\epsilon/2)$-fraction of pixels that do not belong to M. ■

4.2 Testing for line imprints

A natural way to define a line in the sparse-images model is to consider the coordinates of pixels in the image as whole numbers, and to imagine them superimposed on the real plane. Any line in the real plane intersects a subset of these pixels, and we think of this subset as the *imprint* of the line.

Definition 4.1 *The* imprint *of a line (or line segment) is the set of pixels the line (segment) intersects.*

The property of being a line imprint corresponds naturally to the dense half-plane property studied by Raskhodnikova [Ras03]. However, while there the half-plane must go from one side of the image to another, we are willing to allow the line imprint (or, perhaps more appropriately, the line-segment imprint) to begin and end at arbitrary locations in the image. We describe an algorithm that tests for the property of being an imprint of a line. Adapting it to require the imprint to span the image is straightforward.

The approach described in Lemma 4.1 will not lead us directly to a good testing algorithm for a line imprint. While taking a sample of $\Theta(1/\epsilon)$ 1-pixels and considering the distance between the furthest pair of pixels will give us a good estimation of $w(M)$ if M is indeed a line imprint, it may be far from the mark if M is not. We may use the approach described in Lemma 4.2, but this will give us a two-sided error testing algorithm that depends on n, and as we will now show, we can do better.

We first introduce the following definition:

Definition 4.2 *The* sleeve *defined by two pixels is the union of all the imprints of line segments starting in one pixel and ending in the other.*

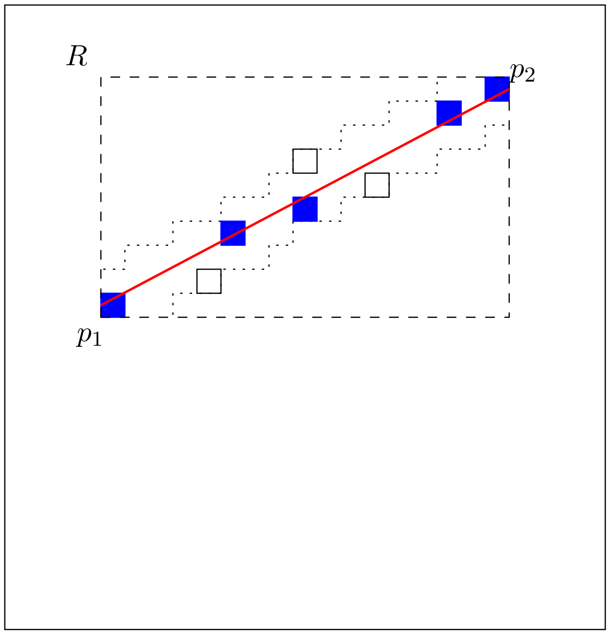

Figure 4: An illustration for the execution of Algorithm 3 and its analysis. The furthest two 1-pixels p_1 and p_2 define the sleeve P whose borders are marked by dotted lines. The additional 1-pixels belong to $S_1 \cup S_2$ as well as the queries in T that were answered by 1. The queries in T that were answered by 0 are marked by empty squares. A line whose imprint is consistent with the samples and queries is marked as well. The rectangle R is as defined in the analysis of the algorithm.

Algorithm 3: **Testing for a Line Imprint**

1. *Take a sample S_1 of $m_1 = \Theta(1/\epsilon)$ uniformly distributed 1-pixels in M. Let p_1,p_2 be two 1-pixels from S_1 with maximum distance between them.*

2. *Take an additional sample S_2 of $m_2 = \Theta(1/\epsilon)$ uniformly distributed 1-pixels in M.*

3. *Let P be the sleeve defined by p_1 and p_2. Draw uniformly a set T of $m_3 = \Theta(\log(1/\epsilon)/\epsilon)$ pixels from P and query M on the pixels in T. If there exists a line imprint that is consistent with the 1-pixels in $S_1 \cup S_2$ and with the answers to the queries in T, then* ACCEPT*, otherwise* REJECT.

Before proving the correctness of Algorithm 3, we state without proof two claims that are relatively straightforward and that we use in the proof of correctness.

Claim 4.3 *Every sleeve between two 1-pixels contains at most three pixels in each row or at most three pixels in each column.*

Claim 4.4 *For every two pixels p_1 and p_2, the class of all line imprints containing p_1 and p_2 has constant VC-dimension.*

Theorem 4.1 *Algorithm 3 is a one-sided error testing algorithm for the property of being a line imprint. Its sample and query complexities are $O(\log(1/\epsilon)/\epsilon)$. Its running time is polynomial in $1/\epsilon$.*

Proof: The bound on the sample and query complexities follows immediately from the description of the algorithm. The algorithm can be implemented so that it runs in time polynomial in $1/\epsilon$, e.g., by linear programming. One direction of the theorem is straightforward. If M is indeed the imprint of a line ℓ, then the algorithm will accept with probability 1 since it rejects only if the 1-pixels it samples and the answers to queries it performs, are not consistent with any line imprint.

It remains to show that an image M that is ϵ-far from all line imprints will be rejected with high constant probability. Let $R(S_1)$ be the minimal rectangle (submatrix) that contains all 1-pixels in the sample S_1. Given the size of S_1, with high constant probability, the number of 1-pixels of M that fall outside of R is at most $(\epsilon/3) \cdot w(M)$. Assume from this point on that this is in fact the case.

As M is ϵ-far from every line imprint, it is, in particular, ϵ-far from all line imprints that intersect R along the sleeve P defined by p_1 and p_2. Thus at least one of the two holds:

1. There are at least $(\epsilon/3) \cdot w(M)$ 1-pixels of M inside R and outside P.

2. For every line imprint that contains p_1 and p_2, there are at least $(\epsilon/3) \cdot w(M)$ pixels in P that disagree with the line imprint.

In the first case, since the size m_2 of the sample S_2 is $\Theta(1/\epsilon)$, with high constant probability (for an appropriate constant in the $\Theta(\cdot)$ notation) S_2 will contains a 1-pixel inside R and outside the sleeve, causing the algorithm to reject. In the second case, by Claim 4.4, every line imprint that contains p_1 and p_2 has distance at least $\epsilon/9$ from M with respect to the uniform distribution over the sleeve P. Applying Claim 4.3 and an argument similar to the one applied in Lemma 4.1, given the setting of m_3 (the size of the query set T), with high constant probability, every line-imprint that contains p_1 and p_2 will be inconsistent with M on at least one query in T, causing the algorithm to reject. Summing up the probabilities of “bad” events, the theorem follows. ■

5 Testing Convexity

As noted in the introduction, we say that an image M is *convex* if there exists a convex shape that is connected, closed and such that all the pixels that the shape intersects in M are 1-pixels, and there are no other 1-pixels in the image. Some work has been done in the field of computational geometry, testing in different models whether a set of points is in convex formation or is far from convex formation (e.g., [RV04, EKK$^+$00, CLM06, CSZ00, CS05]). Generally speaking, the object under inquiry in these papers is a set of points or edges, and not an image. Convexity is also tested for in the dense image model [Ras03] but the difference in models leads to different algorithms and results. We assume without loss of generality that the convex shape is a polygon, and we consider a slightly restricted version of this property where we require the slope of the lines defining the convex shape to be of the form $1/r$ for an integer r. For the sake of succinctness, we refer to this notion as a convex shape. This variant of convexity leads to an alternative definition that builds on the notion of *blocks*, which are defined next.

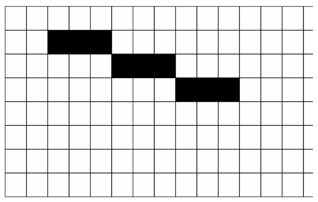

Figure 5: An illustration of a horizontal block of period-length 3.

Definition 5.1 *A* horizontal (vertical) block *(see Figure 5) is a maximal* connected *set of* 1*-pixels that has the following properties:*

1. *There is a single* 1*-pixel in every column (row).*
2. *The number of* 1*-pixels in every row (column) is identical.*
3. *All the* 1*-pixels in a row (column) are connected (in a single interval).*

Each subset of 1*-pixels in a block that all belong to the same row (column) is called a* sub-block*. The number of* 1*-pixels in each sub-block (its length) is considered the* period-length *of the block, and the total number of* 1*-pixels in a block is its* size*. To avoid ambiguity, a block of period-length* 1 *is considered horizontal.*

Thus a block is the imprint of a line beginning and ending at the corner of a pixel where the gradient of the line (either horizontally or vertically) is of the form $1/r$ (for some $r \in [n]$).

Definition 5.2 *A convex image (see Figure 6) is an image that has the following properties:*

1. *The image is connected (as defined in Section 3).*
2. *All the top-most, bottom-most, left-most and right-most* 1*-pixels are (within each set) connected. These are called* extremal 1*-pixels, and the corresponding (*extremal*) sets (blocks) are denoted by T, B, L and R, respectively.*

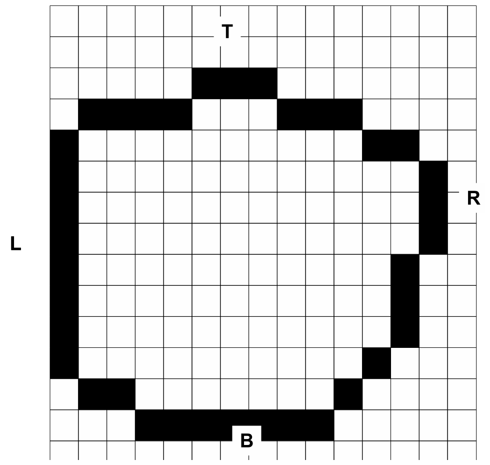

Figure 6: An illustration of a convex shape.

3. *The top-most* 1*-pixel in* L *and the left-most* 1*-pixel in* T *are connected by a series of vertical blocks of monotonically decreasing period-length followed by a series of horizontal blocks of monotonically increasing period-length. For both series, each block is above and to the right of the previous block. The right-most* 1*-pixel in* T *and the top-most* 1*-pixel in* R *are connected by a series of horizontal blocks of monotonically decreasing period-length, followed by a series of vertical blocks of monotonically increasing period-length. For both series, each block is below and to the right of the previous block. The blocks* R *and* B*, as well as* B *and* L*, are connected in a similar manner.*

We note that if the number of 1-pixels in T is at least as large as the sum of the period-lengths of the two adjacent horizontal blocks, then there is more than one way to view the image as an imprint of lines, but this is immaterial to our testing problem. A similar statement holds for B, L and R.

One approach to testing convexity is through learning – if an image is convex, then we can approximately (almost exactly) reconstruct it (where this notion of "almost exact" reconstruction is explained presently) by performing $\tilde{O}(\sqrt{w(M)})$ queries, and taking a sample of size that depends on the quality of the approximation. This is based on three observations (where the first two will also be used by our more efficient algorithm).

The first observation is that given a 1-pixel in a convex image, we can determine (learn) the sub-block it belongs to by performing a binary search (using $O(\log w(M))$ queries, where we double the estimated length of the sub-block in each iteration). The only worry is that in the process of performing this search we may erroneously "reach the other side" of the shape (for an illustration, see Figure 7). However, if the image is convex, then this may occur only in rows/columns adjacent to the extremal blocks T, B, L and R, and we address this issue below.

The second observation is, that by building on the sub-block reconstruction procedure, we can reconstruct the whole block that a 1-pixel belongs to (recall that all sub-blocks in a block have the same length). This again can be done by a binary (doubling) search that uses $O(\log w(M))$ queries. The third observation is that between, e.g., the set (block) L and the set (block) T, there can only be fewer than $2\sqrt{w(M)}$ different horizontal (vertical) blocks, as the sum of their sizes cannot exceed $w(M)$ (and the blocks have different period-lengths).

Therefore, starting from any 1-pixel in a convex shape (and in particular, a uniformly selected 1-pixel), it is possible to approximately reconstruct the shape, by reconstructing block after block. The reason that this reconstruction may be approximate is that two (sub-)blocks from different sides of the image may be relatively close, and we think they are connected. As noted previously this may occur (in the case of a convex image), only in the (sub-)blocks just adjacent to the extremal blocks (so that we effectively "cut off" one/some of them). Thus, after the reconstruction procedure is completed, we take a sample of size $\Theta(1/\gamma)$ of 1-pixels, where γ is a precision parameter. If any 1-pixel in the sample belongs to a row (column) just above/below (to the left/right) of the top-most/bottom-most (left-most/right-most) 1-pixels we have observed, then we learn its sub-block, and modify the image accordingly. Alternatively, we can uniformly query $\Theta(1/\gamma)$ pixels in the four extreme sub-blocks we have found, and if we get an answer of 0, then we "break" the sub-block into two blocks and determine the correct extreme sub-block by performing $O(\log(w(M))$ additional queries. In either case, this ensures with high probability that the image we reconstruct and the actual image differ on $O(\gamma\cdot w(M))$ pixels. Note that *exact* reconstruction in general requires $\Omega(w(M))$ queries.

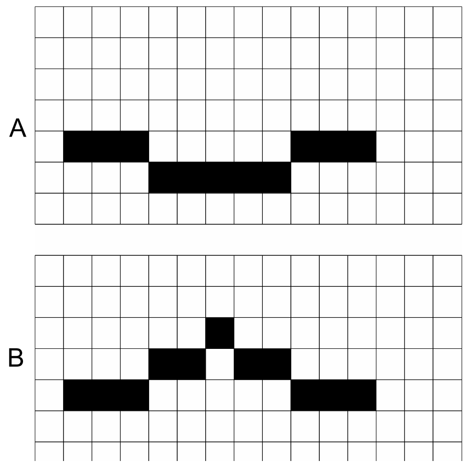

Figure 7: While the lengths of the sub-blocks in (A) can easily be determined by a binary search, the two sub-blocks in (B) that are right below the top block may be considered to be a single sub-block when running the binary search procedure.

The above almost-exact learning/reconstruction procedure can be easily modified to give a (one-sided error) testing algorithm with query complexity $\tilde{O}(\sqrt{w(M)})$ and sample complexity $O(1/\epsilon)$. We now turn to describe a more efficient (though somewhat more involved) algorithm. The query and sample complexities of this algorithm are $\tilde{O}(w(M)^{1/4})\text{poly}(1/\epsilon)$. We assume that the algorithm is given an estimate $\widehat{w}$ such that $w(M)/c \le \widehat{w} \le c\cdot w(M)$ for some constant c, and we later show how to remove this assumption. The high level structure of the algorithm is as follows. The algorithm samples $\tilde{\Theta}(\widehat{w}^{1/4}/\epsilon)$ 1-pixels, and applies the subroutines for learning sub-blocks and blocks (as described above) to learn the blocks they belong to. These blocks and the intervals between them serve as a "backbone". The algorithm then uses additional samples and queries to verify (probabilistically) that pixels in the intervals of the "backbone" exist as expected, and that few 1-pixels fall outside their expected location. A small number of intervals (which we denote "special") can not be checked by the algorithm. It thus verifies that these intervals account for only a small number of the 1-pixels in the image.

Algorithm 4: **Test-Convexity**

1. *Let $\epsilon' = \epsilon/(100c)$ (where c is the constant (known by the algorithm) for which $w(M)/c \leq \widehat{w} \leq c \cdot w(M)$).*

2. *Take a sample S_1 of $\tilde{\Theta}(\widehat{w}^{1/4}/\epsilon)$ uniformly distributed 1-pixels in M. For each of the 1-pixels selected, learn its block (using $O(\log(\widehat{w}))$ queries).*

3. *If the set of pixels sampled and queried is inconsistent with a convex shape having at most $c \cdot \widehat{w}$ 1-pixels, then* REJECT *(recall that $w(M) \leq \widehat{w}/c$).*

4. *Considering blocks in a clockwise order, if there are two blocks that are separated by more than $\epsilon' \cdot \widehat{w}^{3/4}$ columns or more than $\epsilon' \cdot \widehat{w}^{3/4}$ rows, then* REJECT.

5. *Considering blocks in a clockwise order, we define the* area *of the interval between any two consecutive blocks as the set of pixels that can be 1-pixels connecting the two blocks in consistent convex images.*

   If there are pairs of consecutive horizontal (vertical) blocks whose period-lengths differ by more than $\widehat{w}^{1/4}$ and the smaller period length among the two is at most $\widehat{w}^{1/2}$, then mark the interval (area) between these two blocks as "special". Likewise, mark intervals between two blocks where one is horizontal and the other is vertical, or intervals following a block that may be the last sampled block (in clockwise order) before reaching an extremal set as special (there are at most 8 such intervals).

6. *Select a sample S_2 of $\Theta(1/\epsilon)$ blocks adjacent to non-special intervals where the probability of selecting a block is proportional to the number of columns/rows (depending on whether the block is horizontal or vertical) of the block and the interval following it (in a clockwise direction). For each selected block learn the blocks in the interval (area). If in any step an inconsistency is detected, then* REJECT.

7. *For each sequence of blocks learned in Step 6 as well as the preceding block (learned in Step 2) do the following. perform $\Theta(1/\epsilon)$ queries to uniformly selected pixels within each of these sequences. With the exception of the four extreme sub-blocks, if any query is answered by 0, then* REJECT. *If a query within one of the four extreme sub-blocks is answered by 0, then perform a binary search to learn the two sub-blocks it should be broken into and* REJECT *if any inconsistency is found.*

8. *Take a sample S_3 of $\Theta(1/\epsilon)$ 1-pixels uniformly at random. If any 1-pixel in S_3 does not belong to the area of any interval, then* REJECT. *Otherwise, if the number of 1-pixels in S_3 that fall within the areas of special intervals is greater than expected (given the number of rows/columns in special intervals) by more than an $\epsilon' \cdot \widehat{w}$ additive factor, then* REJECT. *Note that for special intervals that are so defined because they may be the last sampled block before reaching an extremal set we expect the set of points in unsampled blocks following them in clockwise order to be of size $O(\widehat{w}^{1/4})$.*

9. For each sampled 1-pixel that isn't in the area of a special interval, learn the sequence of blocks beginning from the block preceding it in the clockwise order. If any inconsistency is detected, then REJECT. *(Note that if in this process we do not learn the block containing the 1-pixel, we will reject).*

10. If no step caused rejection, then ACCEPT

In all that follows, let $\epsilon' = \frac{\epsilon}{100c}$. We first establish the next simple claim.

Claim 5.1 *If the algorithm* **Test-Convexity** *passes Step 3, then there are at most* $8\widehat{w}^{1/4}+8$ *special intervals. Furthermore, if the image is convex, then in each non-special interval there are at most* $\widehat{w}^{1/4}$ *different blocks.*

Proof: Since (by the premise of the claim) the algorithm passes Step 3, the sampled and queried pixels are consistent with a convex shape. Hence, the set of blocks learned contains at most 8 different monotone sequences of blocks. Consider such a sequence. The smallest period-length of a block in the sequence is 1, and each special interval that is not associated with an extremal set is adjacent to a block of period-length at most $\widehat{w}^{1/2}$. But since the period-length of blocks that are adjacent to special intervals differ by at least $\widehat{w}^{1/4}$, the number of special intervals in the sequence can be at most $\widehat{w}^{1/4}$. Adding the at most 8 special blocks that do not belong to the monotone sequences, we get the bound in the claim.

Turning to the second part of the claim, for non-special intervals that neighbor blocks of period-length at most $\widehat{w}^{1/2}$, by the definition of special intervals, these (non-special) intervals contain at most $\widehat{w}^{1/4}$ different blocks. Next observe that if the two blocks defining a non-special interval have period length more than $\widehat{w}^{1/2}$, then the number of sub-blocks (and hence blocks) between them is at most $\widehat{w}^{1/4}$ (given that the image is consistent with a convex image and there are at most $\epsilon'\widehat{w}^{3/4}$ columns and rows in the interval). This immediately gives a bound on the number of different blocks in these intervals, and the claim follows. ■

We are now ready to state and prove our main theorem concerning testing convexity.

Theorem 5.1 *Given an estimate* $\widehat{w}$ *such that* $w(M)/c \leq \widehat{w} \leq c \cdot w(M)$, *the algorithm* **Test-Convexity** *has the following properties:*

1. *Completeness: When given access to a convex image, the algorithm accepts with probability at least* $2/3$.

2. *Soundness: When given access to an image that is* ϵ*-far from any convex image, the algorithm rejects with probability at least* $2/3$.

3. *The algorithm takes* $\tilde{O}(\widehat{w}^{1/4}/\epsilon)$ *samples and performs* $\tilde{O}(\widehat{w}^{1/4}/\epsilon^2)$ *queries. Its running time is* $\tilde{O}(\widehat{w}^{1/4}/\epsilon^2)$ *as well.*

Proof: The bounds on the sample and query complexities follow directly from the description of the algorithm and the second part of Claim 5.1 (which bounds the number of queries performed in Steps 6 and 9). Its running time is at most a polylogarithmic factor larger than its sample and query complexities. Hence, we turn to proving completeness and soundness.

Completeness. Let M be a convex image. By the premise of the theorem concerning the estimate $\widehat{w}$, the image M will not be rejected in Steps 3 of the algorithm. The image will also not be rejected in Steps 3, 6, 7, and 9, because all pixels queried and sampled are consistent with M, and M is convex.

We next bound the probability of rejection in Step 4. Now consider a partition of the 1-pixels of M to consecutive (connected) subsets of size $\epsilon'\widehat{w}^{3/4}/2$ each. By an appropriate choice of the constant in the $\Theta(\cdot)$ notation for the size of S_1, the probability that there exists a subset that does not include any sample 1-pixel from S_1 is at most 1/6. Thus, the probability that the algorithm rejects in Step 4 is at most 1/6.

Turning to Step 8, recall that by Claim 5.1, there are at most $8\widehat{w}^{1/4}+8$ special intervals. Since each should contain at most $\epsilon'\widehat{w}^{3/4}$ pixels (or $2\epsilon'\widehat{w}^{3/4}$ for the extreme intervals), the total number of 1-pixels that belong to the areas of special intervals should be at most $\epsilon'\widehat{w} \leq c\epsilon' w(M)$. This implies (using a multiplicative Chernoff bound) that for an appropriate constant in the $\Theta(\cdot)$ notation for the size of S_3, the probability that the estimate of the number of 1-pixels that belong to special intervals exceeds this upper bound by more than $\epsilon'\widehat{w}$ is at most 1/6. Therefore, the algorithm rejects in Step 8 with probability at most 1/6. Having established that the algorithm may reject only in Step 4 or in Step 8, and in each with probability at most 1/6, by a simple union bound we have that it rejects with probability at most 1/3, and completeness is proved.

Soundness. Let M be an image that is ϵ-far from any convex image. We will show that it is rejected with probability at least 2/3. If M is rejected by the end of Step 4 then we are done. Otherwise, M is in particular far from every convex image that is consistent with the blocks constructed based on the sample S_1. For a fixed sample S_1 (that didn't cause rejection), and the blocks learned from each sampled 1-pixel in S_1, let $\mathcal{C}(S_1)$ denote the set of convex images that are consistent with the learned blocks. If M is ϵ-far from being convex, then in particular it is at least ϵ-far from every image in $\mathcal{C}(S_1)$. We next consider several cases that together cover all cases in which there is such a distance to all images in $\mathcal{C}(S_1)$.

1. In the first case, the number of 1-pixels that do not belong to the areas of intervals that are determined by the learned blocks is at least $(\epsilon/4)w(M)$. In this case the algorithm will reject with high constant probability in Step 8. Therefore, assume from this point on that this is not the case. It follows that M differs from every image in $\mathcal{C}(S_1)$ on at least $3\epsilon/4$ pixels that are within the areas of the intervals.

2. In the second case, there are at least $(\epsilon/8)w(M)$ 1-pixels in M that belong to areas of special intervals. In this case, since, as noted previously, the number of 1-pixels in special intervals must be much smaller (at most $9\epsilon'\widehat{w} \leq \epsilon w(M)/10$), this will be detected with high constant probability in Step 8, and the algorithm will reject. Therefore, assume from this point on that this is not the case. It follows that M differs from every image in $\mathcal{C}(S_1)$ on at least $(\epsilon/2)w(M)$ pixels within the areas of non-special intervals (since by removing at most all 1-pixels in M from the areas of the special intervals and adding less than this number of 1-pixels, M can be made consistent within these areas with an image in $\mathcal{C}(S_1)$).

3. To describe the third case, consider any non-special interval between blocks B_i and B_{i+1} (that were learned in Step 2), where from this point on we consider B_i as part of the interval. We shall say that this interval is *good* if the process of learning the sequence of blocks in

this interval would succeed if started from block B_i, and the fraction of missing pixels in this sequence is at most $\epsilon/(8c^2)$. Otherwise it is *bad*.

If the fraction of bad intervals is at least $\epsilon/(8c^2)$, then with high constant probability, at least one such interval is selected in Step 6. Conditioned on this event, either the algorithm rejects in this step, or it rejects with high constant probability in Step 7. Thus, assume from this point on that the fraction of bad intervals is at most $\epsilon/(8c^2)$.

Recall that each interval contains at most $\epsilon' \cdot \widehat{w}^{3/4}$ rows and columns, and that the blocks learned in Step 2 are consistent with a convex image that has at most $c \cdot \widehat{w}$ pixels. Therefore, by adding at most $(\epsilon/(4c^2)) \cdot c \cdot \widehat{w} \leq (\epsilon/4) \cdot w(M)$ 1-pixels to M we can obtain an image M' whose set of 1-pixels within non-special intervals *contains* the set of 1-pixels in these intervals from an image in $\mathcal{C}(S_1)$. It follows that M must contain at least $(\epsilon/4) \cdot w(M)$ 1-pixels in non-special images that are not in M'.

4. It remains to deal with the final case described above. Since M contains at least $(\epsilon/4) \cdot w(M)$ 1-pixels that are not in M', with high constant probability at least one such pixel will be selected in Step 8. Conditioned on this event occurring, the algorithm rejects in Step 9 either because the process of learning a sequence of blocks fails, or because the selected pixel falls outside the sequence of blocks learned.

It thus follows that if M is ϵ-far from being convex then it will be rejected with high constant probability (due to one of the reasons described above). ■

5.1 Estimating $w(M)$

In this subsection we describe a procedure for estimating $w(M)$ to within a multiplicative constant c under the assumption that M is a convex image. Using the estimate, we can then run Algorithm 4. We note that the estimation procedure may reject the image if it finds evidence that the image is not convex. This evidence may be statistical evidence, so that it is possible that a convex image is rejected (with small probability). However, Algorithm 4 has two-sided error as well and so the error probability introduced by the estimation algorithm is simply added to the one introduced by Algorithm 4 (where, as usual, we can decrease these probabilities (exponentially) by repetitions (or directly, by increasing the sample and query complexities)).

The estimation procedure starts by taking a sufficiently large constant-size sample, and considering the distance between the furthest pair of points. If M is convex, then this value, denoted $\widetilde{w}$, is at most $w(M)$ and, with high constant probability, is at least $w(M)/8$. However, if the image is not convex, then $\widetilde{w}(M)$ could differ significantly from $w(M)$. In particular, it could be much larger than $w(M)$ so that if we set $\widehat{w} = \widetilde{w}$ and run Algorithm 4 with this estimate, then the resulting complexity of Algorithm 4 might be much larger than $w(M)^{1/4}$. Therefore, we run a procedure that tries to verify if indeed $w(M)$ is of the same order as $\widetilde{w}$. Our starting point is the next simple claim.

Claim 5.2 *Let M be a convex image. At least $w(M)/2 - \sqrt{w(M)}$ of the 1-pixels in M are in blocks of size at least $\sqrt{w(M)}/4$ 1-pixels (each).*

Proof: Since all blocks with period-length greater than $\sqrt{w(M)}/4$ must trivially contain at least $\sqrt{w(M)}/4$ 1-pixels, we consider the blocks of period-length less than $\sqrt{w(M)}/4$. With the possible exception of the four extreme blocks, for every period-length in $\{1, \ldots, (\sqrt{w(M)}/4) - 1\}$ there can be at most 8 blocks, and by definition each includes at most $\sqrt{w(M)}/4$ 1-pixels. Thus, the total number of pixels in such blocks cannot exceed $8 \cdot \sqrt{w(M)}/4 \cdot \sqrt{w(M)}/4 = w(M)/2$. Adding the at most $4 \cdot \sqrt{w(M)}/4$ pixels that belong to the extreme blocks (if they have size at most $\sqrt{w(M)}/4$), we get $w(M)/2 + \sqrt{w(M)}$. It follows that all remaining at least $w(M)/2 - \sqrt{w(M)}$ pixels belong to blocks that contain (each) at least $\sqrt{w(M)}/4$ 1-pixels. ■

It follows from Claim 5.2 that a sampled 1-pixel will be in a block containing at least $\sqrt{w(M)}/4$ 1-pixels with probability very close to 1/2. Thus, we first take a constant size sample of 1-pixels, and for each 1-pixel selected, we learn the block it belongs to. In order to ensure that in this process we perform only $O(\log w(M))$ queries even if the image is not convex (and in particular not connected) we make the following small modification to the procedure. Whenever our estimate of the size of the block is increased by a factor of 2, we test, by performing a constant number of queries, whether the part of the block learned so far is (close to being) full. In performing this test we need not consider the cumulative probability of error, as even if we err on a small constant fraction of these tests, the total number of 1-pixels in M that belong to the block learned so far, is still high.

If either any of the learning processes fail (since they find evidence that the image is not convex), or less than a 1/4 of the selected pixels belong to blocks of size at least $\sqrt{\widetilde{w}}/32$, then we reject. If M is a convex image, then conditioned on $\widetilde{w} \geq w(M)/8$ (which holds with high constant probability), by Claim 5.2 (and a multiplicative Chernoff bound), the probability that we reject at this stage is a small constant. On the other hand, if the image is not convex, then we have high confidence that $w(M) = \Omega(\sqrt{\widetilde{w}})$ (since we have learned blocks of at least this size and verified that they are close to being full), implying that $\log(\widetilde{w}) = O(\log(w(M)))$. This means that we can learn, using $O(\log(w(M))$ queries the size of any block in which we sample a 1-pixel, unless the size of the block is far greater than $\widetilde{w}$, in which case we may reject. Once again, if the image is convex, then conditioned on $\widetilde{w} \geq w(M)/8$, we won't reject. From this point on we shall assume that this is in fact the case.

Throughout the whole process we next describe we will reject if we encounter evidence that M is not a convex image. We may thus discuss "learning the block of a given 1-pixel" – this means learning the block the pixel belongs to assuming that M is convex, and verifying that the block is (almost) full.

We shall say that a block is *large* it its size is at least $\sqrt{\widetilde{w}}/32$ (where in the case of a convex image this is at least $\sqrt{w(M)}/4$). Now that we have the ability to check the size of a block we hit with a random sample of a 1-pixel, we will attempt to estimate the number of large blocks in M. If this number is as large as expected and distributed as we expected, then we will accept, and otherwise we will reject. Consider partitioning the large blocks into buckets according to their size. The bucket T_i will contain the blocks of size at least $\sqrt{\widetilde{w}}2^{i-5}$ and no more than $\sqrt{\widetilde{w}}2^{i-4}$. Thus the number of buckets is $O(\log(\widetilde{w}))$, and by Claim 5.2, if the image is convex, then they contain close to a half of all the 1-pixels in the image.

We can obtain an estimate to the *fraction* of 1-pixels (out of $w(M)$) that fall into each bucket T_i to within an additive term of $1/(c' \log(\widetilde{w}))$ (where c' is a sufficiently large constant) for each bucket by taking a sample of $\tilde{\Theta}(\log(\widetilde{w})$ 1-pixels, and for each, learning its block size. From this point on we consider only buckets for which we have a sufficiently large sample of pixels, and we refer to them as *significant buckets*. With high probability, the total number of 1-pixels that belong to the other (non-significant) buckets is at most a small constant fraction of $w(M)$. Since within each bucket all blocks have approximately the same size, the estimated fraction of 1-pixels in each significant bucket, together with the estimate $\widetilde{w}$ of $w(M)$, give us an estimate of the number of blocks that should belong to each bucket. Since the blocks in the buckets are of size $\Omega(\sqrt{\widetilde{W}})$, there should be $O(\sqrt{\widetilde{W}})$ blocks in each bucket.

Given the above, we verify that the number of blocks in each significant bucket is indeed as it should be, where we rely on *collision probabilities*, similarly to what was done in the context of testing connectivity. However, while in the case of obtaining an estimate for $w(M)$ in the context of testing connectivity we used collisions between single 1-pixels (implying complexity $O(\sqrt{w(M)})$), here we build on collisions between blocks.

Thus, we do the following. Let b_i denote our estimate of the number of blocks in (significant) bucket T_i. We begin sampling 1-pixels uniformly at random, and for each pixel selected, we learn the block it belongs to (and hence the bucket to which this block belongs to). For each such block we will verify with high probability that it contains all the 1-pixels we would expect. If we sample more than one pixel from the same block in a bucket before we expect to (e.g., before having sampled at least $\sqrt{b_i}/c''$ 1-pixels in this bucket, for some sufficiently large constant c''), we take this as an indication that this bucket does not contain the number of 1-pixels that it should (given the estimate $\widetilde{w}$), and we reject. Otherwise, we continue sampling until we've sampled $\tilde{\Theta}(\widetilde{w}^{1/4})$ 1-pixels. If the number of collisions within buckets is significantly far from its expected value, then we reject, otherwise we use $\widetilde{w}$ as an estimate for $w(M)$. The total number of samples taken and queries performed is $\tilde{\Theta}(w(M)^{1/4})$, are required.

6 Testing Monotonicity

We shall say that a matrix M is *monotone* if for every two entries (i_1, j_1) and (i_2, j_2) for which $M[i_1, j_1] = 1$ and $M[i_2, j_2] = 1$, if $i_1 < i_2$ then $j_1 \le j_2$. In all that follows we assume that the algorithm has a constant factor estimate $\widehat{w}$ of $w(M)$. As shown in Subsection 3.1.2, it is possible to obtain such an estimate by running a procedure that has sample and query complexity $\tilde{O}\left(\min\left\{\sqrt{w(M)}, \frac{n^2}{w(M)}\right\}\right)$. It is an open problem whether it is possible to obtain such an estimate with a procedure that has complexity $o(\sqrt{w(M)})$ (by exploiting the fact that the image should be monotone).

We begin by observing that there is a simple one-sided error algorithm that uses only sampling (that is, only uniformly distributed 1-pixels), whose sample complexity is $O((\widehat{w}/\epsilon)^{1/2})$ (and whose running time is $\tilde{O}((\widehat{w}/\epsilon)^{1/2})$). This algorithm simply takes a sample of $\Theta((\widehat{w}/\epsilon)^{1/2})$ 1-pixels and rejects if and only if it gets a pair that violates monotonicity (that is, (i_1, j_1) and (i_2, j_2) such that $i_1 < i_2$ but $j_1 > j_2$). Clearly, a monotone matrix is never rejected.

In order to prove that if M is ϵ-far from monotone then the algorithm will reject with high constant probability, we consider the following *violation graph*[8] $G_{\mathrm{viol}}(M) = (V(M), E_{\mathrm{viol}}(M))$ where $V(M) = \{(i,j) : M[i,j] = 1\}$ and $E_{\mathrm{viol}}(M)$ consists of all pairs of vertices that correspond to violating pairs in M. By the definition of $G_{\mathrm{viol}}(M)$, the distance of M to being monotone is just the size of the minimum vertex cover of $G_{\mathrm{viol}}(M)$ divided by $w(M)$. This implies that if M is ϵ-far from

[8]The notion of a violation graph in the context of monotonicity was used previously for testing monotonicity of functions (see e.g. [DGL+99]).

being monotone, then there exist a matching in $G_{\text{viol}}(M)$ of size at least $\epsilon|V(M)|/2 = \epsilon w(M)/2$. Therefore, which high constant probability, a sample of size $\Theta((\widehat{w}/\epsilon)^{1/2}) = \Theta((w(M)/\epsilon)^{1/2})$ will contain a violating pair, causing the algorithm to reject as required.

We next give an algorithm whose complexity is $\tilde{O}(n^{2/3}/(\epsilon^2 w(M)^{1/3}))$, which improves on the simple sampling algorithm (in terms of the dependence on n) when $w(M) = \Omega(n^{4/5})$. We then give an almost matching lower bound of $\Omega(\min\{w(M)^{1/2}, n^{2/3}/w(M)^{1/3}\})$ which holds for $w(M) = O(n)$ (and holds when the algorithm has a constant factor estimate of $w(M)$).

6.1 The Algorithm

Indeed, by sampling alone, we cannot reduce the complexity below $\Theta(w(M)^{1/2})$) for any $w(M) = O(n)$. We give the argument for any one-sided error algorithm and $w(M) = \Theta(n)$. It is not hard to extend the argument to get a lower bound for two-sided error algorithms and any $w(M) = O(n)$. These bounds hold when the algorithm has a constant factor estimate of $w(M)$. Suppose that the 1-pixels in the matrix reside on two diagonals: the main diagonal (that is, all pixels (i,i) for $1 \leq i \leq n$) and the third diagonal (that is, all pixels $(i, i+2)$ for $1 \leq i \leq n-2$). This matrix is $(1/2)$-far from being monotone (or, more precisely $(1/2 - O(1/n))$)-far from being monotone). However, unless the sample contains a pair of 1-pixels of the form $(i, i+2)$ and $(i+1, i+1)$, then no violation is observed, and any one-sided error algorithm must accept. The probability that such a pair appears in a sample of size $o(n^{1/2})$ is $o(1)$.

Therefore, in order to reduce the dependence on n (for $w(M) = \Omega(n^{4/5})$), and in particular reduce the dependence on n from $n^{1/2}$ to $n^{1/3}$ when $w(M) = \Theta(n)$, we shall perform queries in addition to sampling. Roughly speaking, by sampling we try to detect violations that occur at relatively large distances, and by performing queries we try to detect violations that occur at relatively small distances (such as those in the above lower-bound construction).

The algorithm first takes a sample that with high probability either contains evidence that the matrix is not monotone (in a form of a violating ("distant") pair), or it (the sample) can be used to determine a set of submatrices with the following properties. First, the fraction of 1-pixels that reside outside the submatrices is relatively small. Second, there can be no violations between pairs of 1-pixels that reside in two different submatrices. Therefore, if the matrix is far from being monotone, then the violations are within the submatrices. In the second stage of the algorithm we take an additional (small) sample, and for each sampled 1-pixel we perform queries within its submatrix (at varying distances from the selected 1-pixel) in order to detect violations with the sampled 1-pixels.

For the sake of simplicity (since the algorithm and its analysis include quite a lot of details), we assume that the algorithm is given $w(M)$. The algorithm and its analysis can be adapted to work with an estimate $\widehat{w}$ such that $w(M)/c \leq \widehat{w} \leq c \cdot w(M)$. We add a few more words about this issue following the algorithm.

The following notations will be used in the algorithm and its analysis.

- $g_1(n, w(M)) \stackrel{\text{def}}{=} n^{2/3}/w(M)^{1/3}$;
- $g_2(n, w(M)) \stackrel{\text{def}}{=} w(M)^{2/3}/n^{1/3}$;
- $g_3(n, w(M)) \stackrel{\text{def}}{=} w(M)^{4/3}/n^{2/3}$; and

- $g_4(n, w(M)) \overset{\text{def}}{=} n^{1/3}w(M)^{1/3}$.

For the sake of succinctness we shall use the shorthand g_i for $g_i(n, w(M))$ (where $i = 1, \ldots, 4$). For example, when $w(M) = n$ we have that $g_1 = g_2 = n^{1/3}$ and $g_3 = g_4 = n^{2/3}$. In general g_3 is just $(g_2)^2$, and we also have that $g_1 \cdot g_3 = w(M)$, $g_1 \cdot g_4 = n$, and $g_3/g_4 = w(M)/n$.

Algorithm 5: Testing monotonicity

1. *Take a sample S_1 of $t_1 = \Theta(g_1 \log n/\epsilon^2)$ 1-pixels. S_1 contains a violating pair, then* REJECT, *otherwise, continue.*
2. *Take a sample S_2 of $t_2 = \Theta(1/\epsilon)$ 1-pixels. If there is a violating pair in $S_1 \cup S_2$, then* REJECT. *Otherwise, for each of the 1-pixels (a, b) in S_2 perform the following sub-test:*
   - *For $\ell = 1$ to g_2, where ℓ increases by a multiplicative factor of 2 in each iteration, uniformly select $t_3(\ell) = \Theta(\ell\cdot(n/w(M))\cdot\log(n)/\epsilon^2)$ entries in the submatrix of dimensions $\ell \times \ell$ that (a, b) is the bottom-right corner of, and similarly for the $\ell \times \ell$ submatrix that (a, b) is the top-left corner of, and perform queries on all these pixels. If any is answered by '1' then* REJECT.
3. *If no step caused rejected, then* accept.

If the algorithm is only given an estimate $\widehat{w}$ of $w(M)$ that is within a constant multiplicative factor c away from $w(M)$ then we simply replace $g_1 = g_1(n, w(M))$ by $g_1(n, \widehat{w}/c)$ (so that t_1 is increased by a constant factor), we replace $g_2 = g_1(n, w(M))$ by $g_2(n, c\cdot w(M))$ (so that the maximum value of ℓ is increased by a constant factor), and in $t_3(\ell)$ we replace $w(M)$ by $\widehat{w}/c$ (so that $t_3(\ell)$ is increased by a constant factor). This increases the complexity of the algorithm by a constant factor, but the proof of correctness follows from Theorem 6.1 given the assumption on $\widehat{w}$ and the fact that the algorithm has one-sided error.

Theorem 6.1 *Algorithm 5 is a one-sided error algorithm for monotonicity of matrices. Its sample and query complexity as well as its running time are $\tilde{O}(n^{2/3}/(\epsilon^2 w(M)^{1/3}))$.*

Since the algorithm only rejects when it has evidence of violation of monotonicity, it clearly has one-sided error. The sample complexity of the algorithm is $t_1 + t_2 = O(g_1 \log n/\epsilon^2) = \tilde{O}\left(n^{2/3}/(\epsilon^2 w(M)^{1/3}\right)$, and the query complexity is of the order of $\log(n) \cdot g_2 \cdot (n/w(M)) \cdot \log(n)/\epsilon^2$ which is $\tilde{O}\left(n^{2/3}/(\epsilon^2 w(M)^{1/3})\right)$ as well. The running time is at most a factor of $\log(n)$ larger. Thus it remains to prove that if M is ϵ-far from being monotone, then it is rejected with high constant probability.

For the sake of the analysis, it will be convenient to partition the first sample, S_1, into two subsamples, S_1^1 and S_1^2, where S_1^1 is of size $\Theta(g_1/\epsilon)$, and S_1^2 is of size $\Theta(g_1 \log n/\epsilon^2)$. We also introduce the following two definitions.

Definition 6.1 *For a subset of 1-pixels S that obey monotonicity, we can sort the 1-pixels in S in a monotonically increasing order: $(i_1, j_1), \ldots, (i_{|S|}, j_{|S|})$, so that every two consecutive 1-pixels (i_r, j_r) and (i_{r+1}, j_{r+1}) define a submatrix, M_r, where (i_r, j_r) is the bottom-left corner of the submatrix* *and (i_{r+1}, j_{r+1}) is the top-right corner. We note that these submatrices may have width* 1 *(if $i_r = i_{r+1}$) or height* 1 *(if $j_r = j_{r+1}$). If we also add the submatrix determined by* $(1,1)$ *and* (i_1, j_1) *(assuming that $(i_1, j_1) \neq (1,1)$) and the submatrix determined by $(i_{|S|}, j_{|S|})$ and (n,n) assuming that $((i_{|S|}, j_{|S|}) \neq (n,n)$, then we denote the resulting set of submatrices by $\mathcal{M}(S)$.*

Definition 6.2 *We say that a submatrix T is* heavy *if it contains more than g_3 $(= (w(M))^{4/3}/n^{2/3})$* 1*-pixels.*

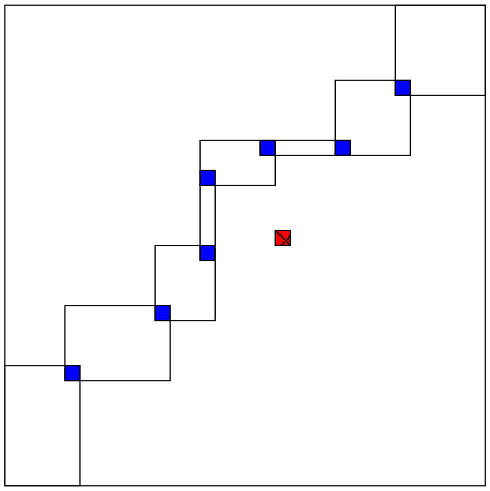

Figure 8: An illustration for the partition into submatrices that is induced by a set of monotonically increasing 1-pixels, and a violating 1-pixel with respect to this set.

We start by establishing the following lemma.

Lemma 6.1 *With high constant probability over the choice of the sample S_1^1, either S_1^1 contains a violating pair or there exists a subset of S_1^1 of size at most $6g_1$ $(= 6n^{2/3}/w(M)^{1/3})$, denoted $\widetilde{S}_1^1$ such that at most an $(\epsilon/16)$-fraction of the* 1*-pixels in M belong to heavy submatrices in $\mathcal{M}(\widetilde{S}_1^1)$.*

In order to prove Lemma 6.1 we introduce one more notion and a simple claim.

Definition 6.3 *For a submatrix T, whose bottom-left corner is (x,y) and whose top-right corner is (x',y'), we say that a* 1*-pixel (a,b) in T is a* separating 1*-pixel (with respect to T), if both the submatrix determined by (x,y) and (a,b) and the submatrix determined by (a,b) and (x',y') contain at most $2/3$ of the* 1*-pixels in T.*

Note that in the above definition, we do not care how many 1-pixels belong to the two remaining submatrices of T (these are 1-pixels that are violating with respect to (a,b)).

Claim 6.2 *For any (non-empty) submatrix T, at least $1/3$ of the* 1*-pixels in T are separating* 1*-pixels (with respect to T).*

Proof: Consider ordering the 1-pixels in T according to their lexicographical order. That is, we have $(a_1, b_1) = (x,y), (a_2, b_2), \ldots, (a_q, b_q) = (x', y')$, where $a_r \leq a_{r+1}$ and if $a_r = a_{r+1}$ then

$b_r < b_{r+1}$. Then each of the 1-pixels that are in the middle third according to this order are separating 1-pixels. ■

Proof of Lemma 6.1: The proof of the lemma is based on the fact that as long as an $\Omega(\epsilon)$-portion of the 1-pixels belong to heavy submatrices, we are $\Omega(\epsilon)$-likely to select a separating 1-pixel for one of these submatrices. After $6g_1$ such splits we show that the fraction of 1-pixels that belong to heavy submatrices in $\mathcal{M}(\widetilde{S}_1^1)$ must go below $\epsilon/16$. Details follow.

Consider selecting the 1-pixels in S_1^1 one after the other. We denote the r^{th} selected 1-pixel by (x_r, y_r) and each subset $\{(x_1, y_1), \ldots, (x_r, y_r)\}$ by $S_1^1(r)$. We shall say that (x_r, y_r) is *violating* (with respect to the previously selected 1-pixels $S_1^1(r-1)$) if there exists a 1-pixel $(x_{r'}, y_{r'}) \in S_1^1(r-1)$ such that $(x_{r'}, y_{r'})$ and (x_r, y_r) constitute a violating pair.

As long as no violating 1-pixel is selected, we construct the set $\widetilde{S}_1^1 \subset S_1^1$ iteratively, starting from $\widetilde{S}_1^1 = \emptyset$. Let us say that an iteration r (in which a new 1-pixel is selected) is a *successful* iteration, if (x_r, y_r) is either a violating 1-pixel with respect to $S_1^1(r-1)$, or it is a separating 1-pixel with respect to some heavy $T \in \mathcal{M}(\widetilde{S}_1^1)$. In the latter case we add the 1-pixel to $\widetilde{S}_1^1$. Given the definition of heavy matrices and separating 1-pixels, we show that if there is no successful iteration in which we get a violating 1-pixel, then after at most $6g_1$ successful iterations, the fraction of 1-pixels that belong to heavy submatrices in $\mathcal{M}(\widetilde{S}_1^1)$ must go below $\epsilon/16$.

To verify this, consider the binary "tree of submatrices" that is defined by a sequence of successful iterations that do not include a violating 1-pixel. That is, the root of the tree is M, and for each successful iteration, if the new 1-pixel (which is a separating 1-pixel) belongs to the submatrix T, then the two children of T in the tree, are the two submatrices that the new 1-pixel defines with the (bottom-left and top-right) corners of T. We claim that the number of leaves in this tree is upper bounded by $3g_1$, implying that $|\widetilde{S}_1^1| \leq 6g_1$.

We shall prove this by assigning each leaf in the tree at least $g_3/3$ $(= w(M)^{4/3}/3n^{2/3})$ distinct 1-pixels in M. Consider any leaf in the tree. By the definition of the tree, its parent submatrix is a heavy submatrix, that is, it contains at least g_3 1-pixels. If the sibling of this leaf is also a leaf, then we assign half of the 1-pixels of the parent submatrix to each. Otherwise (the sibling is not a leaf), we assign the leaf all 1-pixels in its parent submatrix that do not belong to its sibling submatrix. By the definition of separating 1-pixels, it is assigned at least $g_3/3$ 1-pixels. Thus the number of leaves in the tree is at most

$$\frac{w(M)}{g_3/3} = \frac{3w(M)}{w(M)^{4/3}/n^{2/3}} = \frac{3n^{2/3}}{w(M)^{1/3}} = 3g_1 \ . \tag{1}$$

Finally, since as long as the fraction of 1-pixels that belong to heavy submatrices with respect to $\mathcal{M}(\widetilde{S}_1^1)$ is at least $\epsilon/16$, and among them at least a third are separating 1-pixels, the probability that an iteration is successful is $\Omega(\epsilon)$. It follows that with high constant probability, after cg_1/ϵ iterations (where c is a sufficiently large constant), either we observe a violation or the fraction of 1-pixels that belong to heavy submatrices (with respect to $\mathcal{M}(\widetilde{S}_1^1)$), is at most $\epsilon/16$. ■

Assuming that the sample S_1^1 does not cause rejection, we next make the following simple observation (based on the fact that the number of 1-pixels in S_1^2 is $\Omega(1/\epsilon)$).

Claim 6.3 *For any fixed choice of $\widetilde{S}_1^1 \subseteq S_1^1$, if the fraction of 1-pixels that do not belong to any submatrix in $\mathcal{M}(\widetilde{S}_1^1)$ is at least $\epsilon/16$, then with high probability, at least one such 1-pixel is selected in S_1^2 (causing the algorithm to reject).*

Let M be a matrix that is ϵ-far from being monotone. By Lemma 6.1 and Claim 6.3, with high constant probability over the choice of the samples S_1^1 and S_1^2, either M is rejected because S_1^1 contains a violating pair, or it is rejected because there is a violating pair in $S_1^1 \times S_1^2$, or the following holds:

- The sample S_1^1 contains a subset $\widetilde{S}_1^1$ of size at most $6g_1$ such that the fraction of 1-pixels that belong to heavy submatrices in $\mathcal{M}(\widetilde{S}_1^1)$ is at most $\epsilon/16$, and the fraction of 1-pixels outside of the submatrices in $\mathcal{M}(\widetilde{S}_1^1)$ is at most $\epsilon/16$.

If the former holds (that is, there is a violating pair either in S_1^1 or in $S_1^1 \times S_1^2$), then we are done. Thus assume from this point on that the latter holds (where we take into account the small constant probability that neither holds).

Since M is ϵ-far from being monotone, $G_{\rm viol}(M)$ contains a vertex cover of size at least $(\epsilon/2)w(M)$. Given the bound of at most $\epsilon/8$ on the fraction of 1-pixels that do not belong to any submatrix in $\mathcal{M}(\widetilde{S}_1^1)$ or that belong to a heavy submatrix in $\mathcal{M}(\widetilde{S}_1^1)$, we have the following. Let $G^1_{\rm viol}(M)$ be the subgraph of $G_{\rm viol}(M)$ that is induced by vertices that correspond to the 1-pixels in M that belong to non-heavy submatrices in $\mathcal{M}(\widetilde{S}_1^1)$. Then $|V(G^1_{\rm viol}(M))| \geq (1-\epsilon/8)w(M)$, and the size of a minimum vertex cover of $G^1_{\rm viol}(M)$ is at least $\frac{3\epsilon}{8}w(M)$.

Suppose that we further consider the subgraph of $G^1_{\rm viol}(M)$, denoted $G^2_{\rm viol}(M)$, that is induced by the vertices in $G^1_{\rm viol}(M)$ that correspond to 1-pixels belonging to matrices in $\mathcal{M}(\widetilde{S}_1^1)$ whose length in each dimension is at most $16g_4/\epsilon$ $(= 16n^{1/3}w(M)^{1/3}/\epsilon)$. Observe that the number of matrices that have a larger length in one of these dimensions is at most

$$2 \cdot \frac{n}{16g_4/\epsilon} = \frac{\epsilon n}{8n^{1/3}w(M)^{1/3}} = \frac{\epsilon n^{2/3}}{8w(M)^{1/3}}\ , \tag{2}$$

and each contains at most $g_4 = w(M)^{4/3}/n^{2/3}$ 1-pixels. Therefore, $|V(G^2_{\rm viol}(M))| \geq (1-\epsilon/4)w(M)$, and the size of a minimum vertex cover in $G^2_{\rm viol}(M)$ is at least $\frac{\epsilon}{4}w(M)$.

We perform one more step of this form. For each submatrix T of $\mathcal{M}(\widetilde{S}_1^1)$, consider the size of a minimum vertex cover of the subgraph induced by the vertices that correspond to 1-pixels in T. We shall say that the submatrix is *significant* if this size is at least $\epsilon g_3/15$ $(=\frac{\epsilon}{15}w(M)^{4/3}/n^{2/3})$. Let $G^3_{\rm viol}(M)$ be the subgraph of $G^2_{\rm viol}(M)$ that is induced by the vertices of $G^2_{\rm viol}(M)$ that correspond to 1-pixels belonging to significant submatrices. Since there are no edges in $G_{\rm viol}(M)$ (and hence in $G^3_{\rm viol}(M)$) between vertices that correspond to 1-pixels that belong to different submatrices in $\mathcal{M}(\widetilde{S}_1^1)$, and the total number of submatrices in $\mathcal{M}(\widetilde{S}_1^1)$ is at most $6g_1 + 2 = 6n^{2/3}/w(M)^{1/3}) + 2$, we have that $|V(G^3_{\rm viol}(M))| > (1-\epsilon)w(M)$.

Thus $G^3_{\rm viol}(M)$ is the disjoint union of subgraphs that each is induced by a subset of 1-pixels (vertices) that reside in a submatrix of size at most $16g_4/\epsilon$ $(= 16n^{1/3}w^{1/3}/\epsilon)$, where the size of a minimum vertex cover of each submatrix (subgraph) is at least $\epsilon g_3/15$ $(= \epsilon w(M)^{4/3}/(15n^{2/3}))$. The next lemma will allow us to complete our analysis (where we refer interchangeably to 1-pixels in a submatrix and the vertices they correspond to in the underlying violation graph) .

Lemma 6.4 *Consider a submatrix whose length in each dimension is at most s. Suppose that the size of the minimum vertex-cover for this submatrix is at least k. Then for at least $k/2$ of the 1-pixels in the submatrix, there exists for each some $\ell = s/2^t$, such that the number of neighbors* *that the* 1*-pixel has in an* $\ell \times \ell$ *submatrix that it is either the bottom-right or the top-left corner of, is at least* $\frac{\ell\cdot(k/s)}{8\log n}$.

We prove the lemma momentarily, but first show how to apply it in order to show that conditioned on the abovementioned properties of $G^3_{\rm viol}(M)$ (which hold with high constant probability), M is rejected either due to a violating pair in S^2_1, or due to a violating pair that is found in the last step of the algorithm (in which queries are performed). Setting $s = 16g_4/\epsilon$ and $k = \epsilon g_3/15$, in each significant submatrix there are either at least $k/4$ 1-pixels for which the statement in the lemma holds for $\ell \leq g_2$, or there are at least $k/4$ 1-pixels for which the statement in the lemma holds for $\ell > g_2$. We refer to the former type of 1-pixels as *closely useful* and to that latter type of 1-pixels as *distantly useful*.

If at least half of the significant submatrices contain each at least $k/4$ distantly useful 1-pixels, then we claim that with high constant probability, the sample S^2_1 will contain a violating pair. To see why this is true, consider partitioning S^2_1 further in to two parts, $S^{2,1}_1$ and $S^{2,2}_1$, where $S^{2,1}_1$ is of size $\Theta(g_1/\epsilon)$ and S^2_1 is of size $\Theta(g_1\log n/\epsilon^2)$. Since there are $\Omega(g_1)$ submatrices that are significant and contain each at least $k/4 = \Omega(\epsilon g_3)$ distantly useful 1-pixels and at most g_3 1-pixels, with high constant probability, $S^{2,1}_1$ will contain some distantly useful 1-pixel from at least a constant fraction of the $\Omega(g_1)$ significant submatrices. Each of these $\Omega(g_1)$ 1-pixels has at least $\frac{\ell(k/s)}{8\log n}$ neighbors in $G^3_{\rm viol}(M)$ for $\ell > g_2$. That is, $\Omega\left(\frac{g_2\cdot\epsilon^2\cdot g_3}{g_4\cdot\log n}\right)$ neighbors (where two 1-pixels from different submatrices have different neighbors). The probability of getting a 1-pixel in the the union of these neighbors in a single sample is lower bounded by

$$\begin{aligned}
&\Omega\left(\frac{g_1\cdot g_2\cdot\epsilon^2\cdot g_3}{g_4\cdot\log n\cdot w(M)}\right)\\
&= \Omega\left(\frac{\epsilon^2\cdot(n^{2/3}/w(M)^{1/3})\cdot(w(M)^{2/3}/n^{1/3})\cdot w(M)^{4/3}/n^{2/3})}{n^{1/3}w(M)^{1/3}\cdot\log n\cdot w(M)}\right) &\quad (3)\\
&= \frac{\epsilon^2 w(M)^{1/3}}{n^{2/3}\log n}\ . &\quad (4)
\end{aligned}$$

Since S^2_1 is of size $\Theta(g_1\log n/\epsilon^2)$, it will contain one such neighboring 1-pixel with high constant probability.

On the other hand, if at least half of the significant submatrices contain each at least $k/4$ closely useful 1-pixel, then with high constant probability, the sample S_2 will contain at least one such 1-pixel. Assuming this is indeed the case, let (x,y) be the 1-pixel and let $\ell(x,y) \leq g_2$ be such that (x,y) has at least $\frac{\ell(x,y)\cdot(k/s)}{8\log n}$ neighbors in the $\ell(x,y)\times\ell(x,y)$ submatrix that it is the bottom-right or top-left corner of. Once, in the last step of the algorithm, we reach ℓ such that $\ell(x,y) \leq \ell \leq 2\ell(x,y)$, the probability of "hitting" a neighbor of (x,y) (in one of the abovementioned submatrices) in a single query is $\Omega(k/(\ell\cdot s\log n)) = \Omega(\epsilon^2 g_3/(\ell g_4\log n)) = \epsilon^2 w(M)/(\ell n\log n)$. Since the algorithm asks $\Theta(\ell\cdot(n/w(M))\cdot\log n\cdot\epsilon^{-2})$ queries in this submatrix, one such neighbor will be obtained with high constant probability.

It remains to prove Lemma 6.4. To this end we first prove the next claim (for an illustration see Figure 9).

Claim 6.5 *Consider a submatrix whose length in each dimension is at most ℓ, and that is determined by a 1-pixel (x_0, y_0) in its bottom-left corner and a 1-pixel (x_5, y_5) in its top-right corner. Then there exists a subset of at most four 1-pixels $\{(x_i, y_i)\}_{i=1}^4$ for which the following holds. The subsequence $\{(x_i, y_i)\}_{i=0}^5$ is monotonically non-decreasing, and each of the submatrices defined by a pair of 1-pixels (x_{i-1}, y_{i-1}) and (x_i, y_i) either has length at most $\ell/2$ in each dimension, or contains no other 1-pixels.*

Proof: We first partition the submatrix into four fourths. We shall have at most two 1-pixels in the bottom-left fourth (one of them is (x_0, y_0)), at most two 1-pixels in the top-right fourth (one of them is (x_5, y_5)), and at most two 1-pixels in either the top-left fourth or the bottom right fourth. Given (x_i, y_i) (where $0 \leq i \leq 3$), if (x_i, y_i) is not already in the top-right fourth (this may occur for $i < 4$), then we select (x_{i+1}, y_{i+1}) as follows.

1. If (x_i, y_i) is the first 1-pixel in the fourth it belongs to (that is, (x_{i-1}, y_{i-1}) belongs to a different fourth or doesn't exist), then we do the following. We consider all 1-pixels $(x, y) > (x_i, y_i)$ in the same fourth as (x_i, y_i). If there is no such 1-pixel, then we proceed as in the next item (that is, when (x_i, y_i) is the second 1-pixel in its fourth). If there is at least one such 1-pixel then we further consider those 1-pixel with the largest y coordinate, and amongst them let (x_{i+1}, y_{i+1}) have the largest x coordinate. In this case, the submatrix defined by (x_i, y_i) and (x_{i+1}, y_{i+1}) has length at most $\ell/2$ in each dimension (since both 1-pixels belong to the same fourth). Furthermore, there are no 1-pixels $(x, y) > (x_{i+1}, y_{i+1})$ in the same fourth as (x_{i+1}, y_{i+1}).

2. Otherwise ((x_i, y_i) is the second 1-pixel in its submatrix) we do the following. Observe that in this case there are no 1-pixels $(x, y) > (x_i, y_i)$ in the same fourth as (x_i, y_i). Here we let $(x_{i+1}, y_{i+1}) > (x_i, y_i)$ be a 1-pixel in one of the other fourths such that the submatrix defined by (x_i, y_i) and (x_{i+1}, y_{i+1}) contains no other 1-pixels. Such a 1-pixel can be defined by considering all 1-pixels $(x, y) > (x_i, y_i)$ that have the smallest y coordinate, and among them let (x_{i+1}, y_{i+1}) have the smallest x coordinate.

By the above construction, the matrix defined by (x_i, y_i) and (x_{i+1}, y_{i+1}) has either length at most $\ell/2$ in each dimension (when both 1-pixels belong to the same fourth of the submatrix), or it contains no additional 1-pixels. Since by the construction there can be at most two 1-pixels in each fourth (and there can't be both a 1-pixel in the top-left fourth and in the bottom-right), the claim follows. ■

Proof of Lemma 6.4: Assume that the claim in the lemma does not hold. We'll reach a contradiction to the premise of the lemma (regarding the size of a minimum vertex cover) by constructing a smaller vertex cover. We start by putting in the cover all, 1-pixels that for some $\ell = s/2^t$ have at least $\frac{\ell \cdot (k/s)}{8 \log n}$ neighbors in an $\ell \times \ell$ submatrix that they are the corner of. By the counter assumption, there are fewer than $k/2$ such 1-pixels.

We continue working in $O(\log n)$ iterations, where in each iteration we select a new set of 1-pixels. Each such set is monotonically increasing and has no neighbors among previously selected 1-pixels. We then add to the cover all their neighbors. We'll stop when each of the submatrices between consecutive 1-pixels contains no edges within it. Starting from $t = 1$, we'll ensure that in each iteration t, every submatrix has length at most $s/2^t$ in each dimension. In iteration t we

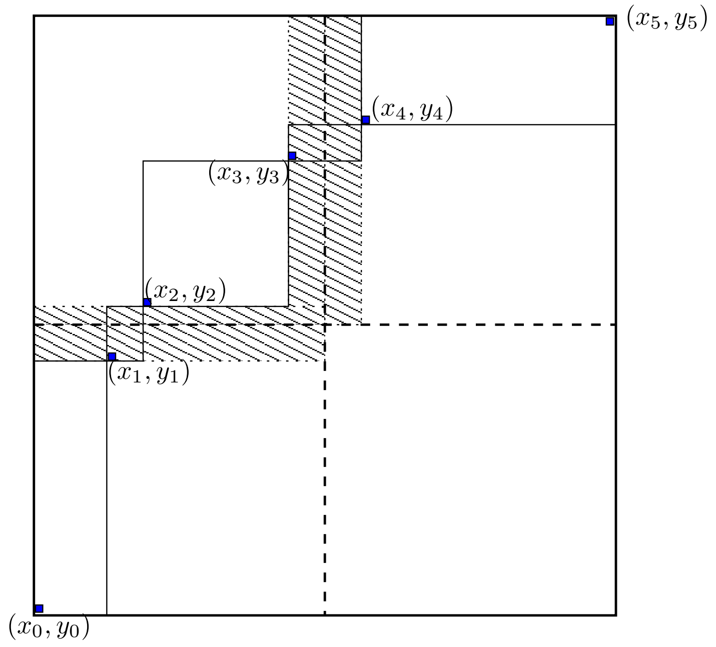

Figure 9: An illustration for the proof of Claim 6.5.

consider those submatrices that have length at least $s/2^{t+1}$ in each dimension, where the total number of such matrices is at most 2^{t+2}.

By Claim 6.5, for each such submatrix, there exist at most four 1-pixels that break the submatrix into smaller submatrices with the following properties. Each of the smaller submatrices either has length at most $s/2^{t+1}$ in each dimension, or it contains no edges within it. Since each of these four 1-pixels has at most $\frac{(s/2^t)\cdot(k/s)}{8\log n} = \frac{k}{2^{t+3}\log n}$ neighbors within the (bigger) submatrix, each iteration adds to the cover at most $2^{t+2}\cdot\frac{k}{2^{t+3}\log n} = \frac{k}{2\log n}$ 1-pixels. Since there are less than $\log n$ iterations, this stage contributes at most $k/2$ 1-pixels to the cover. The resulting cover is of size less that $k/2 + k/2 = k$, and we reach a contradiction. ■

6.2 A Lower Bound

We next show that there is a lower bound that almost matches our upper bound (for $w(M) = O(n)$).

Theorem 6.2 *Any (two-sided error) testing algorithm for monotonicity must have complexity* $\Omega(\min\{w(M)^{1/2}, n^{2/3}/w(M)^{1/3}\})$ *(for constant ϵ). The lower bound holds when $w(M) = O(n)$ and the algorithm is given an estimate $\widehat{w}$ such that $w(M)/2 \le \widehat{w} \le 2w(M)$.*

We note that we can obtain the lower bound also when $w(M)$ is known exactly, but since our algorithm only uses a constant factor estimate of $w(M)$, for the sake of simplicity we prove the theorem as stated.

Proof: We first establish the lower bound for $\widehat{w} \le n^{4/5}$ (where it is $\Omega(w(M)^{1/2})$), we then establish it for $\widehat{w} = n$ (where it is $\Omega(n^{2/3}/w(M)^{1/3} = \Omega(n^{1/3}))$, and after that we deal with the general case that $\widehat{w}$ ranges between $n^{4/5}$ and n (where the lower bound should behave like $\Omega(n^{2/3}/w(M)^{1/3})$).

The case $\widehat{w} \leq n^{4/5}$. We define the following two families of matrices. In both families we consider a partition of the matrices into submatrices of size $(n/\widehat{w}) \times (n/\widehat{w})$. Note that since $\widehat{w} \leq n^{4/5}$, the number of entries in each submatrix is at least $w(M)^{1/2}$. In the first family each matrix is determined by selecting, in each submatrix that is on the diagonal of these submatrices, a random entry either in the top-left fourth of the submatrix or in the bottom-right fourth, and setting it to 1. In the second family each matrix is determined by selecting, in each submatrix on the diagonal, both a random entry in the top-left fourth of the submatrix, and a random entry in the bottom-right fourth, and setting them both to 1. Clearly every matrix in the first family is monotone, and every matrix in the second family is 1/2-far from monotone. For an illustration, see Figure 10. The construction is such that for each matrix M in the first family we have that $w(M) = \widehat{w}/2$, and in the second family $w(M) = \widehat{w}$. In order to get the exact same value of $w(M)$ we can simply modify the second family so that only half of the diagonal submatrices are occupied (with two 1-pixels).

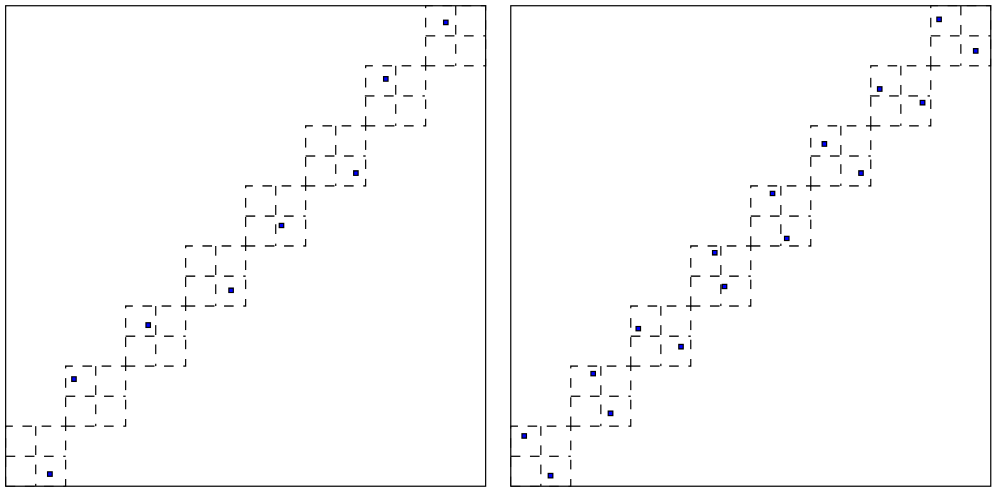

Figure 10: An illustration for the two families used for the lower bound when $\widehat{w} \leq n^{4/5}$. The submatrices on the diagonal (and their partition into 4 fourths) are denoted by dashed lines.

However, suppose that the algorithm performs $o(w(M)^{1/2})$ queries and takes a sample of at most that size. Consider its execution when the matrix is either chosen uniformly in the first family or is chosen uniformly in the second family. We may assume without loss of generality that the algorithm first takes a sample of 1-pixels and then performs queries. Observe that in both distributions over matrices, the probability of the sample "hitting" the same submatrix twice in a sample of size $o(w(M)^{1/2})$ is $o(1)$ (since there are $O(w(M))$ submatrices). However, if there is no such collision then the samples are distributed identically.

Turning to the queries, note that the number of pixels in each (fourth of a) submatrix is $\Omega(n^2/w(M)^2) = \Omega(w(M)^{1/2})$. Since in the distribution over the first family there is a single 1 in each diagonal submatrix whose location is equally distributed in the top-left or bottom-right fourth of each submatrix, and in the second there are two 1-pixels, each uniformly distributed in one of the two fourths, if the algorithm performs $o(w(M)^{1/2})$ queries, they are all answered by 0 with probability $1-o(1)$ [9]. We have thus showed that with high probability it is not possible to distinguish between a uniformly selected matrix in the first family (which should be accepted with high constant probability) and a uniformly selected submatrix in the second family (which should be rejected with high constant probability).

D: more elaboration?

G: Is the footnote sufficient? Necessary?

The case $\widehat{w} = n$. In this case we simply "scale up" the lower bound construction of the previous case in the following manner. Here we think of a partition of each $n \times n$ matrix into submatrices of size $n^{1/3} \times n^{1/3}$, where only the $n^{2/3}$ diagonal submatrices are non-empty. In the first family, each matrix is determined by selecting, for each diagonal submatrix, either the top-left fourth or the bottom-right fourth and within the selected fourth, selecting one of the $n^{1/3}/2$ rows to be filled with 1's. In the second family such a row is selected in both families. Here too every matrix M in the first family is monotone and $w(M) = \widehat{w}/2$, and every matrix M in the second family is 1/2-far from monotone and $w(M) = \widehat{w}$. For an illustration, see Figure 11.

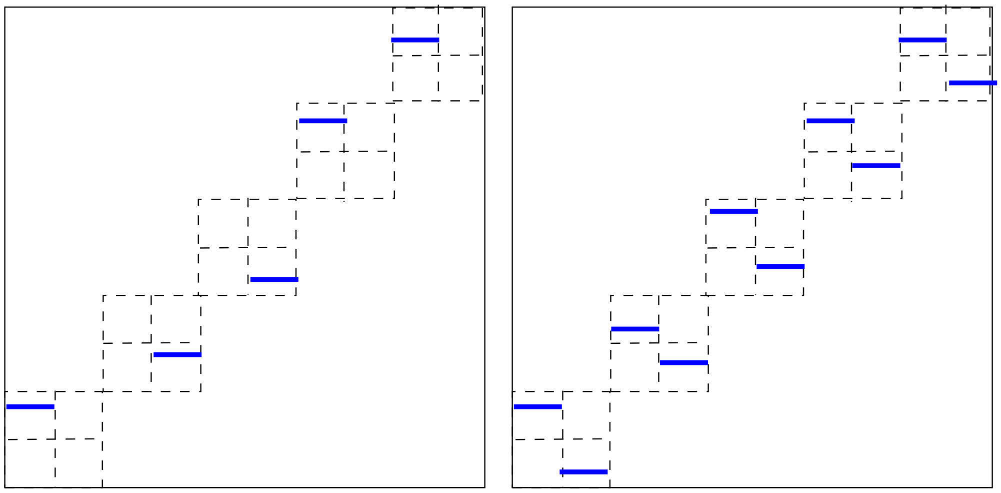

Figure 11: An illustration for the two families used for the lower bound when $\widehat{w} = n$. The submatrices on the diagonal (and their partition into 4 fourths) are denoted by dashed lines.

Since there are $n^{2/3}$ non-empty submatrices, if the algorithm takes a sample of size $o(n^{1/3})$, then with probability $1-o(1)$ it won't hit the same submatrix twice (in both distributions). Conditioned on there being no collisions, the distributions on sampled 1-pixels are identical for both families. As for the queries, since the number of rows in each fourth of a submatrix is $n^{1/3}/2$, if the algorithm performs $o(n^{1/3})$ queries, with probability $1-o(1)$ (over a uniformly selected matrix in either one of the two families), the answers to its queries are all 0. Thus we get a lower bound of $\Omega(n^{1/3})$ for the case that $w(M) = \Theta(\widehat{w}) = \Theta(n)$ (that is, $\Omega(n^{2/3}/w(M)^{1/3})$).

The case $n^{4/5} < \widehat{w} < n$. For the general case we partition the matrices into submatrices of size $\ell \times \ell$ where $\ell = \widehat{w}^{2/3}/n^{1/3}$, so that there are $t = n/\ell = n^{4/3}/\widehat{w}^{2/3}$ submatrices on the diagonal. Similarly to the case $\widehat{w} = n$, in one distribution (over a family of matrices) a row is selected uniformly either from the top-left fourth or the bottom right fourth, and in the second distribution a random row is selected from each. However, here each selected row is not completely filled with

[9]To see this note that we may consider the queries to be non-adaptive so long as a 1-pixel isn't encountered. Thus, the probability of a given algorithm hitting on a pixel that wasn't previously sampled must be at most $O(1/w(M)^{1/2})$ per query.

1's but rather each is randomly filled with $k = \widehat{w}/2t = \widehat{w}^{5/3}/2n^{4/3}$ 1's, so that the density of 1's in the selected rows is $\alpha = k/(\ell/2) = \widehat{w}/n$. Note that this coincides (for the appropriate rounding) with our constructions for $\widehat{w} = n$ (where $\ell = n^{2/3}$, $t = n^{2/3}$, $k = n^{1/3}/2$ and $\alpha = 1$) and $\widehat{w} = n^{4/5}$ (where $\ell = n^{1/5}$, $t = n^{4/5}$, $k = 1$ and $\alpha = 1/(n^{1/5}/2)$). We refer to these rows as the "occupied" rows.

Similarly to the arguments given for the cases $\widehat{w} \leq n^{4/5}$ and $\widehat{w} = n$, if the algorithm takes a sample of size $o(n^{2/3}/w(M)^{1/3}) = o(\sqrt{t})$ (where t is the number of submatrices on the diagonal), with probability $1 - o(1)$, it won't hit the same submatrix twice. Conditioned on this, the distributions on sampled 1-pixels are identical. Turning to the queries, here too we claim that if the algorithm performs $o(n^{2/3}/w(M)^{1/3}) = o(\ell^2/k) = o(\ell/\alpha)$ queries (where ℓ is the length of each submatrix, k is the number of 1's in the submatrix and $\alpha = \widehat{w}/n$ is the density of the 1's in each row), then with probability $1 - o(1)$ all queries are answered by 0.

The verify this, suppose that the algorithm performs $\beta \cdot n^{2/3}/w(M)^{1/3}$ queries for $\beta = o(1)$. The main observation is that we may assume without loss of generality that the algorithm selects its queries non-adaptively (though possibly based on the 1-pixels it got in the sampling stage). This is true because conditioned on all its queries being answered by 0, there is no use in adaptivity (and once the algorithm gets an answer of 1 on *any* query, it has "won", in the sense that it may be able to distinguish between the two distributions).

Thus consider any fixed choice of $\beta \cdot n^{2/3}/w(M)^{1/3} = \beta \cdot (\ell/\alpha)$ queries for $\beta = o(1)$ (which corresponds to a setting of the coin-flips of the algorithm). We shall show that for both distributions, the probability that any one of these queries is answered by 1 is $o(1)$. For $1 \leq i \leq t$ (recall that t is the number of submatrices on the diagonal) and for $1 \leq j \leq \log(\ell/2)$, let $s^1_{i,j}$ be the number of subrows in the top-left fourth of the i^{th} diagonal submatrix that contain between 2^{j-1} and 2^j queries, and similarly define $s^2_{i,j}$ for the bottom-right fourth. Then the probability that any query in one of the subrows is answered by 1 (under both distributions) is upper bounded (using the union bound) by

$$\sum_{i=1}^{t} \sum_{j=1}^{\log(\ell/2)} \left(\frac{s^1_{i,j}}{\ell/2} \cdot 2^j \cdot \alpha + \frac{s^2_{i,j}}{\ell/2} \cdot 2^j \cdot \alpha \right)$$

$$= \frac{2\alpha}{\ell} \sum_{i=1}^{t} \sum_{j=1}^{\log(\ell/2)} (s^1_{i,j} + s^2_{i,j}) \cdot 2^j \qquad (5)$$

$$\leq \frac{2\alpha}{\ell} \cdot \frac{2\beta\ell}{\alpha} = 4\beta = o(1) , \qquad (6)$$

and the proof is completed. ■

Acknowledgements

We wish to thank the FOCS 2010 reviewers for taking the time to read the article in depth, and for their many insightful comments.

References

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

[AK02] N. Alon and M. Krivelevich. Testing k-colorability. *SIAM Journal on Discrete Math*, 15(2):211–227, 2002.

[BGJ$^+$09] A. Bhattacharyya, E. Grigorescu, K. Jung, S. Raskhodnikova, and D. Woodruff. Transitive-closure spanners. In *soda09*, pages 932–941, 2009.

[BRW05] T. Batu, R. Rubinfeld, and P. White. Fast approximate PCPs for multidimensional bin-packing problems. *Information and Computation*, 196(1):42–56, 2005.

[CLM06] B. Chazelle, D. Liu, and A. Magen. Sublinear geometric algorithms. *SIAM Journal on Computing*, 35(3):627–646, 2006.

[CS01] A. Czumaj and C. Sohler. Property testing with geometric queries. In *Proceedings of the Ninth Annual European Symposium on Algorithms (ESA)*, pages 266–277, 2001.

[CS05] A. Czumaj and C. Sohler. Abstract combinatorial programs and efficient property testers. *SIAM Journal on Computing*, 34(3):580–615, 2005.

[CSZ00] A. Czumaj, C. Sohler, and M. Ziegler. Property testing in computation geometry. In *Proceedings of the Eighth Annual European Symposium on Algorithms (ESA)*, pages 155–166, 2000.

[DGL$^+$99] Y. Dodis, O. Goldreich, E. Lehman, S. Raskhodnikova, D. Ron, and A. Samorodnitsky. Improved testing algorithms for monotonicity. In *Proceedings of the Third International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM)*, pages 97–108, 1999.

[EKK$^+$00] F. Ergun, S. Kannan, S. R. Kumar, R. Rubinfeld, and M. Viswanathan. Spot-checkers. *Journal of Computer and System Sciences*, 60(3):717–751, 2000.

[Fis04] E. Fischer. On the strength of comparisons in property testing. *Inf. Comput.*, 189(1):107–116, 2004.

[FLN$^+$02] E. Fischer, E. Lehman, I. Newman, S. Raskhodnikova, R. Rubinfeld, and A. Samorodnitsky. Monotonicity testing over general poset domains. In *Proceedings of the Thirty-Fourth Annual ACM Symposium on the Theory of Computing (STOC)*, pages 474–483, 2002.

[GGL$^+$00] O. Goldreich, S. Goldwasser, E. Lehman, D. Ron, and A. Samorodnitsky. Testing monotonicity. *Combinatorica*, 20(3):301–337, 2000.

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

[GR02] O. Goldreich and D. Ron. Property testing in bounded degree graphs. *Algorithmica*, pages 302–343, 2002.

[HK03] S. Halevy and E. Kushilevitz. Distribution-free property testing. In *Proceedings of the Seventh International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM)*, pages 341–353, 2003.

[HK04] S. Halevy and E. Kushilevitz. Testing monotonicity over graph products. In *Automata, Languages and Programming: Thirty-First International Colloquium (ICALP)*, pages 721–732, 2004.

[KKN10] I. Kleiner, D. Keren, and I. Newman. Applying property testing to an image partitioning problem. To appear in IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), 2010.

[KKR04] T. Kaufman, M. Krivelevich, and D. Ron. Tight bounds for testing bipartiteness in general graphs. *SIAM Journal on Computing*, 33(6):1441–1483, 2004.

[KV94] M. Kearns and U. Vazirani. *An introduction to computational learning theory.* MIT Press, Cambridge, MA, USA, 1994.

[MBCGS10] A. Matsliah, J. Briet, S. Chakraborty, and D. Garci'a-Soriano. Monotonicity testing and shortest-path routing on the cube. In *Proceedings of the Fourteenth International Workshop on Randomization and Computation (RANDOM)*, 2010.

[PR02] M. Parnas and D. Ron. Testing the diameter of graphs. *Random Structures and Algorithms*, 20(2):165–183, 2002.

[Ras03] S. Raskhodnikova. Approximate testing of visual properties. In *Proceedings of the Seventh International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM)*, pages 370–381, 2003.

[RS96] R. Rubinfeld and M. Sudan. Robust characterization of polynomials with applications to program testing. *SIAM Journal on Computing*, 25(2):252–271, 1996.

[RV04] L. Rademacher and S. Vempala. Testing geometric convexity. In *Proceedings of the International Conference on the Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 469–480, 2004.

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

[VC71] V. N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its applications*, 17(2):264–280, 1971.