Deleting and Testing Forbidden Patterns in Multi-Dimensional Arrays

Omri Ben-Eliezer[1], Simon Korman[2], and Daniel Reichman[3]

1 **Blavatnik School of Computer Science, Tel Aviv University, Tel Aviv, Israel**
omribene@gmail.com
2 **Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot, Israel**
simon.korman@gmail.com
3 **Electrical Engineering and Computer Science, University of California, Berkeley, CA, USA**
daniel.reichman@gmail.com

Abstract

Analyzing multi-dimensional data is a fundamental problem in various areas of computer science. As the amount of data is often huge, it is desirable to obtain sublinear time algorithms to understand local properties of the data.

We focus on the natural problem of testing *pattern freeness*: given a large d-dimensional array A and a fixed d-dimensional pattern P over a finite alphabet Γ, we say that A is *P-free* if it does not contain a copy of the forbidden pattern P as a consecutive subarray. The distance of A to P-freeness is the fraction of the entries of A that need to be modified to make it P-free.

For any $\epsilon > 0$ and any large enough pattern P over any alphabet – other than a very small set of exceptional patterns – we design a *tolerant tester* that distinguishes between the case that the distance is at least ϵ and the case that the distance is at most $a_d\epsilon$, with query complexity and running time $c_d\epsilon^{-1}$, where $a_d < 1$ and c_d depend only on the dimension d. These testers only need to access uniformly random blocks of samples from the input A.

To analyze the testers we establish several combinatorial results, including the following d-dimensional *modification lemma*, which might be of independent interest: For any large enough d-dimensional pattern P over any alphabet (excluding a small set of exceptional patterns for the binary case), and any d-dimensional array A containing a copy of P, one can delete this copy by modifying one of its locations without creating new P-copies in A.

Our results address an open question of Fischer and Newman, who asked whether there exist efficient testers for properties related to tight substructures in multi-dimensional structured data.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Property testing, Sublinear algorithms, Pattern matching

Digital Object Identifier 10.4230/LIPIcs.ICALP.2017.9

1 Introduction

Pattern matching is the algorithmic problem of finding occurrences of a fixed pattern in a given string. This problem appears in many settings and has applications in diverse domains such as computational biology, computer vision, natural language processing and web search. There has been extensive research concerned with developing algorithms that search for patterns in strings, resulting with a wide range of efficient algorithms [12, 24, 19, 14, 26, 25]. Higher-dimensional analogues where one searches for a d-dimensional pattern in a d-dimensional

44th International Colloquium on Automata, Languages, and Programming (ICALP 2017).
Editors: Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl;
Article No. 9; pp. 9:1–9:14

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

array have received attention as well. For example, the 2D case arises in analyzing aerial photographs [7, 8] and the 3D case has applications in medical imaging.

Given a string S of length n and a pattern P of length $k \leq n$, any algorithm which determines whether P occurs in S has running time $\Omega(n)$ [13, 29] and a linear lower bound carries over to higher dimensions. For the 2D case, when an $n \times n$ image is concerned, algorithms whose run time is $O(n^2)$ are known [8]. These algorithms have been generalized to the 3D case to yield running time of $O(n^3)$ [18] for $n \times n \times n$ arrays. Finally it is also known (e.g., [21]) that for the d-dimensional case it is possible to solve the pattern matching problem in time $O(d^2 n^d \log m)$ (where the pattern is an array of size m^d). It is natural to ask which tasks of this type can be performed in sublinear (namely $o(n^d)$) time for d-dimensional arrays.

The field of *property testing* [20, 30] deals with decision problems regarding discrete objects (e.g., graphs, functions, images) that either satisfy a certain property $\mathcal{P}$ or are *far* from satisfying $\mathcal{P}$. Here, the property $\mathcal{P}$ consists of all d-dimensional arrays that avoid a predetermined pattern P. *Tolerant* property testing [27] is a useful extension of the standard notion, in which the tester needs to distinguish between objects that are *close* to satisfying the property and those that are *far* from satisfying it.

A d-dimensional $k_1 \times \ldots \times k_d$ array A over an alphabet Γ is a function from $[k_1] \times \ldots \times [k_d]$ to Γ, where for an integer $k > 0$ we let $[k]$ denote the set $\{0, \ldots, k-1\}$ and write $[k]^d = [k] \times \ldots \times [k]$. For simplicity of presentation, all results in this paper are presented for *square* arrays in which $k_1 = \ldots = k_d$, but they generalize to non-square arrays in a straightforward manner. We consider the (tolerant) *pattern-freeness problem*. An (ϵ_1, ϵ_2)-tester Q for this problem is a randomized algorithm that is given access to a d-dimensional array A, as well as its size and proximity parameters $0 \leq \epsilon_1 < \epsilon_2 < 1$. Q needs to distinguish with probability at least $2/3$ between the case that A is ϵ_1-close to being P-free and the case that A is ϵ_2-far from being P-free. The query complexity of Q is the number of queries it makes in A.

Our interest in the pattern-freeness problem stems from several applications. In certain scenarios of interest, we might be interested in identifying quickly that an array is far from not containing a given pattern. In the one dimensional case, being far from not containing a given text may indicate a potential anomaly which requires attention (e.g., an offensive word in social network media), hence such testing algorithms may provide useful in anomaly detection. Many computer vision methods for classifying images are feature based; being far from not containing a certain pattern associated with a feature may be useful in rejection methods that enable us to quickly discard images that do not possess a certain visual property.

Beyond practical applications, devising property testing algorithms for the pattern freeness problem is of theoretical interest. In the first place, it leads to a combinatorial characterization of the distance from being P-free. Such a characterization has proved fruitful in graph property testing [3, 4] where celebrated graph removal lemmas were developed en route of devising algorithms for testing subgraph freeness. We encounter a similar phenomena in studying patterns and arrays: at the core of our approach for testing pattern freeness lies a *modification lemma* for patterns which we state next. We believe that this lemma may be of independent interest and find applications beyond testing algorithms.

For a pattern P of size $k \times k \times \ldots \times k$, an entry whose location in $\mathcal{P}$ is in $\{0, k-1\} \times \ldots \times \{0, k-1\}$ is said to be a *corner* of P. We say that P is *almost homogeneous* if all of its entries but one are equal, and the different entry lies in a corner of P. Finally, P is *removable* (with respect to the alphabet Γ) if for any d-dimensional array A over Γ and any copy of P in A, one can destroy the copy by modifying one of its entries without creating new P-copies in A. The modification lemma states that for any d, and any large enough pattern P, when the

alphabet is binary it holds that P is removable *if and only if* it is not almost homogeneous, and when the alphabet is not binary, P is removable provided that it is large enough.

Recent works [10, 11] have obtained tolerant testers for visual properties. As observed in these works, tolerance is an attractive property for testing visual properties as real-world images are often noisy. With the modification lemma at hand, we show that when P is removable, the (relative) *hitting number* of P in A, which is the minimal size of a set of entries that intersects all P-copies in A divided by $|A|$, differs from the distance of A from P-freeness by a multiplicative factor that depends only on the dimension d of the array. This relation allows us to devise very fast $(5^{-d}\epsilon, \epsilon)$-tolerant testers for P-freeness, as the hitting number of P in A can be well approximated using only a very small sample of blocks of entries from A. The query complexity of our tester is c_d/ϵ, where c_d is a positive constant depending only on d. Note that our characterization in terms of the hitting number is crucial: Merely building on the fact that A contains many occurrences of P (as can be derived directly from the modification lemma) and randomly sampling $O(1/\epsilon)$ possible locations in A, checking whether the sub-array starting at these locations equals P, would lead to query complexity of $O(k^d/\epsilon)$. Note that our tester is optimal (up to a multiplicative factor that depends on d), as any tester for this problem must make $\Omega(1/\epsilon)$ queries.

The one dimensional setting, where one seeks to determine quickly whether a string S is ϵ-far from being P-free is of particular interest. We are able to leverage the modification lemma and show that the distance of a string S from being P-free for a fixed pattern P (that is not almost homogeneous) is *exactly equal* to the hitting number of P in A. For an arbitrary constant $0 < c < 1$, this characterization allows us to devise a $((1-c)\epsilon, \epsilon)$-tolerant tester making $O_c(1/\epsilon)$ queries for this case. For the case of almost homogeneous patterns, and an arbitrary constant $c > 0$, we devise a $((1/(16+c))\epsilon, \epsilon)$-tolerant tester that makes $O_c(1/\epsilon)$ queries. Whether tolerant testers exist for almost homogenous patterns of dimension larger than 1 is an open question.

2 Related Work

The problem of testing pattern freeness is related to the study of testing subgraph freeness (see, for example, [1, 4, 2]). This line of work examines how one can test quickly whether a given graph G is H-free or ϵ-far from being H-free, where H is a fixed subgraph. In this problem, a graph is ϵ-far from being H-free if at least an ϵ-fraction of its edges and non-edges need to be altered in order to ensure that the resulting graph does not contain H as a (not necessarily induced) subgraph. A key component in these works are removal lemmas: typically such lemmas imply that if G is ϵ-far from being H-free, then it contains a large number of copies of H. For example, the *triangle removal lemma* asserts that for every $\epsilon > 0$, there exists $\delta = \delta(\epsilon) > 0$ such that if an n-vertex graph G is ϵ-far from being triangle free, then G contains at least δn^3 triangles (see, e.g., [6] and the references within).

Alon *et. al.* [3] studied the problem of testing regular languages. Testing pattern-freeness (1-dimensional, binary alphabet, constant pattern length k) is a special case of the former, since the language L of all strings avoiding a fixed pattern is regular. The query complexity of their tester is $O\left(\frac{s^3}{\epsilon} \cdot \ln^3(\frac{1}{\epsilon})\right)$, where s is the minimal size of a DFA that accepts the regular language L. In the case of the regular language considered here a simple pumping-lemma inspired argument shows that $s \geq \Omega(k)$. Hence the upper bound on testing pattern freeness implied by their algorithm is $O\left(\frac{k^3}{\epsilon} \cdot \ln^3(\frac{1}{\epsilon})\right)$. Our 1D tester solves a very restricted case of the problem the tester of [3] deals with, but it achieves a query complexity of $O(1/\epsilon)$ in this setting. Moreover, our tester is much simpler and can be applied in the more general high

dimensional setting, or when the pattern length k is allowed to grow as a function of the string length n.

The problem of testing *submatrix freeness* is investigated in [15, 16, 5, 17, 2]. As opposed to our case, which is concerned with *tight* submatrices, all of these results deal with submatrices that are not necessarily tight (i.e. the rows and the columns need not be consecutive). Quantitatively, the submatrix case is very different from our case: in our case P-freeness can be testable using $O(\epsilon^{-1})$ queries, while in the submatrix case, for a binary submatrix of size $k \times k$ a lower bound of $\epsilon^{-\Omega(k^2)}$ on the needed number of queries is easy to obtain, and in the non-binary case there exist 2×2 matrices for which there exists a super polynomial lower bound of $\epsilon^{-\Omega(\log 1/\epsilon)}$.

The 2D part of our work adds to a growing literature concerned with testing properties of images [28, 31, 10]. Ideas and techniques from the property testing literature have recently been used in the fields of computer vision and pattern recognition [22, 23].

3 Notation and definitions

An (n, d)-*array* A over an alphabet Γ is a function from $[n]^d$ to Γ. The $x = (x_1, \ldots, x_d)$ *entry* of A, denoted by A_x, is the value of the function A at location x. Let P be a (k, d)-array over an alphabet Γ of size at least two. We say that a d-dimensional array A *contains* a copy of P (or a P-copy) *starting in location* $x = (x_1, \ldots, x_d)$ if for any $y \in [k]^d$ we have $A_{x+y} = P_y$. Finally, A is P-free if it does not contain copies of P.

A property $\mathcal{P}$ of d-dimensional arrays is simply a family of such arrays over an alphabet Γ. For an array A and a property $\mathcal{P}$, the *absolute distance* $d_{\mathcal{P}}(A)$ of A to $\mathcal{P}$ is the minimal number of entries that one needs to change in A to get an array that satisfies $\mathcal{P}$. The *relative distance* of A to $\mathcal{P}$ is $\delta_{\mathcal{P}}(A) = d_{\mathcal{P}}(A)/|A|$, where clearly $0 \leq \delta_{\mathcal{P}}(A) \leq 1$ for any nontrivial $\mathcal{P}$ and A. We say that A is ϵ-close to $\mathcal{P}$ if $\delta_{\mathcal{P}}(A) \leq \epsilon$, and ϵ-far if $\delta_{\mathcal{P}}(A) \geq \epsilon$. In this paper we consider the property of P-freeness, which consists of all P-free arrays. The absolute and relative distance to P-freeness are denoted by $d_P(A)$ and $\delta_P(A)$, respectively.

For an array A and a pattern P, a *deletion set* is a set of entries in A whose modification can turn it to be P-free, and $d_P(A)$ is called the *deletion number*, since it is the size of a minimal deletion set. Similarly, a given set of entries in A is a *hitting set* if every P-copy in A contains at least one of these entries. The *hitting number* $h_P(A)$ is the size of the minimal hitting set for P in A. For all notation here and above, in the 1-dimensional case we replace A by S (for String).

We use several definitions from [27]. Let $\mathcal{P}$ be a property of arrays and let $h_1, h_2 : [0, 1] \to [0, 1]$ be two monotone increasing functions. An (h_1, h_2)-*distance approximation* algorithm for $\mathcal{P}$ is given query access to an unknown array A. The algorithm outputs an estimate $\hat{\delta}$ to $\delta_P(A)$, such that with probability at least $2/3$ it holds that $h_1(\delta_P(A)) \leq \hat{\delta} \leq h_2(\delta_P(A))$. For a property $\mathcal{P}$ and for $0 \leq \epsilon_1 < \epsilon_2 \leq 1$, an (ϵ_1, ϵ_2)-*tolerant tester* for $\mathcal{P}$ is given query access to an array A. The tester *accepts* with probability at least $2/3$ if A is ϵ_1-close to $\mathcal{P}$, and *rejects* with probability at least $2/3$ if A is ϵ_2-far from $\mathcal{P}$. In the "standard" notion of property testing, $\epsilon_1 = 0$. Thus, any tolerant tester is also a tester in the standard notion. Finally, we define the additive (multiplicative) *tolerance* of the tester above as $\epsilon_2 - \epsilon_1$ (ϵ_2/ϵ_1 respectively).

Table 1 Summary of results. $0 < \tau < 1$ and $c > 0$ are arbitrary constants. α_c is a constant that depends only on c. $\beta_{d,\tau}$ is a constant that depends only on d and τ. 'modification lemma' specifies if patterns are classified as removable or not. The 'tester tolerance' is multiplicative.

dim.	template type	modification lemma	tester tolerance	query complexity
1D	general	removable for any k	$1/(1-\tau)$	$O(1/\epsilon\tau^3)$
	almost homog.	not removable for any k	$16+c$	α_c/ϵ
2^+D	general	removable for $k > 3 \cdot 2^d$	$(1-\tau)^{-d}\alpha_d$	$\beta_{d,\tau}/\epsilon$
	almost homog.	not removable for any k	–	–

4 Main Results

The modification lemma presented is central in the study of minimal deletion sets. It classifies the possible patterns into ones that are removable and ones that are not. The result that the vast majority of patterns are removable is used extensively throughout the paper in the design and proofs of algorithms for efficient testing of pattern freeness (in 1 and higher dimensions).

Our 1-dimensional modification lemma (Lemma 1) gives the following full characterization of 1-dimensional patterns (i.e. strings). A binary pattern is removable *if and only if* it is not almost homogeneous, while *any* pattern over a larger alphabet is removable. The multidimensional version of the lemma (Lemma 2) makes the exact same classification, but for (k,d)-arrays for which $k \geq 3 \cdot 2^d$.

The fact that most patterns are removable is very important for analyzing the deletion number. For example, observe that a removable pattern appears at least $d_P(A)$ times (possibly with overlaps) in the array A, so an ϵ-tester can simply check for the presence of the d-dimensional pattern in $1/\epsilon$ random locations in the array, using $O(k^d/\epsilon)$ queries.

Another important part of our work makes explicit connections between the deletion number and the hitting number for both 1 and higher dimensions. These are needed in order to get improved testers (e.g. for getting rid of k in the sample complexity) in d-dimensions.

In the 1-dimensional removable case we show that the deletion number $d_P(S)$ equals the hitting number $h_P(S)$. We derive a $((1-\tau)\epsilon,\epsilon)$-tolerant tester for any fixed $\tau > 0$ and any $0 < \epsilon \leq 1$, whose number of queries and running time are $O(\epsilon^{-1}\tau^{-3})$ (Corollary 13).

For higher dimensions, we show (Lemma 11) that $h_P(A) \leq d_P(A) \leq \alpha_d h_P(A) \leq \alpha_d k^{-d}$, where $\alpha_d = 4^d + 2^d$ depends only on the dimension d. This bound gives a $((1-\tau)^d\alpha_d^{-1}\epsilon,\epsilon)$-tolerant tester making $C_\tau\epsilon^{-1}$ queries, where $C_\tau = O(1/\tau^d(1-(1-\tau)^d)^2)$ (Theorem 15). The running time here is $C'_\tau\epsilon^{-1}$ where C'_τ depends only on τ.

In the full version of the paper [9], for the 1-dimensional setting we also provide dedicated algorithms to handle the almost homogeneous (non-removable) patterns, obtaining an $O(n)$ algorithm for computing the deletion number as well as an $(\epsilon/(16+c),\epsilon)$-tolerant tester, for any constant $c > 0$, using $\alpha_c\epsilon^{-1}$ queries, where α_c depends only on c.

Finally, we provide a lower bound of $\Omega(1/\epsilon)$ (full proof in can be found in [9]) for any general tester of pattern freeness. Our main results are summarized in Table 1.

A natural question is what happens if one is concerned with pattern freeness with respect to several patterns simultaneously. Namely, testing quickly whether an array satisfies the property of not containing a fixed set of patterns $P_1,\ldots,P_r$ with $r > 1$, or is far from satisfying this property. Our results do not apply to this setting, with the main obstacle being our modification lemmas. Namely, the difficulty is that for several distinct patterns $P_1 \ldots P_r$, modifying an occurrence of P_i may create an new occurrence of P_j (where $i \neq j$).

5 Modification Lemma

We begin with the proof of the 1-dimensional modification lemma. The main strategy here is to consider the longest streaks of zeros and ones (0s and 1s) in the pattern - a strategy that cannot be used in higher dimensions.

▶ **Theorem 1** (1D Modification Lemma). *For a 1-dimensional pattern P over an alphabet Γ:*

1. *If $|\Gamma| = 2$ then P is removable if and only if it is not almost homogeneous.*
2. *If $|\Gamma| \geq 3$ then P is removable.*

▶ Remark. The second statement of the theorem can be easily derived from the first statement. If P does not contain all letters in Γ then it is clearly removable, as changing any of its entries to any of the missing letters cannot create new P-copies. Otherwise, we can reduce the problem to the binary case: let σ_1, σ_2 be the letters in Γ that appear the smallest number of times in P (specifically, if P is of length 3 and has exactly three different letters, we pick σ_1 and σ_2 to be the first letter and the last letter in P, respectively). Consider the following pattern P' over $\{0,1\}$: $P'_x = 0$ if $P_x \in \{\sigma_1, \sigma_2\}$ and $P'_x = 1$ otherwise. Observe that P' is not almost homogeneous, implying that it is removable. It is not hard to verify now that P is removable as well.

Proof of Theorem 1. As discussed above, it is enough to consider the binary case. Let $P = P_0 \dots P_{k-1}$ be a binary pattern of length k that is not almost homogeneous, and let S be an arbitrary binary string containing P. We need to show that one can flip one of the bits of P without creating a new P-copy in S. We assume that P contains both 0s and 1s (i.e. it is not homogeneous) otherwise flipping any bit would work. We may assume that $k \geq 3$ (since for $k = 1, 2$ all patterns are homogeneous or almost homogeneous). Let us also assume that P starts with a 1, i.e. $P_0 = 1$ and let $t \leq k-1$ be the length of the longest 0-streak (sub-string of consecutive 0s) in P. Let $i > 0$ be the leftmost index in which such a 0-streak of length t begins. Clearly, $P_{i-1} = 1$ and $P_i = \dots = P_{i+t-1} = 0$.

If $i + t \leq k - 1$ (i.e. the streak is not at the end of P) then $P_{i+t} = 1$ and in such a case if we modify P_{i+t} to 0, the copy of P is removed without creating new P-copies in S. To see this, observe that a new copy cannot start at the bit flip location $i + t$ or within the 0-streak at any of its locations $i, \dots, i + t - 1$ since the bits in these locations are 0 while the starting bit of P is 1. Note that flipping P_{i+t} does not create new P-copies starting after the location of P_{i+t} in S. Furthermore, no new copy starting before P_i is created since otherwise it would contain a 0-streak of length $t + 1$.

Thus, we may assume that P contains exactly one 0-streak of length t, lying at its last t locations, so $P_{k-1} = 0$. Denote by s the length of the longest 1-streak in P; a symmetric reasoning shows that P begins with its only 1-streak of length s. If P is *not* of the form 1^s0^t, it can be verified that flipping P_s (the leftmost 0 in P) to 1 does not create any P-copy. The only case left is $P = 1^s0^t$, where $s, t \geq 2$ since P is not almost homogeneous. Consider the bit of the string S that is to the left of P. If it is a 0 then we flip P_1 to 0 and otherwise, we flip P_0 to 0, where in both cases no new copy is created. ◀

We now turn to proving the high dimensional version of the modification lemma. Here, as opposed to the 1-dimensional case, there exist patterns that are neither removable nor almost-homogeneous. However, we show that if a pattern is large enough and not almost homogeneous then it is removable.

▶ **Theorem 2** (Modification Lemma). *Let $d > 1$ and let P be a (k, d)-array over the alphabet Γ where $k \geq 3 \cdot 2^d$.*

1. *If $|\Gamma| = 2$ then P is removable if and only if it is not almost homogeneous.*
2. *If $|\Gamma| \geq 3$ then P is removable.*

▶ Remark. Theorem 2 states that any large enough binary pattern which is not almost homogeneous is removable. The requirement that the pattern is large enough is crucial, as can be seen from the following 2-dimensional example. The 2×2 pattern $P = [01; 01]$ is in the center of the 4×4 matrix $A = [0101; 0011; 0011; 0101]$. Flipping any bit in P creates a new P-copy in A. This example easily generalizes to a $2 \times \ldots \times 2$ pattern in a $4 \times \ldots \times 4$ array, while the dependence of k on d is exponential in the statement of Theorem 2. It will be interesting to understand what is the correct order of magnitude of this dependence.

Proof of Theorem 2. The reduction from a general alphabet to a binary one, described after the statement of Theorem 1, can be used here as well. Thus, it is enough to prove the first statement of the theorem. If P is binary and almost homogeneous then it is not removable: Without loss of generality $P_{(0,\ldots,0)} = 1$ and $P_x = 0$ for any $x \neq (0, \ldots, 0)$. Consider a $(2k, d)$-array A such that $M_{(0,\ldots,0)} = M_{(1,\ldots,1)} = 1$ and $A = 0$ elsewhere. Modifying any bit of the P-copy starting at $(1, \ldots, 1)$ creates a new P-copy in A, hence P is not removable.

The rest of the proof is dedicated to the other direction. Suppose that P is a binary (k, d)-array that is not removable. We would like to show that P must be almost homogeneous. As P is not removable, there exists a binary array A containing a copy of P such that flipping any single bit in this copy creates a new copy of P in A. This copy of P will be called the *template* of P in A.

Clearly, all of the new copies created by flipping bits in the template must intersect the template, so we may assume that A is of size $(3k - 2)^d$ and that the template starts in location $\tilde{k} = (k - 1, \ldots, k - 1)$. For convenience, let $I = [k]^d$ denote the set of indices of P. For any $x \in I$ let $\bar{x} = x + \tilde{k}$; $\bar{x}$ is the location in A of bit x of the template.

Roughly speaking, our general strategy for the proof is to show that there exist at most two "special" entries in P such that when we flip a bit in the template (creating a new copy of P in A) the flipped bit usually plays the role of one of the special entries in the new copy. We then show that in fact, there must be exactly one special entry, which must lie in a corner of P, and that all non-special entries are equal while the special entry is equal to their negation. This will finish the proof that P is almost homogeneous.

▶ **Definition 3.** Let $i \leq d$ and let δ be positive integers. Let $x = (x_1, \ldots, x_d)$ and $y = (y_1, \ldots, y_d)$ be d-dimensional points. The pair (x, y) is (i, δ)-*related* if $y_i - x_i = \delta$ and $y_j = x_j$ for any $j \neq i$. An (i, δ)-related pair (x, y) is said to be an (i, δ)-*jump* in P if $P_x \neq P_y$.

In other words, (x, y) is (i, δ)-*related* if there is an increasing path of length δ along coordinate i in the hypergrid graph on $[n]^d$.

▶ **Lemma 4.** *For any $1 \leq i \leq d$ there exists $0 < \Delta_i < k/3$ such that at most two of the (i, Δ_i)-related pairs of points from I are (i, Δ)-jumps in P.*

Proof. Recall that, by our assumption, flipping any of the $K = k^d$ bits of the template creates a new copy of P in A. Consider the following mapping $m : I \to [2k-1]^d$. $m(x_1, \ldots, x_d)$ is the starting location of a new copy of P created in A as a result of flipping bit $x = (x_1, \ldots, x_d)$ of the template (which is bit $\bar{x}$ of A). If more than one copy is created by this flip, then we choose the starting location of one of the copies arbitrarily.

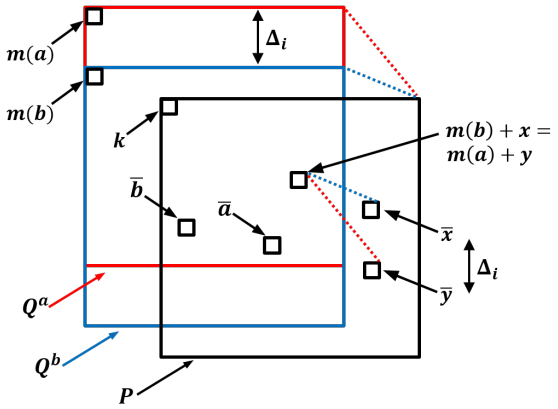

Figure 1 Illustration for Lemma 4. A 2-dimensional example, where i is the vertical coordinate: Flipping the bit (of the template P) at location $\bar{a}$ creates the P-copy Q^a at location $m(a)$. Similarly, the copy Q^b is created at location $m(b)$. Note that the pair of points $(\bar{x}, \bar{y})$ (which is (x, y) in P) and the copy locations pair $(m(a), m(b))$ are both (i, Δ_i)-related. The values P_x and P_y ($M_{\bar{x}}$ and $M_{\bar{y}}$) must be equal.

Observe that m is injective, and let S be the image of m, where $|S| = K$. Let $1 \leq i \leq d$ and consider the collection of (1-dimensional) lines

$$\mathcal{L}_i = \big\{\{x_1\} \times \ldots \times \{x_{i-1}\} \times [2k-1] \times \{x_{i+1}\} \times \ldots \times \{x_d\} \quad | \quad \forall j \neq i : x_j \in [2k-1]\big\}.$$

Clearly $\sum_{\ell \in \mathcal{L}_i} |S \cap \ell| = K$. On the other hand, $|\mathcal{L}_i| = \prod_{j \neq i}(2k-1) < 2^{d-1} \prod_{j \neq i} k = 2^{d-1}K/k$, so there exists a line $\ell \in \mathcal{L}_i$ for which $|S \cap \ell| > k/2^{d-1} \geq 6$. Hence $|S \cap \ell| \geq 7$. Let $\alpha_1 < \ldots < \alpha_7$ be the smallest i-indices of elements in $S \cap \ell$. Since $\alpha_7 - \alpha_1 < 2k-1$ there exists some $1 \leq l \leq 6$ such that $\alpha_{l+1} - \alpha_l < k/3$. That is, S contains an (i, Δ_i)-related pair with $0 < \Delta_i < k/3$. In other words, there are two points $a, b \in I$ such that flipping $\bar{a}$ ($\bar{b}$) would create a new P-copy, denoted by Q^a (Q^b respectively), which starts in location $m(a)$ ($m(b)$ respectively) in A, and $(m(a), m(b))$ is an (i, Δ_i)-related pair.

The following claim finishes the proof of the lemma and will also be useful later on.

▶ **Claim 5.** *For a and b as above, let (x, y) be a pair of points from I that are (i, Δ_i)-related and suppose that $y \neq \bar{a} - m(a)$ and that $x \neq \bar{b} - m(b)$. Then $P_x = P_y$.*

Proof. The bits that were flipped in A to create Q^a and Q^b are $\bar{a}, \bar{b}$ respectively. Since $y + m(a) \neq \bar{a}$, the entry $M_{y+m(a)}$ serves as the entry of the P-copy Q^a in location y, so $P_y = M_{y+m(a)}$. Similarly, since $x + m(b) \neq \bar{b}$, we have $P_x = M_{x+m(b)}$. But since both pairs (x, y) and $(m(a), m(b))$ are (i, Δ_i)-related, we get that $m(b) - m(a) = y - x$, implying that $x + m(b) = y + m(a)$, and therefore $P_x = M_{x+m(b)} = M_{y+m(a)} = P_y$, as desired. ◀

Clearly, the number of (i, Δ_i)-related pairs that do not satisfy the conditions of the claim is at most two, finishing the proof of Lemma 4. ◀

Let $\Delta = (\Delta_1, \ldots, \Delta_d)$ where for any $1 \leq i \leq d$, we take Δ_i that satisfies the statement of Lemma 4 (its specific value will be determined later).

▶ **Definition 6.** Let $x \in I$. The set of Δ-*neighbours* of x is

$$N_x = \big\{y \in I \quad | \quad \exists i : (x, y) \text{ is } (i, \Delta_i)\text{-related or } (y, x) \text{ is } (i, \Delta_i)\text{-related}\big\}$$

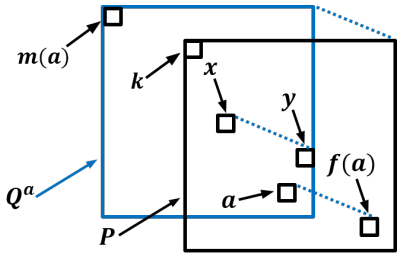

Figure 2 Illustration for Definition 8. Recall that flipping a bit $\bar{a}$ in A creates a new P-copy Q^a (which contains $\bar{a}$), located at the point $m(a)$ in the coordinates of A. The bits x and a are *mapped* to y and $f(a)$ respectively.

and the number of Δ-neighbours of x is $n_x = |N_x|$, where $d \leq n_x \leq 2d$. We say that x is a Δ-*corner* if $n_x(\Delta) = d$ and that it is Δ-*internal* if $n_x(\Delta) = 2d$. Furthermore, x is (Δ, P)-*isolated* if $P_x \neq P_y$ for any $y \in N_x$, while it is (Δ, P)-*generic* if $P_x = P_y$ for any $y \in N_x$.

When using the above notation, we sometimes omit the parameters (e.g. simply writing *isolated* instead of (Δ, P)-isolated) as the context is usually clear.

The definition imposes a symmetric neighborhood relation, that is, $x \in N_y$ holds if and only if $y \in N_x$. If $x \in N_y$ we say that x and y are Δ-neighbours. Note that a point $x = (x_1, \ldots, x_d) \in I$ is a Δ-corner if $x_i < \Delta_i$ or $x_i \geq k - \Delta_i$ for any $1 \leq i \leq d$, and that x is Δ-internal if $\Delta_i \leq x_i < k - \Delta_i$ for any $1 \leq i \leq d$.

▶ **Claim 7.** *Two (Δ, P)-isolated points in I cannot be Δ-neighbors.*

Proof. Suppose towards contradiction that $x = (x_1, \ldots, x_d)$ and $y = (y_1, \ldots, y_d)$ are two distinct (Δ, P)-isolated points and that (x, y) is (i, Δ_i)-related for some $1 \leq i \leq d$. Since $\Delta_i < k/3$, at least one of x or y participates in two different (i, Δ_i)-related pairs: if $x_i < k/3$ then $y_i + \Delta_i = x_i + 2\Delta_i < k$ so y is in two such pairs, and otherwise $x_i \geq \Delta_i$, meaning that x participates in two such pairs. Assume without loss of generality that the two (i, Δ_i)-related pairs are (t, x) and (x, y), then $P_t \neq P_x$ and $P_x \neq P_y$ as x is isolated. By Lemma 4, these are the only (i, Δ_i)-jumps in P.

Choose an arbitrary $j \neq i$ and take $v = (v_1, \ldots, v_d)$ where $v_j = \Delta_j$ and $v_l = 0$ for any $l \neq j$. Recall that $\Delta_j < k/3$, implying that either $x_j + v_j < k$ or $x_j - v_j \geq 0$. Without loss of generality assume the former, and let $x' = x + v$ and $y' = y + v$. Since x and y are (Δ, P)-isolated, and since $x' \in N_x$ and $y' \in N_y$, we get that $P_{x'} \neq P_x \neq P_y \neq P_{y'}$, and thus $P_{x'} \neq P_{y'}$ (as the alphabet is binary). Therefore, (x', y') is also an (i, Δ_i)-jump in P, a contradiction. ◀

▶ **Definition 8.** For three points $x, y, a \in I$, we say that x is *mapped* to y as a result of the flipping of a if $\bar{x} = m(a) + y$. Moreover, define the function $f : I \to I$ as follows: $f(x) = \bar{x} - m(x)$ is the location to which x is mapped as a result of flipping x.

In other words, x is mapped to y as a result of flipping the bit a if bit $\bar{x}$ of A "plays the role" of bit y in the new P-copy Q_a that is created by flipping a. Note that

- If $\bar{x} - m(a) \notin I$ then x is not mapped to any point. However, this cannot hold when $x = a$, so the function f is well defined.

- For a fixed a, the mapping as a result of flipping a is linear: if x and y are mapped to x' and y' respectively, then $y - x = y' - x'$. In particular, if (x, y) is (i, Δ_i)-related for some $1 \leq i \leq d$ then (x', y') is also (i, Δ_i)-related.
- If x is mapped to y as a result of flipping a and $x \neq a$, then $P_x = P_y$.
- On the other hand, we always have $P_x \neq P_{f(x)}$.
- If x is Δ-internal and (Δ, P)-generic, then $f(x)$ must be (Δ, P)-isolated.

The first four statements are easy to verify. To verify the last one, suppose that x is internal and generic and let $z \in N_{f(x)}$; we will show that $P_{f(x)} \neq P_z$. Since x is internal, there exists $y \in N_x$ such that $y - x = z - f(x)$. Then y is mapped to z as a result of flipping x, since $\bar{y} = y + \tilde{k} = z + (x + \tilde{k}) - f(x) = z + \bar{x} - f(x) = z + m(x)$. Therefore $P_y = P_z$. On the other hand, $P_x = P_y$ as x is generic and $P_x \neq P_{f(x)}$, and we conclude that $P_z \neq P_{f(x)}$.

▶ **Lemma 9.** *There is exactly one (Δ, P)-isolated point in I.*

Proof. Let $\mathcal{S}$ be the set of isolated points; our goal is to show that $|\mathcal{S}| = 1$. Consider the set

$$C = \{(x, y) : x, y \in I, (x, y) \text{ is an } (i, \Delta_i)\text{-jump for some } 1 \leq i \leq d\}.$$

Clearly, each point in $\mathcal{S}$ is contained in at least d pairs from C. By claim 7 no pair of isolated points are Δ-neighbours and therefore every pair in C contains at most one point from $\mathcal{S}$. By Lemma 4, $|C| \leq 2d$ which implies that $|\mathcal{S}| \leq 2$. On the other hand we have $|\mathcal{S}| \geq 1$. To see this, observe that the number of (Δ, P)-internal points in I is greater than $\prod_{i=1}^{d} k/3 \geq 2^{d^2}$, while the number of non-Δ-generic points is at most $2|C| \leq 4d$, implying that at least $2^{d^2} - 4d > 0$ of the internal points are generic. Therefore, pick an internal generic point $z \in I$. As we have seen before, $f(z)$ must be isolated.

To complete the proof it remains to rule out the possibility that $|\mathcal{S}| = 2$. If two different (Δ, P)-isolated points $a = (a_1, \ldots, a_d)$ and $b = (b_1, \ldots, b_d)$ exist, each of them must participate in exactly d pairs in C. This implies that both of them are Δ-corners with d neighbors. It follows that every Δ-internal point z must be generic (since an internal point and a corner point cannot be neighbours), implying that either $f(z) = a$ or $f(z) = b$.

Let $1 \leq i \leq d$ and define $\delta_i > 0$ to be the smallest integer such that there exists an (i, δ_i)-related pair (x, y) of generic internal points with $f(x) = f(y)$. For this choice of x and y we have $m(y) - m(x) = \bar{y} - f(y) - (\bar{x} - f(x)) = \bar{y} - \bar{x} = y - x$, so $(m(x), m(y))$ is also (i, δ_i)-related. In particular, we may take $\Delta_i = \delta_i$ (Recall that until now, we only used the fact that $\Delta_i < k/3$, without committing to a specific value). Without loss of generality we may assume that $f(x) = f(y) = a$. By Claim 5, any pair (s, t) of (i, Δ_i)-related points for which $s \neq \bar{y} - m(y) = f(y) = a$ and $t \neq \bar{x} - m(x) = f(x) = a$ is not an (i, Δ_i)-jump. Since b is not a Δ-neighbour of a, it does not participate in any (i, Δ_i)-jump, contradicting the fact that it is (Δ, P)-isolated. This finishes the proof of the lemma. ◀

Finally, we are ready to show that P is almost homogeneous. Let $a = (a_1, \ldots, a_d)$ be the single (Δ, P)-isolated point in I. Consider the set

$$J = \{x = (x_1, \ldots, x_d) \in I : \Delta_i \leq x_i < \Delta_i + 2^d \text{ for any } 1 \leq i \leq d\}$$

and note that all points in J are Δ-internal. Let $1 \leq i \leq d$ and partition J into $(i, 1)$-related pairs of points. There are $2^{d^2-1} \geq 4d$ pairs in the partition. On the other hand, the number of non-generic points in J is at most $2|C| - (d - 1) < 4d$ (to see it, count the number of elements in pairs from C and recall that a is contained in at least d pairs). Therefore, there exists a pair (x, y) in the above partition such that x and y are both generic. As before,

$f(x)$ and $f(y)$ must be isolated, and thus $f(x) = f(y) = a$, implying that $\Delta_i = \delta_i = 1$. We conclude that $\Delta = (1, \ldots, 1)$.

Claim 5 now implies that any pair (s, t) of $(i, 1)$-related points for which $s \neq \bar{y} - m(y) = f(y) = a$ and $t \neq \bar{x} - m(x) = f(x) = a$ is not an $(i, 1)$-jump. That is, for any two neighbouring points $s, t \neq a$ in I, $P_s = P_t$, implying that $P_x = P_y$ for any $x, y \neq a$ (since $\Delta = (1, \ldots, 1)$, a Δ-neighbour is a neighbour in the usual sense). To see this, observe that for any two points $x, y \neq a$ there exists a path $x_0 x_1 \ldots x_t$ in I where x_j and x_{j+1} are neighbours for any $0 \leq j \leq t-1$, the endpoints are $x_0 = x$ and $x_t = y$, and $x_j \neq a$ for any $0 < j < t$. Since a is isolated, it is also true that $P_a \neq P_x$ for any $x \neq a$.

To finish the proof that P is almost homogeneous, it remains to show that a is a corner. Suppose to the contrary that $0 < a_i < k - 1$ for some $1 \leq i \leq d$ and let $b, c \in I$ be the unique points such that (a, b) and (c, a) are $(i, 1)$-related, respectively. Clearly $f(b) = a$, so a is mapped to $\bar{a} - m(b) = \bar{a} - \bar{b} + f(b) = c - a + a = c$ as a result of flipping b, which is a contradiction - as $P_a \neq P_c$ and $b \neq a, c$. This finishes the proof. ◀

6 From Deletion to Testing

We use the modification lemmas of Section 5 to investigate combinatorial characterizations of the deletion number, which in turn allow efficient approximation and testing of pattern freeness for removable patterns. In particular, we prove that minimal deletion sets and minimal hitting sets are closely related. Due to space considerations, the proofs of all results in this Section do not appear here, and are given in the full version of this paper [9].

The characterizations for almost homogeneous 1-dimensional patterns are also given in the full version of the paper [9], along with an optimal tester for pattern freeness in that case. The rest of this section deals with removable patterns, for both the 1-dimensional and multi-dimensional settings.

In the 1-dimensional case, we show that for any removable pattern there exist certain minimal hitting sets which are in fact minimal deletion sets. These are sets where none of the flips create new occurrences. Our constructive proof shows how to build such a set.

▶ **Theorem 10** ($d_P(S)$ equals $h_P(S)$)**.** *For a binary string S of length n and a binary pattern P of length k that is removable, the deletion number $d_P(S)$ equals the hitting number $h_P(S)$.*

For the multidimensional case, we show that when P is removable, the hitting number $h_P(A)$ of A approximates the deletion number up to a multiplicative constant that depends only on the dimension d. This is done in two stages, the first of which involves the analysis of a procedure that proves the existence of a large collection of P-copies with small pairwise overlaps, among the set of all P-copies in A. This procedure heavily relies on the fact that P is removable. The second stage shows that since these copies have small overlaps, their hitting number cannot be much different than their deletion number. The result is summarized in Lemma 11.

▶ **Lemma 11** (Relation between distance and hitting number)**.** *Let P be a removable (k, d)-array over an alphabet Γ, and let A be an (n, d)-array over Γ. Let $\alpha_d = 4^d + 2^d$. It holds that: $h_P(A) \leq d_P(A) \leq \alpha_d h_P(A) \leq \alpha_d (n/k)^d$.*

We describe efficient distance approximation algorithms and testers for both the 1-dimensional and the d-dimensional removable patterns. The tolerance of the testers depends only on d, and the query complexity is linear in ϵ^{-1} where the constant depends only on d (and not on k; using a completely naive tester, it can be seen that the tolerance and the query

complexity depend on k). The distance approximation algorithms and the testers essentially estimate the hitting number by sampling a small set of $O(k) \times \ldots \times O(k)$ uniformly chosen consecutive subarrays of the input array, and calculating the hitting number of the pattern P in each of these samples.

▶ **Theorem 12** (Approximating the deletion number in 1-dimension). *Let P be a removable string of length k and fix constants $0 < \tau < 1, 0 < \delta < 1/k$. Let $h_1, h_2 : [0,1] \to [0,1]$ be defined as $h_1(\epsilon) = (1-\tau)\epsilon - \delta$ and $h_2(\epsilon) = \epsilon + \delta$. There exists an (h_1, h_2)-distance approximation algorithm for P-freeness with query complexity and running time of $O(1/k\tau\delta^2)$.*

Note that $d_P(S) = h_P(S) \leq n/k$ always holds, so having an additive error parameter of $\delta \geq 1/k$ is pointless. The proof of Theorem 12 can be adapted to derive an (ϵ_1, ϵ_2)-tolerant tester for any $0 \leq \epsilon_1 < \epsilon_2 \leq 1$, yielding the following multiplicative-error tester.

▶ **Corollary 13** (Multiplicative tolerant tester for pattern freeness in 1-dimension). *Fix $0 < \tau < 1$. For any $0 < \epsilon \leq 1$ there exists a $((1-\tau)\epsilon, \epsilon)$-tolerant tester whose number of queries and running time are $O(\epsilon^{-1}\tau^{-3})$.*

For the multidimensional case, our distance approximation algorithm and tolerant tester for P-freeness are given in Theorems 14 and 15.

▶ **Theorem 14** (Approximating the deletion number in multidimensional arrays). *Let P be a removable (k,d)-array and fix constants $0 < \tau \leq 1, 0 \leq \delta \leq 1/k^d$. Let $h_1, h_2 : [0,1] \to [0,1]$ be defined as $h_1(\epsilon) = (1-\tau)^d \alpha_d^{-1}\epsilon - \delta$ and $h_2(\epsilon) = \epsilon + \delta$. There exists an (h_1, h_2)-distance approximation algorithm for P-freeness making at most $\gamma/k^d\tau^d\delta^2$ queries, where $\gamma > 0$ is an absolute constant, and has running time $\zeta_\tau/k^d\delta^2$ where ζ_τ is a constant depending only on τ.*

▶ **Theorem 15** (Multiplicative tolerant tester for pattern freeness in multidimensional arrays). *Fix $0 < \tau \leq 1$ and let P be a removable (k,d)-array. For any $0 < \epsilon \leq 1$ there exists a $((1-\tau)^d \alpha_d^{-1}\epsilon, \epsilon)$-tolerant tester making $C_\tau \epsilon^{-1}$ queries, where $C_\tau = O(1/\tau^d(1-(1-\tau)^d)^2)$. The running time is $C'_\tau \epsilon^{-1}$ where C'_τ depends only on τ.*

7 Discussion and Open Questions

We have provided efficient algorithms for testing whether high-dimensional arrays do not contain P for any fixed removable pattern P. The results suggest several interesting open questions on the problem of pattern-freeness and more generally, on local properties - where we say that a property $\mathcal{P}$ is k-local (for $k \ll n$) if for any array A not satisfying $\mathcal{P}$, there exists a consecutive subarray of A of size at most $k \times \ldots \times k$ which does not satisfy $\mathcal{P}$ as well. That is, a property is local if any array not satisfying $\mathcal{P}$ contains a small 'proof' for this fact. Note that P-freeness is indeed k-local where k is the side length of P, and that a property $\mathcal{P}$ is k-local if and only if there exists a *family* $\mathcal{F}$ of arrays of size at most $k \times \ldots \times k$ each, such that A satisfies $\mathcal{P}$ if and only if it does not contain any consecutive sub-array from $\mathcal{F}$. That is, to understand the general problem of testing local properties of arrays we will need to understand the testing of $\mathcal{F}$-freeness, where $\mathcal{F}$ is a family of forbidden patterns (rather than a single forbidden pattern). As mentioned, it is not clear how to apply our methods to develop testers for families of patterns. Devising testers for this case is an interesting open question.

The problem of approximate pattern matching is of interest as well. The family of forbidden patterns for this problem might consist of a pattern and all patterns that are close enough to it, and the distance measures between patterns might also differ from the Hamming distance (e.g., ℓ_1 distance for grey-scale patterns).

Finally, it is desirable to settle the problem of testing pattern freeness for the almost homogenous case by either finding an efficient tester for the almost homogeneous multi-dimensional case, or proving that an efficient tester cannot exist for such patterns. It is also of interest to examine which of the $[k]^d$ patterns with $k < 3 \cdot 2^d$ are removable.

Acknowledgements. We are grateful to Swastik Kopparty for numerous useful comments. We are thankful to Sofya Raskhodnikova for her useful feedback.

References

1. N. Alon (2002). Testing subgraphs in large graphs, *Random structures and algorithms*, 21(3–4):359–370.
2. N. Alon, O. Ben-Eliezer and E. Fischer (2017). *Testing hereditary properties of ordered graphs and matrices,* arXiv preprint 1704.02367.
3. N. Alon, M. Krivelevich, I. Newman and M. Szegedy (2001). Regular languages are testable with a constant number of queries, *SIAM Journal on Computing*, 30, 1842–1862.
4. N. Alon, E. Fischer, M. Krivelevich and M. Szegedy (2000). Efficient testing of large graphs, *Combinatorica*, 20, 451–476.
5. N. Alon, E. Fischer and I. Newman (2007). Efficient testing of bipartite graphs for forbidden induced subgraphs, *SIAM Journal on Computing*, 37.3, 959–976.
6. N. Alon and J. Spencer (2008). *The Probablistic Method.* Wiley.
7. A. Amir, G. Benson (1998). Two-Dimensional Periodicity in Rectangular Arrays, *SIAM Journal on Computing*, 27, 90–106.
8. A. Amir, G. Benson, M. Farach (1994). An Alphabet Independent Approach to Two-Dimensional Pattern Matching, *SIAM Journal on Computing*, 23, 313–323.
9. O. Ben-Eliezer, S. Korman and D. Reichman (2017). *Deleting and Testing Forbidden Patterns in Multi-Dimensional Arrays,* arXiv preprint 1607.03961.
10. P. Berman, M. Murzabulatov, S. Raskhodnikova (2015). Constant-Time Testing and Learning of Image Properties, arXiv prepreint 1503.01363.
11. P. Berman, M. Murzabulatov and Sofya Raskhodnikova (2016). Tolerant Testers of Image Properties. *ICALP*, 1–90:14.
12. R.S. Boyer and J.S. Moore (1977). A fast string searching algorithm, *Comm. ACM*, 20(10), 762–772.
13. R. Cole (1991). Tight Bounds on the Complexity of the Boyer-Moore String Matching Algorithm, *SODA*, 224–233.
14. C. Maxime, A. Czumaj, L. Gasieniec, S. Jarominek, T. Lecroq, W. Plandowski, and W. Rytter (1994). Speeding up two string-matching algorithms, *Algorithmica*, 12, 247–267.
15. E. Fischer, and I. Newman (2001). Testing of matrix properties, *STOC*, 286–295
16. E. Fischer and I. Newman (2007). Testing of matrix-poset properties, *Combinatorica*, 27(3), 293–327.
17. E. Fischer, E. Rozenberg, Lower bounds for testing forbidden induced substructures in bipartite-graph-like combinatorial objects, Proc. RANDOM 2007, 464–478.
18. Z. Galil, J. G. Park and K. Park. (2004). Three-dimensional periodicity and its application to pattern matching. *SIAM Journal on Discrete Mathematics*, 18(2), 362–381.
19. Z. Galil and J. I. Seiferas (1983). Time-Space-Optimal String Matching, *J. Comput. Syst. Sci*, 26(3), 280–294.
20. O. Goldreich, S. Goldwasser and D. Ron (1998). Property testing and its connection to learning and approximation, *JACM*, 45, 653–750.
21. J. Kärkkäinen and E. Ukkonen (2008). Multidimensional string matching. In *Encyclopedia of Algorithms*, 559–562.

22 I. Kleiner, D. Keren, I. Newman, O. Ben-Zwi (2011). Applying Property Testing to an Image Partitioning Problem, *IEEE Trans. Pattern Anal. Mach. Intell*, 33(2), 256–265.

23 S. Korman, D. Reichman, G. Tsur and S. Avidan. Fast-Match: Fast Affine Template Matching, International Journal of Computer Vision, to appear.

24 D. E. Knuth, J. H. Morris Jr. and V. R. Pratt (1977). Fast Pattern Matching in Strings, *SIAM J. Comput.* 6(2): 323–350.

25 T. Lecroq (2007). Fast exact string matching algorithms, *Information Processing Letters*, 102(6), 229–235.

26 G. Navarro and M. Raffinot (2000). Fast and flexible string matching by combining bit-parallelism and suffix automata, *Journal of Experimental Algorithmics (JEA)* 5:4.

27 M. Parnas, D. Ron and R. Rubinfeld (2006). Tolerant property testing and distance approximation. *Journal of Computer and System Sciences*, 72(6), 1012–1042.

28 S. Raskhodnikova (2003). Approximate testing of visual properties, *RANDOM*, 370–381.

29 R. L. Rivest (1977). On the Worst-Case Behavior of String-Searching Algorithms, *SIAM J. Comput.* 6(4): 669–674.

30 R. Rubinfeld and M. Sudan (1996). Robust characterization of polynomials with applications to program testing, *SIAM J. Comput.* 25, 252–271.

31 G. Tsur and D. Ron (2014). Testing properties of sparse images, *ACM Transactions on Algorithms* 4.

32 A. C. Yao (1977). Probabilistic computation, towards a unified measure of complexity, *FOCS*, 222–227.