Property Testing in Computational Geometry*

(Extended Abstract)

Artur Czumaj[1], Christian Sohler[2], and Martin Ziegler[2]

[1] Department of Computer and Information Science, New Jersey Institute of Technology, Newark, NJ 07102-1982, USA. czumaj@cis.njit.edu

[2] Heinz Nixdorf Institute and Department of Mathematics & Computer Science, University of Paderborn, D-33095 Paderborn, Germany. csohler,ziegler@uni-paderborn.de

Abstract. We consider the notion of *property testing* as applied to computational geometry. We aim at developing efficient algorithms which determine whether a given (geometrical) object has a predetermined property Q or is "far" from any object having the property. We show that many basic geometric properties have very efficient testing algorithms, whose running time is significantly smaller than the object description size.

1 Introduction

Property testing is a rapidly emerging new field in computer science with many theoretical as well as practical applications. It is concerned with the computational task of determining whether a given object has a predetermined property or is "far" from any object having the property. It is a relaxation of the standard definition of a decision task, because we allow arbitrary behavior when the object does not have the property, and yet it is close to an object having the property. A notion of property testing was first explicitly formulated in [15] and then extended and further developed in many follow-up works (see, e.g., [1, 2, 4, 7–12, 16]). Property testing arises naturally in the context of program verification, learning theory, and, in a more theoretical setting, in probabilistically checkable proofs. For example, in the context of program checking, one may first choose to test whether the program's output satisfies certain properties before checking that it is as desired. This approach is a very common practice in software development, where it is (typically) infeasible to require to formally test that a program is correct, but by verifying whether the output satisfies certain properties one can gain a reasonable confidence about the quality of the program's output.

In this work we consider property testing as applied to *geometric objects in Euclidean spaces*. We investigate property testing for some basic and most representative problems/properties in computational geometry. The main goal of this research is to develop algorithms which perform only a very small (sublinear, polylogarithmic, or even a constant) number of operations in order to check with a reasonable confidence the required geometric property.

We study the following family of tasks: *Given oracle access to an unknown geometric object, determine whether the object has a certain predefined property or is "far" from any object having this property.* Distance between objects is measured in terms of

* Research supported in part by DFG Grant Me872/7-1.

a *relative edit distance*, i.e., an object X (e.g., a set of points in $\mathbb{R}^d$) from a class $\mathcal{C}$ (e.g., a class of all point sets in $\mathbb{R}^d$) is called *ϵ-far from satisfying a property Q* (over $\mathcal{C}$) if no object Y from $\mathcal{C}$ which differs from X in no more than an ϵ fraction of places (e.g., Y can be constructed from X by adding and removing no more than $\epsilon \cdot |X|$ points) satisfies Q. The notion of "*oracle access*" corresponds to the representation of the objects in $\mathcal{C}$, and it is more problem dependent. And thus, for example, if a geometric object is defined as a collection of points in Euclidean space, then it is reasonable to require that the oracle allows the algorithm to ask only on the coordinate-position of each single input point. If however, a geometric object is, say, a polygon, then the oracle shall typically allow the algorithm to query for the position of each point as well as for each single neighbor of each point in the polygon. In this paper we shall always use the most natural notions of "oracle" for each problem at hand.

An *ϵ-tester* for a property Q is a (typically randomized) algorithm which always accepts any object satisfying Q and with probability at least $\frac{2}{3}$ rejects any object being ϵ-far from satisfying Q [1]. There are two types of possible complexity measures we focus on: the *query complexity* and the *running time* complexity of an ϵ-tester. The query complexity of a tester is measured only by the number of queries to the oracle, while the running time complexity counts also the time needed by the algorithm to perform other tasks (e.g., to compute a convex hull of some point set).

To exemplify the notion of property testing, let us consider the standard geometrical property Q of a point set P in Euclidean space $\mathbb{R}^d$ of being in *convex position*[2]. In this case, we aim at designing an algorithm which for a given oracle access to P has to decide whether P is in convex position or P is "ϵ-far" from being in convex position. Here, we say a *set P is ϵ-far from being in convex position*, $0 \leq \epsilon \leq 1$, if for every subset $X \subseteq P$ of size at most $\epsilon \cdot |P|$ the set $P \setminus X$ is not in convex position. Moreover, we assume that the oracle allows to query for the size of P and for the position of each ith point in P.

Another type of property we shall consider in this paper is that of being an *Euclidean minimum spanning tree* (*EMST*). That is, for a given set P of points in the plane we have to verify whether a given graph G with the vertex set P is an EMST for P or it is ϵ-far from being an EMST for P. Here we assume the algorithm can query the size of P, the position of each single ith point x in P as well as query the jth neighbor of x in G (with the special symbol indicating non-existence of such a neighbor). We say here that an Euclidean graph G with vertex set P is ϵ-far from being an EMST for P if the minimum spanning tree of P differs from G on at least $\epsilon \cdot |P|$ edges.

1.1 Description of New Results

In this paper we provide efficient ϵ-testers for some basic problems in computational geometry. We investigate problems which we found most representative for our study and which play an important role in the field. The following property testers are studied:

[1] One could consider also a *two-sided error* model or consider a confidence parameter δ rather than fixing an error bound $\frac{1}{3}$. These models are not discussed in this paper.

[2] A set of points P in $\mathbb{R}^d$ is in convex position if every point of P is *extreme*, that is, it is a vertex of $\mathrm{CH}(P)$, the convex hull of P.

Problem	Query complexity	Lower bound	Running time
convex position	$\mathcal{O}(n^d/\epsilon)^{1/(d+1)}$	$\Omega(n^d/\epsilon)^{1/(d+1)}$	$\mathcal{O}(n^{2/3}(1/\epsilon)^{1/3}\log(n/\epsilon))$ $d=2$ $\mathcal{O}(n^{3/4}(1/\epsilon)^{1/4}\log(n/\epsilon))$ $d=3$ $\mathcal{O}\Big(n\,\text{polylog}(1/\epsilon)+$ $d\geq 4$ $(\frac{n}{\epsilon})^{(\lfloor d/2\rfloor)/(1+\lfloor d/2\rfloor)}\,\text{polylog}(n)\Big)$
disjointness of geometric objects	$\mathcal{O}(\sqrt{n/\epsilon})$	$\Omega(\sqrt{n/\epsilon})$	$T(\mathcal{O}(\sqrt{n/\epsilon}))$
disjointness of polytopes	$\mathcal{O}\left((d/\epsilon)\cdot\log(d/\epsilon)\right)$	$\Omega(1/\epsilon)$	$\mathcal{O}(1/\epsilon)$ deterministic(!)
Delaunay triangulation			
Hamming distance	$\mathcal{O}(n)$	$\Omega(n)$	$\mathcal{O}(n)$
Distance defined in Theorem 5.2	$\mathcal{O}(1/\epsilon)$	$\Omega(1/\epsilon)$	$\mathcal{O}(1/\epsilon)$
EMST	$\mathcal{O}(\sqrt{n/\epsilon}\,\log^2(1/\epsilon))$	-	$\mathcal{O}(\sqrt{n/\epsilon}\,\log^2(1/\epsilon)\,\log n)$

Table 1. ϵ-testers obtained in the paper. $T(m)$ denote the best known running time of the algorithm deciding given property and we consider lower bounds with respect to the query complexity.

convex position: property that a set of points is in a *convex position*,

disjointness of geometric objects: property that an arrangement of objects is intersection-free,

disjointness of V-polytopes: property that two polytopes are intersection-free,

EMST: property that a given graph is a Euclidean *minimum spanning tree* of a given point set in the plane, and

Delaunay triangulation: property that a triangulation has the Delaunay property.

Table 1 summarizes the bounds of our ϵ-tester developed in the paper in details. As this table shows, all our algorithms are tight or almost tight regarding their query complexity. We also prove a general lower bound which shows that for many problems no *deterministic* ϵ-tester exists with the query complexity $o(n)$ already for $\epsilon = \frac{1}{4}$.

There are two main approaches used in our testers. The first one, which is quite general and is used in all our algorithms, is to sample at random a sufficiently large subset of input basic objects (e.g., input points) and then analyze the sample to obtain information about the whole input. The key issue in this method is to provide a good description of the input's property using a small input's sample. This approach is used in a simple way in our ϵ-testers for disjointness of geometric objects and Delaunay triangulation, where such a description is easy to show. In our ϵ-testers for convex position and disjointness of polytopes this approach is used as well, but it is more complicated to prove the desired property of the whole input out of the analysis of small sample. The second approach (combined with the first one) is used in our tester for EMST: after sampling a portion of the input (which are the vertices of the graph) at random, we enlarge the sample in a systematic way by exploring the paths of the length up to $\mathcal{O}(1/\epsilon)$ starting in the vertices chosen. Only then we can show that the sample chosen can be used to certificate whether the input graph is the EMST of the input points.

We mention also an important feature of all our algorithms, which is that all our testers supply proofs of violation of the property for rejected objects (because of space limitations we do not elaborate on this issue here).

Throughout the paper we always assume that d, the dimension of the Euclidean space under consideration, is a constant. Moreover, we assume the size of the input object to be much bigger than d. To simplify the exposition, we suppose also that all points/arrangements are in general position.

1.2 Motivation and Applications

What does this type of approximation mean. Testing a property is a natural relaxation of deciding that property, and it suggests a certain notion of approximation: the tester accepts an object which either has the property or which is "close" to some other object having the property. Under this notion, in applications where objects close to having the property are almost as good as ones having the property, a tester which is significantly faster than the corresponding decision procedure is a very valuable alternative to the latter. We refer the reader to [7, 10, 15], where many more detailed applications of this notion of approximation have been discussed (see, especially, [10, Section 1.2.1]).

In the context of computational geometry a related research has been presented in [3] and [14]. For example, Mehlhorn et al. [14] studied *program checkers* for some basic geometric tasks. Their approach is to design simple and efficient algorithms which checks whether an output of a geometric task is correct. It is shown in [14] that some basic geometric tasks can be designed so that program checkers can be obtained in a fairly simple (and efficient) way. This approach has been also successfully implemented as a part of the LEDA system. The main difference between our approach and that in [14] is that we (and, generally, property testing algorithms) do not aim at deciding the property at hand (in this case, whether the task output is correct), but instead we provide a certain notion of *approximation* — we can argue that we do not accept the property (or the algorithm's output) only if it is far from being satisfied. This relaxation enables us to obtain algorithms with running times *significantly* better than those in [14].

Output sensitive algorithms. An important, novel feature of our property testing algorithms is that they can be used to design *output sensitive* algorithms for decision problems. Consider for example, the problem of deciding whether a set P of n points on the plane is in convex position, and if it is not, to report a subset of P which is non-convex. Below we show how our ϵ-tester can be used to obtain an "output sensitive" (randomized) algorithm.

We set first $\epsilon = \frac{1}{2}$ and apply our tester to verify whether P is ϵ-far from being in convex position. If it is so, then we report a subset of P which violates the property of being convex. Otherwise, we decrease ϵ to $\epsilon/2$ and use our tester to verify whether P is ϵ-far from being in convex position. We repeat this procedure until we either reject set P, in which case we report also a subset of P which is non-convex, or we reach $\epsilon \leq 1/n$. In the latter case, we run a deterministic algorithm which asks on positions of all the points (and hence, whose query complexity is linear) and then tests whether P is an extreme point set or not (in the latter case we also return a subset of P which is non-convex). Observe that if P is in convex position, the query complexity of our algorithm (cf. Lemma 3.1) is $\mathcal{O}\left(\sum_{k=1}^{\lceil \log_2 n \rceil} n^{2/3} \cdot 2^{k/3}\right) + \mathcal{O}(n) = \mathcal{O}(n)$, which is clearly optimal. On the other hand, if P is ϱ-far from being in convex position, then with

a constant probability our algorithm will reject P for $\epsilon \geq \frac{1}{2}\varrho$. Therefore, the expected query complexity of our algorithm is $\mathcal{O}\left(\sum_{k=1}^{\lceil \log_2(1/\varrho)\rceil} n^{2/3} \cdot 2^{k/3}\right) = \mathcal{O}(n^{2/3}/I^{1/3})$, where I is the number of interior (i.e., non-extreme) points.

We can combine our algorithm with Chan's algorithm [5] to obtain an algorithm that is output-sensitive in h (the number of extreme points) and relative edit distance.

Property testing in computational geometry. Typical applications of property testing are program checking and input filtering. In many applications it is good and often necessary to ensure the correctness of results that are, for example, computed by a possibly unreliable library (e.g., a library that does not use exact precision arithmetic).

Although property testing does not give a 100% guarantee for the correctness, it does provide a good trade-off between running time and safety. A further advantage is the possibility to trade running time against algorithmic simplicity. Since our testers have sublinear running time we do not have to use optimal algorithms when we compute the structures needed while still achieving a sublinear run-time. In this setting especially the query complexity of an algorithm is important.

A particular application of property testing in computational geometry in the context of robustness is *lazy error correction*. It is well known that incrementally computing a complex structure (e.g., a Delaunay triangulation) with fixed precision arithmetic might lead to serious structural defects of the output. Furthermore, early small errors in the structure frequently lead to large structural defects in the final structure. To ensure that the structure is always close to the correct one, we run a property tester after a couple of updates. If the tester rejects, we use some lazy error-correction method (e.g., edge flips with exact arithmetic) to fix the structure and continue. At the end of the algorithm we fix the final structure. We believe this might become an important application of property testing in the context of computational geometry.

2 Disjointness of Generic Geometric Objects

We begin our investigations with a simple tester for the problem of testing whether a collection of objects is disjoint. The main reason to present this algorithm in details is to show the reader the flavor of many testers which run in two phases: (i) sample at random a sufficiently large subset of input objects and then (ii) analyze the sample objects to obtain information about the whole input. This approach works fairly easily for the problem studied in this section; other problems investigated in following sections require more complicated analysis.

The problem of deciding whether an arrangement of objects is intersection-free belongs to the most fundamental problems in computational geometry. A typical example is to verify whether a set of line segments in the plane or a set of hyper-rectangles or polytopes in $\mathbb{R}^d$ is intersection-free. We assume the oracle allows to query for the input objects and we suppose that there is an algorithm A that solves the exact decision problem of testing whether a set of objects is disjoint. Further, we consider the problem only for generic objects[3]. We shall use the following further definitions.

[3] That is, we never use any information about the geometric structure of the objects and our solution can be applied to any collection of objects (in any space, not necessarily metric one).

Definition 2.1. *Let $\mathbb{O}$ be a set of any objects in $\mathbb{R}^d$. We say $\mathbb{O}$ is* (pairwise) disjoint, *if no two objects in $\mathbb{O}$ intersect; $\mathbb{O}$ is ϵ-*far from being pairwise disjoint, *if there is no set $T \subseteq \mathbb{O}$ with $|T| \leq \epsilon\,|\mathbb{O}|$ such that $\mathbb{O} \setminus T$ is disjoint.*

Then the following algorithm is a simple ϵ-tester:

DISJOINTNESS (SET $\mathbb{O}$ consisting of n objects):
  Choose a set $S \subseteq \mathbb{O}$ of size $8\sqrt{n/\epsilon}$ uniformly at random
  Check whether S is disjoint using algorithm A
  if S is disjoint **then** *accept*
  else *reject*

Theorem 2.1. *Algorithm* DISJOINTNESS($\mathbb{O}$) *is an ϵ-tester with the* optimal *query complexity $\Theta(\sqrt{n/\epsilon})$. If the running time of the algorithm A for k objects is $T(k)$, then the running time of* DISJOINTNESS($\mathbb{O}$) *is $\mathcal{O}(T(8\sqrt{n/\epsilon}))$.*

Proof. We prove only that DISJOINTNESS($\mathbb{O}$) is an ϵ-tester and show the required upper bound for the query complexity. (The proof that every ϵ-tester has query complexity of $\Omega(\sqrt{n/\epsilon})$ is omitted.) Clearly, if $\mathbb{O}$ is intersection-free, the algorithm accepts $\mathbb{O}$. So let us suppose that $\mathbb{O}$ is ϵ-far from being intersection-free. In that case we can apply $\frac{1}{2}\,n\,\epsilon$ times the following procedure to $\mathbb{O}$: pick a pair of intersecting objects and remove it from $\mathbb{O}$. We can prove that with probability at least $\frac{2}{3}$ at least one of the pairs is chosen to S (because the objects from each pair intersect each other). Since the size of S is $8\sqrt{n/\epsilon}$, the upper bound for the query complexity follows.

3 Convex Position

In this section we consider one of the most classical properties of a set P of points: being in *convex position*. With respect to vertex representations of convex polytopes, it reflects the concept of "minimality" in that no point may be removed without affecting the convex hull of P. Many algorithms (such as the intersection test presented in Section 4) therefore require that their input be in convex position.

Definition 3.1. *A set P of n points in $\mathbb{R}^d$ is in* convex position *iff each point in P is an extreme point of the convex hull* $\mathrm{CH}(P)$. *We say P is ϵ-*far from being in convex position *if no set Q of size ϵn exists s.t. $P \setminus Q$ is in convex position.*

While it is possible to test whether a single point is extreme in $\mathcal{O}(n)$ time [6, Section 3], no such algorithm is known to compute *all* extreme points of a given set P. The fastest algorithm known due to Chan [5] uses data structures for answering many linear programming queries simultaneously and requires time

$$T(n,h) = n \cdot \log^{\mathcal{O}(1)} h + (n\,h)^{\frac{\lfloor d/2 \rfloor}{\lfloor d/2 \rfloor + 1}} \cdot \log^{\mathcal{O}(1)} n \ , \qquad (1)$$

where h denotes the output size (number of extreme points). It is also conjectured that the problem of testing whether a set of points P is in convex position is asymptotically as hard as the problem of finding all extreme points of P.

In this section we introduce two ϵ-testers for being in convex position. We begin with the algorithm that has asymptotically optimal query complexity.

CONVEX-A(P):
Choose a set $S \subseteq P$ of size $36 \cdot \sqrt[d+1]{n^d/\epsilon}$ uniformly at random.
Compute all h extreme points of S.
if $h < m$ **then** *reject*
else *accept*

Lemma 3.1. *Algorithm* CONVEX-A *is an ϵ-tester for the property of being in convex position. Its query complexity is $\mathcal{O}(\sqrt[d+1]{n^d/\epsilon})$ and it can be implemented to run in time $\mathcal{O}\big(T(\sqrt[d+1]{n^d/\epsilon}, \sqrt[d+1]{n^d/\epsilon})\big)$. Furthermore, its query complexity is asymptotically optimal in the sense that every ϵ-tester for convex position has the query complexity $\Omega(\sqrt[d+1]{n^d/\epsilon})$.*

Proof. It is easy to see that Algorithm CONVEX-A accepts sets in convex position. Suppose now that P is ϵ-far from being in convex position. Let $J = \frac{\epsilon\, n}{d+1}$. We can prove that there exist sets $W_j, U_j \subseteq P$, $j = 1, \ldots, J$ that satisfy the following conditions:

- each set W_j is of size $d+1$ and all sets W_j are pairwise disjoint,
- for each $u \in U_j$ the set $W_j \cup \{u\}$ is not in convex position,
- each W_j is pairwise disjoint with any U_i, and
- each U_j is of size greater than or equal to $n \cdot (\frac{1-\epsilon}{d+1} - \epsilon)$.

Fix these sets W_j and U_j, $1 \le j \le J$. We can prove that with probability at least $\frac{2}{3}$ there is some j such that for some $u \in U_j$ set S contains $W_j \cup \{u\}$. Since $W_j \cup \{u\}$ is not in convex position, it will be detected that S as well as P are not in convex position. Hence, Algorithm CONVEX-A is an ϵ-tester for the property of being in convex position.

The sample complexity follows easily from the bound on the size of S and the running time follows from the definition at the beginning of this section. Because of space limitations we omit here the proof that every ϵ-tester for convex position has the query complexity $\Omega(\sqrt[d+1]{n^d/\epsilon})$.

Now, we describe another algorithm for testing convex position which for $d \ge 4$ achieves better running times if the fastest known implementations are used.

CONVEX-B(P):
Choose a set $S \subseteq P$ of size $4/\epsilon$ uniformly at random.
for each $p \in S$ simultaneously
check whether p is extreme for $\mathrm{CH}(P)$
if p is not extreme for $\mathrm{CH}(P)$ **then exit** and *reject*
accept

Lemma 3.2. *Algorithm* CONVEX-B *is an ϵ-tester for the property of being in convex position. It can be implemented to run in time $\mathcal{O}\big(T(n, 1/\epsilon)\big)$.*

The following theorem summarizes the running time complexity of algorithms presented in this section. It follows by plugging Chan's bound (1) to Lemmas 3.1–3.2.

Theorem 3.1. *There is an ϵ-tester for convex position with the running time of order*

dimension $d = 2$	*dimension* $d = 3$	*dimensions* $d \ge 4$
$n^{2/3}\,(1/\epsilon)^{1/3}\,\log(n/\epsilon)$	$n^{3/4}\,(1/\epsilon)^{1/4}\,\log(n/\epsilon)$	$(n/\epsilon)^{\frac{\lfloor d/2 \rfloor}{\lfloor d/2 \rfloor + 1}}\,\log^{\mathcal{O}(1)}(n) + n\,\log^{\mathcal{O}(1)}(1/\epsilon)$

4 Disjointness of V-Polytopes

In this section we consider the property of whether two polytopes $\mathrm{CH}(R)$ and $\mathrm{CH}(B)$, represented by their vertices (so called V-Polytopes), are disjoint. Denote $n = |P|$ and let $P = R \cup B$.

Definition 4.1. *Two polytopes $\mathrm{CH}(R)$, $\mathrm{CH}(B)$ with finite points sets $R, B \subseteq \mathbb{R}^d$ in convex position are ϵ-*far from being disjoint*, if there is no set $V \subseteq R \cup B$, $|V| \leq \epsilon \cdot |R \cup B|$ such that $\mathrm{CH}(R \setminus V)$ and $\mathrm{CH}(B \setminus V)$ are disjoint.*

We propose the following simple ϵ-tester for disjointness of V-polytopes.

DISJOINTNESS (R, B):
Choose a set $S \subseteq P$ of size $\Theta((d/\epsilon)\ \ln(d/\epsilon))$ uniformly at random
Test whether $CH(R \cap S)$ and $CH(B \cap S)$ are disjoint
if $CH(R \cap S)$ and $CH(B \cap S)$ are disjoint **then** *accept*
else *reject*

Theorem 4.1. *Algorithm* DISJOINTNESS *is ϵ-tester for disjointness of V-polytopes with the query complexity $\mathcal{O}((d/\epsilon)\ \ln(d/\epsilon))$.*

Proof. Since the query complexity follows immediately from the upper bound for the size of S, we focus only on showing that DISJOINTNESS is a proper ϵ-tester. It is easy to see that if R and B are disjoint then DISJOINTNESS always accepts the input. So let us suppose that the input is ϵ-far from being disjoint. For any hyperplane h we denote by h^- and h^+ two halfspaces induced by h. Two convex polytopes are disjoint iff they can be separated by some hyperplane. Therefore, our goal is to ensure that with probability at least $\frac{2}{3}$ the sample set S contains, for every hyperplane h, a *"witness"* that h does not separate $\mathrm{CH}(R)$ from $\mathrm{CH}(B)$. It is easy to see that such a witness exists if we could ensure that for every hyperplane h set S contains two points a, b such that either (a, b) or (b, a) belong to one of the following four sets: $(R \cap h^-) \times (R \cap h^+)$, $(R \cap h^-) \times (B \cap h^-)$, $(R \cap h^+) \times (B \cap h^+)$, and $(B \cap h^-) \times (B \cap h^+)$. With our choice for the size of S, we can show that for at least one of these four sets, the both sets in the Cartesian product are of cardinality at least $\epsilon\, n/2$. Let such two sets be called *representative* for h. To complete the proof of the theorem we only must show that S intersects each representative set for *every* hyperplane h in $\mathbb{R}^d$. And this follows from the result on the randomized construction of ε-nets (with $\epsilon = 2\,\varepsilon$) [13]. Indeed, from the result due to Haussler and Welzl [13] it follows that the set S is with probability at least $\frac{2}{3}$ an $(\epsilon/2)$-net of each of R and B for the range space $(\mathbb{R}^d, \mathcal{H})$. Therefore, by the definition of ε-nets (see, e.g., [13]), S intersects every representative of each hyperplane h in $\mathbb{R}^d$.

One can also easily improve the query complexity of algorithm DISJOINTNESS in the special case when $d = 2$ and the input polygons are stored in a sorted array.

Theorem 4.2. *For all pairs of polygons in $\mathbb{R}^2$ represented by a sorted array there exists a deterministic ϵ-tester for disjointness of V-polytopes with the query complexity and the running time of $\mathcal{O}(1/\epsilon)$.*

5 Euclidean Minimum Spanning Tree

In this section we consider the problem to determine if a given input graph G is a Euclidean Minimum Spanning Tree (EMST). The vertices of G are labeled with their position in the plane. We have oracle access to each ith vertex v and the jth neighbor of v (with a special symbol indicating non-existance of such a neighbor). Since the degree of each vertex of the EMST is a constant (it is at most 5), in this model we may think of the graph represented by the adjacency list representation.

The distance measure for the EMST problem is defined as follows:

Definition 5.1. *Given a point set P of n points in general position in the plane and a graph G whose vertices are labeled with the points in P. G is said to be ϵ-far from being the Euclidean minimum spanning tree T, if G has edit distance at least $\epsilon\, n$ from T. The edit distance between G and T is the minimum number of edge deletions and insertions to construct T from G.*

In any testing algorithm it is very important to develop methods to reject inputs that are far away from the desired property. It is known that the longest edge in a cycle of the graph does not belong to the minimum spanning tree. We can therefore reject G, if we find a cycle in the complete Euclidean graph whose longest edge belongs to the input graph (we call this a 'bad' cycle). We start with a useful lemma about the EMST (which holds for any MST).

Lemma 5.1. *Let P be a point set in the plane and let $e = (p_1, p_2)$ be an edge of the EMST. Further, let P' be a subset of P and let $p_1, p_2 \in P'$. Then e belongs to the EMST of P'.*

The lemma above implies that we can reject G if we find a bad cycle in the complete Euclidean graph of *a subset* of P. Until the end of this paragraph we assume that G is ϵ-far from the EMST. Under the assumptions that (1) G is connected, (2) its straight-line embedding is crossing-free and (3) there are no crossings with edges of the correct EMST, we can show that there are many bad cycles in the complete Euclidean graph of P. In fact, there are many bad cycles even if G is only close to properties (1)-(3). Also note that these properties are necessary conditions for G to be an EMST. Therefore, our tester for the EMST runs in two phases. It first checks (with a property tester) whether G satisfies properties (1)-(3). If the input graph passes these tests but is ϵ-far from the EMST, it must have many "bad" cycles. We then run a tester that finds such a cycle w.h.p. and we are done.

We begin with testing whether an input graph is connected and its straight-line embedding is crossing-free. Additionally, we reject graphs with many vertices having degree larger than 5 (we omit this issue in the extended abstract).

Definition 5.2. *Let P be a set of n points in general position in the plane and let $G = (V, E)$ be a graph whose vertices are labeled with the points in P, $|V| = n$. Let S be the set of subsets E' of E s.t. $G' = (V, E \setminus E')$ is crossing-free. We say G is ϵ-far from being a* straight-line, crossing-free, connected embedding, *if $min_{E' \in S}\{|E'| +$ # connected components in $G'\} \geq \epsilon n$.*

For connectivity in graphs represented by the adjacency list there is a test developed in [11] which runs in $\mathcal{O}(\frac{1}{\epsilon})$ time. We combine this algorithm with the tester DISJOINTNESS($\mathbb{O}$) developed in Section 2 to obtain the following result.

Lemma 5.2. *The property of being a straight-line, crossing-free, connected embedding can be tested in $\mathcal{O}(\sqrt{\frac{n}{\epsilon}}\log n)$ time.*

We continue our studies with property (3). We need some further notation:

Definition 5.3. *Let P be a point set in the plane and G be a graph whose vertices are labeled with the points in P. The* EMST-completion *$C(G)$ of G is the straight-line embedding of G together with all segments of the EMST of P.*

We can view $C(G)$ either as a set of segments or as a labeled graph. In the following we will use these both interpretations. From now on we call an edge in $C(G)$ *red*, if it does not belong to the input graph and *blue* otherwise. Since G and the EMST are crossing-free, there can be only red blue intersections in $C(G)$. The following lemma shows that in order to detect a red-blue intersection, it is sufficient to find the blue segment and one endpoint of the red segment.

Lemma 5.3. *Let AB be a blue and CD be a red segment and let them intersect. Then AB is not in the EMST of any set containing either $\{A, B, C\}$ or $\{A, B, D\}$.*

One can show that if $C(G)$ is ϵ-far from (red-blue) intersection-free, then it is sufficient to sample a set of $\mathcal{O}(\sqrt{\frac{n}{\epsilon}})$ points to reject G. Therefore, our algorithm samples a random set S of $\mathcal{O}(\sqrt{\frac{n}{\epsilon}})$ points from P and adds the neighbors in the input graph of each point to S. Then it computes the subgraph G' of G induced by S. This can be easily done in $\mathcal{O}(|S|)$ time, if for each point in S its degree in G is constant. On the other hand, if we detect a vertex with degree larger than 5 we can immediately reject the input. After that we compute the EMST of S and then $C(G')$ in $\mathcal{O}(|S|\log|S|)$ time. Using a sweep-line algorithm we can check whether $C(G')$ is intersection-free in $\mathcal{O}(|S|\log|S|)$ time. We reject the input, if an intersection has been found. Repeating this procedure a constant number of times we achieve the desired $\frac{2}{3}$ probability.

If the input graph was not rejected so far, our goal is to find short cycles in $C(G)$ with at most two red edges. We call a cycle *bad*, if it has length at most $\frac{32}{\epsilon}$ and if it contains at most 2 red edges. (The number of short cycles with less than two red edges might be small and cycles with more than 2 red edges are more difficult to detect.)

Lemma 5.4. *Let $C(G)$ be the EMST-completion of G, let G be not $\frac{\epsilon}{100}$-far from a crossing-free, connected graph and let $C(G)$ have at most $\frac{\epsilon}{100}$ red-blue intersections. Then there are at least $\frac{\epsilon n}{12}$ bad cycles in $C(G)$.*

We find bad cycles by sampling a set of random directed edges and then we walk in both directions along the boundary of the face incident to each edge (we can think of G as a planar map).

Each of the at most $\frac{\epsilon n}{50}$ edges that can be removed to obtain an intersection-free embedding might destroy two bad cycles (we cannot walk along the boundary to close the face). Therefore, $\frac{\epsilon n}{12} - \frac{\epsilon n}{25} \geq \frac{\epsilon n}{25}$ bad cycles remain.

We distinguish between three types of bad cycles. Type i cycles, $0 \leq i \leq 2$, contain i red edges. Type 0 and type 1 cycles are easy to detect. It suffices to sample one of the boundary edges and our walk procedure will find the complete face (recall that the red edges are defined implicitly).

We classify the type 2 cycles according to the length of their longer blue chain into sets C_i. Each C_i contains all cycles whose longer chain has length l, $2^i \leq l < 2^{i+1}$. Each C_i is partitioned into subclasses $C_{i,j}$. $C_{i,j}$ contains all cycles whose shorter chain has length k, $2^j \leq k < 2^{j+1}$. For each $C_{i,j}$ we sample two different sets S_1 and S_2, one to detect the longer chain and one for the shorter one. Set S_1 has size $\frac{1}{l}\sqrt{\frac{n}{\epsilon}}$ and set S_2 has size $\frac{1}{k}\sqrt{\frac{n}{\epsilon}}$. The probability that a type 2 cycle in class $C_{i,j}$ is detected is $\mathcal{O}(\frac{kl \cdot |S_1| \cdot |S_2|}{n^2}) = \mathcal{O}(\frac{1}{\epsilon n})$. Therefore, the probability that any bad cycle is found is $(1 - \mathcal{O}(\frac{1}{\epsilon n}))^{\mathcal{O}(\epsilon n)} = O(1)$. Again amplification yields the desired bound of $\frac{2}{3}$. The query complexity of the algorithm is $\mathcal{O}(\sqrt{\frac{n}{\epsilon}}\log^2(\frac{1}{\epsilon}))$ and its running time is $\mathcal{O}(\sqrt{\frac{n}{\epsilon}}\log^2(\frac{1}{\epsilon})\log n)$.

Theorem 5.1. *There is a randomized ϵ-tester for the EMST with $\mathcal{O}(\sqrt{\frac{n}{\epsilon}}\log^2(\frac{1}{\epsilon}))$ query complexity and running time $\mathcal{O}(\sqrt{\frac{n}{\epsilon}}\log(\frac{1}{\epsilon})^2\log n)$.*

6 Delaunay Triangulation

We summarize our results for the Delaunay triangulation in the following theorems.

Theorem 6.1. *There is no sublinear property $\frac{1}{4}$-tester for Delaunay triangulations using Hamming distance like measure (edge deletion and insertion).*

Theorem 6.2. *If we define a triangulation to be ϵ-far from being Delaunay if there are at least $\epsilon\, n$ edges that can be flipped to improve the minimal angle locally, then there exists an ϵ-tester with the running time of $\mathcal{O}(1/\epsilon)$.*

7 Deterministic Lower Bounds

We present a simple technique to prove lower bounds for deterministic property test algorithms and apply it to the problems described in the previous sections. Our model for the deterministic algorithms is as follows. The input is given as an array of input items. The array has size n and we may access any item by its index in constant time.

Let A be a deterministic tester for property Q. Let I be an instance of size n with property Q. By definition A must accept I. Let $T(n)$ be the running time of A. Clearly, A can access at most $T(n)$ items of I. We color the accessed items red and all other items blue. Since A is deterministic, changing the blue items does not affect the outcome of the algorithm. Thus, if we can construct an instance that is ϵ-far from P by changing only blue items regardless how the red items are chosen by the algorithm then we could obtain a lower bound for the problem at hand. We can apply this approach to obtain lower bounds which are summarized in the following theorem.

Theorem 7.1. *There is no deterministic property $\frac{1}{4}$-tester with $o(n)$ query complexity for the following problems (using the distance measure defined in this paper and the relative edit distance for sorting): Sorting, Disjointness of Objects, Convex Position, Disjointness of Polytopes, and EMST.*

8 Acknowledgments

The second author would like to thank Stefan Funke for the helpful discussion about lazy error correction.

References

1. N. Alon, E. Fischer, M. Krivelevich, and M. Szegedy. Efficient testing of large graphs. In *Proc. 40th IEEE FOCS*, pp. 656–666, 1999.
2. N. Alon, M. Krivelevich, I. Newman, and M. Szegedy. Regular languages are testable with a constant number of queries. In *Proc. 40th IEEE FOCS*, pp. 645–655, 1999.
3. H. Alt, R. Fleischer, M. Kaufmann, K. Mehlhorn, S. Näher, S. Schirra, and C. Uhrig. *Algorithmica*, 8(5/6):365-389, 1992.
4. M. Blum, M. Luby, and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. *Journal of Computer and System Sciences*, 47(3):549–595, 1993.
5. T. M. Chan. Output-sensitive results on convex hulls, extreme points, and related problems. *Discrete & Computational Geometry*, 16:369–387, 1996.
6. M. Dyer and N. Megiddo. Linear programming in low dimensions. In J. E. Goodman and J. O'Rourke, eds., *Handbook of Discrete and Computational Geometry*, ch. 38, pp. 699–710, CRC Press, Boca Raton, FL, 1997.
7. F. Ergün, S. Kannan, S. Ravi Kumar, R. Rubinfeld, and M. Viswanathan. Spot-checkers. In *Proc. 30th ACM STOC*, pp. 259–268, 1998.
8. F. Ergün, S. Ravi Kumar, and R. Rubinfeld. Approximate checking of polynomials and functional equations. In *Proc. 37th IEEE FOCS*, pp. 592–601, 1996.
9. O. Goldreich, S. Goldwasser, E. Lehman, and D. Ron. Testing monotonicity. In *Proc. 39th IEEE FOCS*, pp. 426–435, 1998.
10. 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.
11. O. Goldreich and D. Ron. Property testing in bounded degree graphs. In *Proc. 29th ACM STOC*, pp. 406–415, 1997.
12. O. Goldreich and D. Ron. A sublinear bipartiteness tester for bounded degree graphs. *Combinatorica*, 19(3):335–373, 1999.
13. D. Haussler and E. Welzl. Epsilon-nets and simplex range queries. *Discrete & Computational Geometry*, 2:127–151, 1987.
14. K. Mehlhorn, S. Näher, M. Seel, R. Seidel, T. Schilz, S. Schirra, and C. Uhrig. Checking geometric programs or verification of geometric structures. *Computational Geometry: Theory and Applications*, 12:85–103, 1999.
15. R. Rubinfeld and M. Sudan. Robust characterization of polynomials with applications to program testing. *SIAM Journal on Computing*, 25(2):252–271, 1996.
16. R. Rubinfeld. Robust functional equations and their applications to program testing. In *Proc. 35th IEEE FOCS*, pp. 288–299, 1994.