Property Testing and Geometry

Dissertation of Christian Sohler

September 30, 2002

Acknowledgments

First of all, I would like to thank my thesis advisor Professor Friedhelm Meyer auf der Heide for his great support. He showed me in many helpful discussions how to improve my work and suggested in which direction my research should go. But he also left me the opportunity to go my own way.

I would also like to thank my 'co-advisor' Professor Artur Czumaj for doing an excellent job. Artur introduced me to property testing and suggested to consider it in the context of geometry. His encouragement and ideas were always helpful and it was a pleasure for me to work with him.

Then I would like to thank Valentina Damerow, Alexander May, Thomas Müller, Harald Räcke, and Martin Ziegler for carefully reading parts of this thesis and their helpful suggestions to improve its readability.

Finally, a special thanks to the anonymous person(s) who cheered me up with a gift in form of a 'SchlagerBild' music CD during the last phase of writing.

Contents

1 Introduction

Property Testing is the computational task to decide whether a given object (e.g., a graph, a function, or a point set) has a certain predetermined property (e.g., bipartiteness, monotonicity, or convex position) or is far away from every object having this property. If the input object neither has the property nor is far away from it, the algorithm may answer arbitrarily. In contrast to traditional algorithms a property testing algorithm does not get a complete description of the object under consideration as input. Instead, it has the ability to perform queries about the (local) structure of the object. This way it is possible that a property testing algorithm achieves its goal by looking only at a small (usually randomly selected) part of the whole object.

The type of query used by a property testing algorithm depends on the object under consideration. For example, if the object is a (dense) graph G then the algorithm may query for entries in the adjacency matrix of G. If the object is a function f, it may ask queries of the form: 'What is the value of $f(x)$?' where x is an element of the domain of f. For point sets the algorithm may ask for the position of the i-th point of the set (given an arbitrary, fixed, but unknown ordering among the points).

To specify if an object is far away from a property we need a *distance measure* between objects. The distance between two objects O_1 and O_2 is typically given by the 'fraction of object O_1' that has to be changed in order to obtain object O_2. For example, in the *adjacency matrix model* for graphs the distance between two graphs is the fraction of entries in the adjacency matrix on which the two graphs differ. The distance between functions is the fraction of domain elements on which the value of the two functions differs and the distance of points sets is the fraction of points that are contained in one set but not in the other. To determine when an object is far away from a property a *distance parameter* ϵ is introduced. Given a distance measure and a distance parameter ϵ we say that an object is ϵ-far from a property, if the distance to every object having the property is more than ϵ.

The quality of a property testing algorithm is measured by its *query complexity* and its running time. The query complexity is the number of queries the algorithm asks about the input. The running time is the time required for additional computations.

The concept of property testing was first explicitly formulated by Rubinfeld and Sudan mainly in the context of program checking [75]. The study of property testing for *combinatorial objects* with focus on graphs was initiated by Goldreich et al. [51]. In many follow-up papers the concept of property testing has been applied to different classes of objects including graphs, hypergraphs, matrices, formal languages, Boolean expressions, and point sets.

In this thesis we study geometric problems in the context of property testing and we

apply concepts from computational geometry to property testing problems. We start our investigations in Chapter 3 with the development of property testing algorithms for some fundamental geometric properties [A. Czumaj, *C. Sohler*, and M. Ziegler, Property testing in computational geometry, In: *Proceedings of the 8th Annual European Symposium on Algorithms (ESA)*, pages 155 - 166, 2000]. We show in particular, that disjointness of a set of n geometric objects can be tested with $\mathcal{O}(\sqrt{n/\epsilon})$ query complexity. Next we develop a property testing algorithm for the *convex position* property of point sets in the $\mathbb{R}^d$. A point set is in *convex position* if every point of the set is a vertex of its convex hull. Our testing algorithm has a query complexity of $\mathcal{O}\big(\sqrt[d+1]{n^d/\epsilon}\big)$. For both problems we show that our property testing algorithm is optimal with respect to its asymptotic query complexity. Next we consider the property that a geometric graph in $\mathbb{R}^2$ (a graph with vertex set equal to a point set in the $\mathbb{R}^2$) is a *Euclidean minimum spanning tree*. We develop a property testing algorithm with query complexity $\mathcal{O}(\sqrt{n/\epsilon}\log(n/\epsilon))$ for this problem. Our algorithm is designed in the *adjacency list model* for graphs and it uses an interesting non uniform sampling technique.

In the next chapter we present a general framework for property testing algorithms with one-sided error [A. Czumaj and *C. Sohler*, Abstract combinatorial programs and efficient property testers, In: *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, to appear, 2002]. Our framework is based on a connection of property testing with a new class of problems which we call *abstract combinatorial programs*. Informally, we show that a property can be tested efficiently, if for every problem instance (no matter, if the instance has the property or not) there is an abstract combinatorial program (of small dimension and width) that satisfies a *Feasibility Preserving* property and a *Distance Preserving* property. We apply our framework to a variety of classical combinatorial problems. We prove the testability of some clustering properties of point sets, of the reversal distance property of permutations, and the k-coloring property of graphs and hypergraphs [A. Czumaj and *C. Sohler*, Testing hypergraph coloring, In: *Proceedings of the 28th Annual International Colloquium on Automata, Languages and Programming (ICALP)*, pages 493 - 505, 2001]. Although for most of these properties a property testing algorithm has been known before, our framework can be used to simplify the correctness proofs and in most cases it slightly improves the analyzed query complexity. More important, it shows that our framework is fairly general and can be applied to a couple of different problems. In the last section of Chapter 4 we continue our investigations in the generality of the framework and show: If a *hereditary* graph property can be tested in time independent of the size of the graph (for constant ϵ), then there exists a proof for its testability in our framework. A graph property is called hereditary, if it is closed under taking induced subgraphs, i.e., if $G = (V, E)$ is a graph with a hereditary graph property then every subgraph of G induced by a set $S \subseteq V$ has the property, as well. Although the current formulation of the framework is purely combinatorial and there is no trace of the geometric origin, its development has been influenced by concepts from geometry such as the geometry of linear programming and LP-type problems [76].

In Chapter 5 we return to the development of specific property testing algorithms; this time under a different model of computation [A. Czumaj and *C. Sohler*, Property testing with geometric queries, In: *Proceedings of the 9th Annual European Symposium on*

Algorithms (ESA), pages 266 - 277, 2001]. While in the standard testing model considered in Chapter 3 a testing algorithm for properties of point sets is only allowed to query for the position of the i-th point of the set we now allow a certain kind of *geometric queries* when accessing the input point set. In this new model a testing algorithm may specify a query range R and ask for the i-th point within R. Depending on the type of query range allowed we have different models of computation. The queries we consider are supported by standard spatial data structures as they appear in many applications. We show that in the new model it is possible to design more efficient testing algorithms than in the model considered so far. In particular, we reconsider the convex position property and show that it can be tested with $\mathcal{O}(\log n/\epsilon)$ triangular range queries. Then we show that there is a property testing algorithm for a basic map labeling property. This algorithm uses $\mathcal{O}(1/\epsilon^3)$ rectangular range queries. Finally, we visit the clustering properties from Chapter 4 again.

In Chapter 6 we apply property testing in the context of moving objects [A. Czumaj, *C. Sohler*, Soft kinetic data structures, In: *Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 865 -872, 2001]. When we consider moving object like cars, robots, and mobile phones we do not know the future position of an object. In many situations an object is controlled and moved by a third party which is located outside of our system. Basically, the only query a system can guarantee to answer correctly is for the *current* position of an object. In Chapter 6 we develop a theoretical model for this kind of motion. We are interested in combinatorial structures that are uniquely defined by the current position of all objects. We assume the following: Queries about the structures we maintain occur subsequently in time. Between two queries the objects may move arbitrarily. This movement induces combinatorial changes in the maintained structure because the structure is uniquely defined by the objects' positions. Before each query a data structure may spend some time to 'adjust' its data. The time spent is compared to the number of 'combinatorial changes' that occurred in the data structure since the most recent query. More precisely, we assume that the objects moved on the 'shortest' (in terms of combinatorial changes) route to their current position. If we do not want to ask for the position of every single object before each query, there is no hope to maintain a combinatorial structure correctly. That is, where property testing can be applied. We only maintain approximately correct structures (in the property testing sense). If a structure is far from correct, then the property testing algorithm usually rejects and returns a small counter example. This counter example is then locally corrected. This procedure is applied until the property tester accepts. Using this approach we can show that we can maintain approximately correct data structures spending only slightly more time than the number of combinatorial changes in the structure during the shortest motion from configuration to configuration. We illustrate our approach on sorted sequences, binary search trees, range trees, and the Euclidean minimum spanning tree.

In the remainder of the Introduction we discuss our motivation to consider property testing in general, and in particular in the geometric context. Lateron we give a brief summary of the state of the art in property testing.

1.1 Motivation

Property testing can be viewed as a relaxation of a standard decision procedure: Instead of deciding whether an object has a given property or not, we only have to distinguish between the case when the object has the property and the case when the object is far away from it. This simpler problem can often be solved much faster and by looking only at a small - sometimes constant - part of the input. For this reason the running time of a property testing algorithm is usually much smaller than that of the corresponding decision procedure. Problems that are infeasible in their original formulation as a decision problem (e.g., $\mathcal{NP}$-complete problems) are tractable in the property testing setting. The standard way to deal with intractable problems would be, of course, to apply an approximation algorithm (in the classical sense) to the problem. But sometimes polynomial or even linear time approximation algorithm may also be too slow. In telecommunications, web traffic analysis, and data mining we have massive data sets that cannot be processed by classical algorithms. In such cases the only possibility is to use a sublinear time and space algorithm as provided by property testing.

We also observe that property testing is a natural form of approximation. Whether or not the approximation provided by a standard approximation algorithm is more useful than that of property testing depends on the problem at hand. For example, in the case of graph and hypergraph coloring it might be more useful to obtain a coloring that violates at most an ϵ-fraction of the edges than a coloring that might use many more colors than necessary.

We remark that property testing can be very useful in the field of program verification. In software development people often specify interfaces between different program parts. Even with a correct and complete interface specification it may happen (more often than not ?) that data is passed through the interface that is not consistent with the specification. Typically, data is passed internally using pointers and so it is possible to pass huge objects in constant time. Hence it is infeasible to check the whole object for its correctness without heavy impact on the running time of the system. In this case a property testing algorithm provides a useful alternative: If we cannot give a full guarantee that the input is correct then we can at least detect the case when it is far from correct.

A further reason for the research in property testing algorithms is the fact that in some situations being close to a property is almost as good as having the property. This can be seen in the following example from Computer Graphics: One of the major problems in Computer Graphics is to design efficient *rendering algorithms*. Rendering is the computational task to display a virtual scene on the screen. Such a virtual scene is typically composed of several millions of polygons. Using nowadays graphics hardware and a standard z-buffer algorithm it is not possible to render all polygons in a reasonable time. Therefore, it is necessary to determine in advance if certain parts of the scene cannot be seen from the current point of view. This process is called *occlusion culling*. Unfortunately, it is very difficult to compute exactly those polygons that cannot be seen from a certain point of view. Many approaches have been developed to deal with this problem and some of them work only for objects that have (roughly) a convex shape (see, e.g., the survey [26]). If we could determine quickly in advance whether an object is convex or far away from convex then we could quickly decide if it is useful for the purpose of (online) occlusion culling.

Such a service can be provided by a property testing algorithm. Therefore we consider the closely related problem, if a point set (possibly the set of vertices of an object used for culling purposes) is in convex position, in Chapter 3 and 5 under different property testing models.

One particular situation when property testing can be useful in Computational Geometry is the exact computation of geometric structures. It is well-known that fixed precision arithmetic leads to serious problems in the design of geometric algorithms. Therefore, software packages provide efficient algorithms to evaluate geometric predicates exactly. Unfortunately, precision takes its time and so the exact evaluation of geometric predicates is usually much slower than what we could achieve by using fixed precision arithmetic. One approach to deal with these problems is the following: We first compute the geometric structure using fixed precision arithmetic and then verify the final structure using exact arithmetic. If we find errors we try to fix them locally. Unfortunately, such a heuristic has the drawback that an early error in the computation may cause damage to the whole part of the structure computed later. Therefore, we should make sure that our structure is not 'too far away' from the correct structure during the whole computation. This could be achieved by a property testing algorithm. If the property testing algorithm rejects the current structure, then we can correct it before we finish the computation.

Another situation when property testing is useful is in the case of continuously changing data. Continuously changing data arises in applications that deal with moving objects, e.g., when we want to maintain a mobile ad-hoc network for moving robots. In such a situation we can query each robot for its current position but such a query may be slow and each query increases the network load. It also does not make sense to update the position of each robot before a transmission using the network is performed - such an update would often create much more network load than the transmission itself. Instead we may use property testing techniques to check the structure of the current network. If we find errors in the structure we then correct them locally. In Chapter 6 we develop a theoretical framework for mobile data and we show how to maintain a certain type of approximate range trees. These range trees may be useful for the problem of maintaining such a communication network.

1.2 Related Work

We now give a brief summary of the state of the art in property testing. Property testing was first explicitly formulated by Rubinfeld and Sudan in the context of program checking [75]. In [51] Goldreich, Goldwasser, and Ron initiated the study of property testing for combinatorial objects. Since then, people designed property testing algorithm for a wide variety of problems and tried to classify large classes of testable properties. In the following we try to summarize the known results in the different areas where property testing has been applied. We say that a property can be *tested efficiently*, if there is a property testing algorithm with a query complexity that does not depend on the size of the input (is constant, if the distance parameter ϵ is a constant).

Graph, Hypergraph, and Matrix Properties. Graph properties have been extensively studied in the context of property testing. There are essentially two models for testing graph properties. The *adjacency matrix model* for dense graphs and the *adjacency list model* for sparse graphs. In the adjacency matrix model introduced in [51] it is known that properties that can be formulated as a certain class of graph partitioning properties can be tested efficiently [51]. Among these properties are clique size, cut size, bisection, and k-colorability properties of graphs. Colorability properties have received much attention in subsequent work. The analysis from [51] for the bipartiteness and the (standard) k-colorability property has been improved in [6]. In [5] Alon et al. used a variant of Szemerédi's Regularity Lemma to show that a very general graph coloring property can be tested efficiently. They use this result to prove that every first order logic graph property without $\forall\exists$ quantification is efficiently testable. Though the query complexity of their algorithm is independent of the size of the graph its dependency on ϵ is enormous (i.e., a tower of towers of a polynomial in ϵ). The notion of coloring from [5] has been further generalized in various ways in [42] and shown to be efficiently testable.

Another general property considered in the research is the property of H-freeness (graphs not containing a fixed graph H as an induced subgraph). Alon proved that the query complexity of a one-sided error property tester is polynomial, if and only if H is bipartite [3]. The property of H-freeness can also be tested efficiently in 3-uniform hypergraphs [62].

In the adjacency matrix model there is always a *canonical* property testing algorithm as the following result of Goldreich and Trevisan shows: A property in the adjacency matrix model can be tested with query complexity independent of the size of the graph, if and only if there exists a property testing algorithm that samples a set S of vertices of size independent of the graph uniformly at random and that accepts if and only if the subgraph induced by S has some fixed graph property (possibly different from the tested one) [54]. In this work the authors also characterize graph partition problems that are efficiently testable with one-sided error.

The adjacency list model for graphs has been introduced in [52] by Goldreich and Ron. They show that some graph properties like connectivity, planarity, and cycle-freeness can be tested efficiently. In contrast to the adjacency matrix model in the adjacency list model there is no canonical property testing algorithm known. Typically, non-uniform sampling strategies that depend on the problem at hand have been applied. For this reason the properties considered in the adjacency list model seem to be structurally simplier than the ones analyzed in the adjacency matrix model. Colorability properties have also been considered in the adjacency list model: In [53] Goldreich and Ron showed that bipartiteness can be tested with a query complexity of $\mathcal{O}\left(\mathit{poly}((\log n)/\epsilon)\sqrt{n/\epsilon}\right)$ using a non-uniform sampling strategy based on random walks. The authors also present an almost matching lower bound on the query complexity. Very recently, Bogdanov et al. [19] presented a lower bound of $\Omega(n)$ (for sufficiently small ϵ) on the query complexity of 3-colorability in the adjacency list model. It is also known that the diameter of graphs [67] and acyclicity of directed graphs is testable in sublinear time [15].

The popularity of the adjacency matrix model for graphs lead to the question which matrix properties can be tested efficiently. Fischer and Newman showed that matrix prop-

erties defined by a finite collection of forbidden induced partially ordered sets (where we view a matrix as a partially ordered set in the standard way, that is, as a product order of total orders) are efficiently testable [45]. Parnas and Ron developed property testing algorithms for the problem to test whether a given matrix is a Euclidean metric, a tree metric, or an ultrametric [68].

Properties of Functions. Monotonicity properties of functions have also received a lot of attention. Monotonicity of binary functions $f : \{0,1\}^n \to \{0,1\}$ can be tested with a query complexity of $\mathcal{O}(n/\epsilon)$ [50]. This result has been generalized to functions over arbitrary finite ordered sets in [37]. Sortedness of a sequence of n numbers can be tested in $\mathcal{O}(\log n/\epsilon)$ time [40]. Lower bounds on monotonicity testing have been given in [44].

Properties of Boolean functions that can be tested efficiently are singletons, monomials, and the property that f is a monotone DNF formula with at most ℓ terms. [70]. In [43] Fischer et al. proved that the property that a Boolean function over n variables depends only on k of them can be tested with a number of queries depending only on k and ϵ.

Language Properties. In order to understand the nature of efficiently testable properties people considered property testing in the context of formal languages. Alon et al. showed that a regular language seen as a property can be tested efficiently [7] and there exists a context-free language that cannot be efficiently tested. Further, there is a property testing algorithm for testing Dyck languages (languages containing strings of balanced parenthesis) with query complexity $\widetilde{\mathcal{O}}(n^{2/3}/\epsilon)$ [69]. Newman proved that constant width oblivious read-once branching programs viewed as properties can be tested efficiently [66]. Recently, Fischer and Newman showed that read-twice constant width branching programs are not necessarily testable [46].

Other Results. There are a couple of results on specific topics that do not fit into one of the three categories above. We summarize them in this paragraph.

Alon and Shapira showed that a general satisfiability problem which includes hypergraph k-coloring can be tested efficiently [8].

In the context of clustering Alon et al. considered the radius and diameter clustering problem, i.e. the problem if a given point set can be partitioned into k subsets such that the radius of the smallest sphere enclosing each set (the maximum distance between two points within each set, respectively) is at most b. If we view this problem as a property of point sets then it can be tested efficiently as proved in [4].

Some results related to statistics have been obtained: In [14] Batu et al. gave a property testing algorithm with query complexity $\mathcal{O}(n^{2/3}/\epsilon^{-4}\log n)$ for closeness of distributions. Given a distribution A over $[n] \times [m]$ it can be tested with query complexity $\mathcal{O}(n^{2/3}m^{1/3}\mathit{poly}(\epsilon^{-1}))$ if the distributions obtained by projecting A to each coordinate are independent [13].

Recently, Buhrman et al. introduced the notion of property testing to quantum computing [21].

Surveys. There are three surveys that summarize some of the work done in the field of property testing [42, 74, 49].

2 Preliminaries

In this chapter we introduce some notation used throughout this thesis. In particular, we introduce a formulation of the standard property testing model with one sided error. We introduce a special class of properties which we call combinatorial properties.

Then we show that if there exists a property tester for a combinatorial property of point sets then there exists a property tester with the same query complexity that picks a sample set uniformly at random and decides based on the sample and its internal coin flips. Such a result is used for lower bound constructions for the query complexity of a property testing algorithm.

Finally, we prove two auxiliary lemmas. The first lemma gives a lower bound on the probability that a sample taken uniformly at random from a set Ω contains one of k disjoint sets of size l. The second lemma gives an upper bound on that probability.

2.1 The Standard Testing Model

We begin with some basic notation and definitions. We use the $\widetilde{\mathcal{O}}$-notation to hide polylogarithmic factors, i.e. we have $\mathcal{O}(n \operatorname{polylog}(n)) = \widetilde{\mathcal{O}}(n)$. We write $[n] := \{1, \ldots, n\}$ for the set of integer numbers between 1 and n. Throughout this thesis, let $\mathcal{D}$ denote a finite set called *domain* and let $\mathcal{R}$ denote a (possibly infinite) set called *range*. Further let $\mathcal{F}$ denote a set of functions from $\mathcal{D}$ to $\mathcal{R}$. For a subset $S \subseteq \mathcal{D}$ and a function $f \in \mathcal{F}$ let $f_{|S}$ denote the *restriction* of f to S. That is, $f_{|S} : S \to \mathcal{R}$ with $f_{|S}(x) = f(x)$ for all $x \in S$. Now we define a property of $\mathcal{F}$ as a set of functions from $\mathcal{F}$:

Definition 2.1.1 *A set $\Pi \subseteq \mathcal{F}$ is called a* property.

As already mentioned in the introduction we also need a distance measure between functions in $\mathcal{F}$ to define a property testing problem. In general, such a distance measure can be an arbitrary function $\sigma : \mathcal{F} \times \mathcal{F} \to [0, 1]$:

Definition 2.1.2 *Given a distance measure σ between functions in $\mathcal{F}$ and a real number ϵ, $0 \leq \epsilon < 1$, we say a function $f \in \mathcal{F}$ is* ϵ-far from (having a property) Π *if $\sigma(f, g) > \epsilon$ for every function $g \in \Pi$.*

Typically, we define the distance between two functions $f, g \in \mathcal{F}$ as the fraction of domain elements on which the two functions differ (see, for example, [51]). Therefore, we denote this distance measure as the *standard distance measure*:

Definition 2.1.3 *Given two functions* $f, g \in \mathcal{F}$ *we define* the standard distance measure σ_1 *for functions in* $\mathcal{F}$ *as:*

$$\sigma_1(f,g) = \frac{|\{x \in \mathcal{D} : f(x) \neq g(x)\}|}{|\mathcal{D}|} \quad .$$

The goal of property testing is to develop efficient *property testers*. A property tester for Π is an algorithm that gets a distance parameter ϵ and a (possibly implicit) description of $\mathcal{D}$ (for example, when $\mathcal{D}$ is the set of numbers from $[n]$ it gets n). The property tester has *oracle access* to the input function f (for each $x \in \mathcal{D}$ it may ask queries of the form: 'What is the value of $f(x)$?'). A property tester must

- accept every function $f \in \Pi$, and
- reject every function f that is ϵ-far from Π with probability at least $\frac{2}{3}$.

[1] Notice that *if* $f \notin \Pi$ *and* f *is not* ϵ*-far from* Π*, then the outcome of the algorithm can go either way*.

Complexity of Property Testers. There are two types of possible complexity measures for property testers: The *query complexity* and the *running time*. The query complexity measures the number of queries asked by a property testing algorithm:

Definition 2.1.4 *The number of queries to the oracle is the* query complexity *of the property tester.*

If one counts also the time the algorithm needs to perform other tasks than querying the input function values (for example, to compute certain combinatorial structures or to compute a coloring of a graph), then the obtained complexity is called the *running time* of the property tester. Now we present two examples how objects other than functions can be represented in our model.

Point Sets. When we consider point sets we set $\mathcal{D} = [n]$. Then we can represent a point set $P = \{p_1, \cdots, p_n\}$ in $\mathbb{R}^d$ by a function $f : [n] \to \mathbb{R}^d$ with $f(i) = p_i$ for $1 \leq i \leq n$. Throughout the thesis we assume that the input point set is in *general position* when we deal with property testing algorithms for point sets. General position means that our point set is not in a *degenerate* configuration. Roughly speaking, we call a configuration of points degenerate, if it occurs with probability 0 when the points are subject to small random perturbations.

To be more precise, when we consider the *convex position* property as defined in Chapter 3 then a point set P in the $\mathbb{R}^d$ is degenerate, if there are more than $k+1$ points in a k-dimensional affine subspace with $k < d$. When we consider the Euclidean minimum spanning tree, then a point set is degenerate, if the point set does not have a unique minimum spanning tree or if it has a subset that does not have a unique minimum spanning tree.

[1] We consider a *one-sided error* model, though in the literature also a *two-sided error* model has been considered. In the two-sided error model the goal is to distinguish with probability at least $\frac{2}{3}$ between the case $f \in \Pi$ and the case of f being ϵ-far from Π.

Graphs. In the literature one can find three models for property testing in graphs. The first (and most widely considered) model is called the *adjacency matrix model* [51]. Here one assumes that a graph $G = (V, E)$ is represented as an adjacency matrix, i.e. it is possible to query for each pair (v, u) with $v, u \in V$, if $(v, u) \in E$ or not. Similarly, one can assume that the graph is represented by a function $f : V \times V \to \{0, 1\}$ that encodes the adjacency matrix. W.l.o.g., we assume that $V = [n]$. Hence when we set $\mathcal{D} = [n] \times [n]$ and $\mathcal{R} = \{0, 1\}$ then we can represent graphs as functions from $\mathcal{D}$ to $\mathcal{R}$ as required.

The second model is the *bounded length adjacency list model* [52]. Here one assumes that a degree bound d on the maximum degree of the vertices in the input graph is known. In this case a graph is represented as a function $f : [n] \times [d] \to [n] \cup \{0\}$ where $f(i, j)$ denotes the j-th neighbor of vertex i and where $f(i, j) = 0$, if such a neighbor does not exist.

The third model is the *unbounded length adjacency list model* [67]. In this model, the input is not represented as a function. The model is specified by the different queries that may be asked about the input graph $G = (V, E)$. First of all, the algorithm may ask for the degree $deg(v)$ of each vertex $v \in V$. Further it may ask for the i-th neighbor of vertex v for each $i \leq deg(v)$.

2.2 Combinatorial Properties

We now want to introduce a special class of properties that plays an important role in property testing. We refer to these properties as *combinatorial properties*. Combinatorial properties are only defined for functions whose domain $\mathcal{D}$ is (isomorphic to) a set of the form $[n]^d$ for some constant $d > 1$. They are properties that are closed under permutations (isomorphisms) of the domain. This is typically the case when the function is used to represent a set of objects or a combinatorial structure such like a graph or hypergraph.

Definition 2.2.1 *Let $\mathcal{F}$ be the set of functions with domain $\mathcal{D} = [n]^d$ and arbitrary range $\mathcal{R}$. Let $\mathcal{S}_n$ denote the set of permutations of $[n]$. A property Π of $\mathcal{F}$ is called* combinatorial, *if it is closed under permutations of $[n]$. That is, if $\pi \in \mathcal{S}_n$ is a permutation of $[n]$ then for every $f \in \Pi$ it holds: $\pi(f) \in \Pi$ where $\pi(f)(i_1, \ldots, i_d) := f(\pi(i_1), \ldots, \pi(i_d))$ for $i_1, \cdots, i_d \in [n]$.*

Let us consider a few examples for combinatorial properties. If the considered object is a *set of basic objects* (that is, there is no ordering among the objects) then every property must be combinatorial. This is because the property does not depend on the way the object is represented as a function. Examples for combinatorial properties of sets of objects are the disjointness property considered in Chapter 3 or the clustering properties of point sets in Chapter 4. Other examples are all graph and hypergraph properties (if viewed as properties of an n-vertex graph or hypergraph).

Examples for properties that are not combinatorial are properties like *sortedness*, *convex polygons*[2], and the *reversal distance* property considered in Chapter 4. These properties typically depend heavily on the ordering given implicitly by the representation of

[2]The property that a sequence of points forms a simple convex polygon.

the input object as a function. When we design property testers for these properties we sometimes require non-uniform sampling techniques.

For combinatorial properties of point sets we show in the following section that there is no chance to improve the query complexity of a property tester by using adaptive sampling. If we consider combinatorial properties of point sets and we have an arbitrary property tester with query complexity $q(n, \epsilon)$ then there is a property tester that samples a set $S \subseteq [n]$ of size $q(n, \epsilon)$ uniformly at random and decides based on S and its internal coin flips.

2.3 The Power of Uniform Sampling

We now prove a fundamental lemma that is needed for lower bound constructions. We show that if there is a property tester for a combinatorial property Π of point sets with query complexity $q(\epsilon, n)$ then there is a property tester for Π that picks a sample set $S \subseteq P$ of size $q(\epsilon, n)$ uniformly at random and decides based on S and its internal coin flips. The proof is almost similar to the proof of Lemma 4.1 in the paper [54] by Goldreich and Trevisan which is based on an analogous statement proven by Bar-Yossef et al. in [11].

The idea of the proof is simple: There are $n!$ different representations of the same point set, one for each permutation of $[n]$. When we start our algorithm on a random representation of the same set then choosing the next index adaptively simply means to pick another element uniformly at random. Indeed we cannot gain anything using adaptive sampling:

Lemma 2.3.1 *Let Π be an arbitrary combinatorial property of point sets and let $\mathbb{A}$ be an arbitrary property tester for Π with query complexity $q(\epsilon, n)$. Then there exists a property tester for property Π that selects a set S of $q(\epsilon, n)$ points uniformly at random and decides based on S and its internal coin flips.*

Proof : Recall that a point set $P = \{p_1, \cdots, p_n\}$ in the $\mathbb{R}^d$ is represented by a function $f : [n] \to \mathbb{R}^d$ with $f(i) = p_i$. Since Π is a combinatorial property we know that we can talk about properties of P rather than properties of the function representing P. That is, if one representation of P has property Π then every other representation also has property Π. We use this fact in the following to construct a property tester that samples uniformly at random from an arbitrary property tester for Π. Both property testers have the same query complexity.

Let $\mathbb{A}$ be an arbitrary property tester for Π. Wlog., we assume that algorithm $\mathbb{A}$ operates in iterations. In each iteration, depending on its internal coin flips and the answers obtained in the past iterations, the algorithm selects a new index i and makes a query for the position of the point p_i.

Now we obtain a new algorithm $\mathbb{A}'$ from $\mathbb{A}$ in the following way: When $\mathbb{A}'$ is started with input point set P of size n (given as an oracle) it first choses a permutation π of $[n]$ uniformly at random. Then algorithm $\mathbb{A}$ is invoked with oracle access to the point set $\pi(P)$ that is represented by the function $f_\pi : [n] \to \mathbb{R}^d$ defined by $f_\pi(i) = f(\pi(i))$. We observe

that $\pi(P)$ has property Π, if and only if P has property Π since $\pi(P) = P$ and since Π is combinatorial. Further we know that $\mathbb{A}$ is a property tester for property Π. Thus it must accept, if $\pi(P)$ has property Π. It follows that $\mathbb{A}'$ accepts, if P has property Π.

If P is ϵ-far from Π then so is $\pi(P)$. By the fact that $\mathbb{A}$ is a property tester it follows that $\pi(P)$ is rejected by $\mathbb{A}$ with probability at least $2/3$. It follows that $\mathbb{A}'$ also rejects P with probability at least $2/3$. Thus we know that $\mathbb{A}'$ is a property tester for property Π.

Our next step is to show that in each iteration of algorithm $\mathbb{A}'$ the next chosen point is uniformly distributed among all possible choices (among all points that have not been chosen in a previous iteration). For every sequence r of coin flips let $\mathbb{A}_r$ denote the deterministic algorithm obtained from $\mathbb{A}$ by fixing the outcome of the coin flips according to r. Further let $\mathbb{A}'_r$ denote the algorithm similar to $\mathbb{A}'$ with the exception that $\mathbb{A}_r$ is invoked instead of $\mathbb{A}$. We show that for every sequence r of coin flips and for every possible sequence of queries and answers the choice of the point selected next by algorithm $\mathbb{A}'$ is uniformly distributed among the points not selected so far.

Let $i^\pi_{1,r}, \cdots, i^\pi_{j,r}$ denote the indices selected by $\mathbb{A}'_r$ in iterations 1 to j, i.e., the indices selected when $\mathbb{A}_r$ is invoked with oracle access to the point set $\pi(P)$. Further let $q^\pi_{1,r}, \cdots, q^\pi_{j,r}$ denote the points selected by $\mathbb{A}'_r$ in iterations 1 to j, i.e., $q^\pi_{\ell,r} = f_\pi(i^\pi_{\ell,r})$ for $1 \leq \ell \leq j$. The choice of the index $i^\pi_{j+1,r}$ selected by $\mathbb{A}'_r$ in iteration $j+1$ depends only on the previously selected points $q^\pi_{1,r}, ..., q^\pi_{j,r}$. Thus when we condition π by $f_\pi(i^\pi_{\ell,r}) = q^\pi_{\ell,r}$, $1 \leq \ell \leq j$, the choice of index $i^\pi_{j+1,r}$ is deterministic. Further we prove in the following claim that for every fixed index $i^\pi_{j+1,r}$ under the same conditioning the point $f_\pi(i^\pi_{j+1,r})$ is uniformly distributed among the points not selected so far:

Claim 2.3.2 *Let $i_1, \cdots, i_j \in [n]$ be distinct indices and let $q_1, \cdots, q_j \in P$ be the distinct points. Then for each $i \in [n] \setminus \{i_1, \cdots, i_j\}$ and each $p \in P \setminus \{q_1, \cdots, q_j\}$:*

$$\mathbf{Pr}[f_\pi(i) = p \mid f_\pi(i_\ell) = q_\ell, 1 \leq \ell \leq j] = \frac{1}{n-j}$$

.

Proof : The number of permutations of $[n]$ with $f_\pi(i_\ell) = q_\ell$ for $1 \leq \ell \leq j$ is $(n-j)!$. For each $i \in [n] \setminus \{i_1, \cdots, i_j\}$ and each $p \in P \setminus \{q_1, \cdots, q_j\}$ the number of permutations of $[n]$ with $f_\pi(i) = p$ and $f_\pi(i_\ell) = q_\ell$ for $1 \leq \ell \leq j$ is $(n-j-1)!$. Thus

$$\mathbf{Pr}[f_\pi(i) = p \mid f_\pi(i_\ell) = q_\ell, 1 \leq \ell \leq j] = \frac{(n-j-1)!}{(n-j)!} = \frac{1}{n-j}$$

. □

Thus we know that in each iteration of algorithm $\mathbb{A}'_r$ the next chosen point is uniformly distributed among the points not chosen so far. We use this fact to build an algorithm $\mathbb{A}''$ that has the same output distribution as $\mathbb{A}'$. Then we design an algorithm $\mathbb{A}'''$ that samples a set $S \subseteq P$ of $q(\epsilon, n)$ points uniformly at random and then decides based on internal coin flips and the point positions. We observe that we can execute algorithm $\mathbb{A}'$ by first choosing a sequence r of coin flips uniformly at random and then invoking algorithm $\mathbb{A}'_r$. Algorithm $\mathbb{A}''$ works in a similar manner. It first chooses a sequence r of coin flips

uniformly at random. Then it invokes algorithm $\mathbb{A}''_r$. Algorithm $\mathbb{A}''_r$ behaves similarly to algorithm $\mathbb{A}'_r$ with the exception that in each iteration when $\mathbb{A}'_r$ requests the position of the point with index i algorithm $\mathbb{A}''_r$ selects uniformly at random an index not selected so far and returns the corresponding point. Let us formalize this point and look at iteration j of both algorithms. Let i_j denote the index selected by algorithm A'_r in iteration j and let i'_j denote the index selected by algorithm A''_r. When during iteration $j+1$ algorithm $\mathbb{A}'_r$ requests the position of the point stored at index i_{j+1} then $\mathbb{A}''_r$ selects an index i'_{j+1} uniformly at random from the set $[n] \setminus \{i'_1, \cdots, i'_j\}$ and answers the query with the position of point i'_{j+1}.

Claim 2.3.3 *Let P be any point set in the $\mathbb{R}^d$. If algorithms $\mathbb{A}''$ and $\mathbb{A}'$ are invoked with oracle access to the same function representing P then their output distributions are identical.*

Proof : Let us fix the function representing P and r (for both algorithms). We show by induction on the number of iterations that the output distribution of algorithms $\mathbb{A}''_r$ and $\mathbb{A}'_r$ are identical. We already proved for a fixed selection of points in iterations 1 to j that the next point chosen by algorithm $\mathbb{A}'_r$ is uniformly distributed among all points not selected so far. Clearly, the same is true for algorithm $\mathbb{A}''_r$ (if the distribution of the indices is uniform then so is the distribution of the corresponding points). Thus the induction step holds and thus the output distributions of both algorithms are identical. □

We have already proven that $\mathbb{A}'$ is a property tester for property Π. Since the output distributions of $\mathbb{A}''$ and $\mathbb{A}'$ are identical for every input point set we have shown that algorithm $\mathbb{A}''$ is also a property tester. But $\mathbb{A}''$ uses a non-adaptive sampling. Finally, we show that there is an algorithm $\mathbb{A}'''$ that samples a set S of $q(\epsilon, n)$ points uniformly at random and decides based on S and its internal coin flips.

Algorithm $\mathbb{A}'''$ first samples a set S of $q(\epsilon, n)$ indices uniformly at random and then chooses a permutation $(i_1, \cdots, i_{q(\epsilon,n)})$ of S uniformly at random. It then runs algorithm $\mathbb{A}''$ and answers the j-th query of algorithm $\mathbb{A}''$ by returning the point stored at index i_j.

Claim 2.3.4 *Let P be any point set in $\mathbb{R}^d$. If algorithms $\mathbb{A}'''$ and $\mathbb{A}''$ are invoked with oracle access to the same function representing P then their output distributions are identical.*

Proof : Again we fix the function representing point set P and the random choice of r. We show by induction on the number of iterations that both algorithms have the same output distribution. It suffices to show that the choice of the index in iteration $j+1$ is uniformly distributed among the indices not chosen so far. For every index $i \in [n] \setminus \{i_1, \cdots, i_j\}$ it holds that

$$\mathbf{Pr}[i_{j+1} = i \mid i_\ell \neq i \text{ for } 1 \leq \ell \leq j] = \frac{q(\epsilon,n) - j}{n - j} \cdot \frac{(q(\epsilon,n) - j - 1)!}{(q(\epsilon,n) - j)!} = \frac{1}{n - j}$$

Thus the index is chosen uniformly at random from the set of remaining indices. Hence, the output distributions of algorithms $\mathbb{A}'''$ and $\mathbb{A}''$ are identical. □

Algorithm $\mathbb{A}'''$ satisfies the statement of Lemma 2.3.1 and thus the proof is completed. □

2.4 Two Probability Lemmas

The first lemma shows that if we take a sample of size s from a set Ω of n elements, then with good probability we have at least one of k disjoint sets (each of size l) in our sample.

Lemma 2.4.1 *Let Ω be an arbitrary set of n elements. Let k and l be arbitrary integers (possibly dependent on n) and let s be an arbitrary integer such that $s \geq \frac{2n}{(2k)^{1/l}}$. Let $W_1, \ldots, W_k$ be arbitrary disjoint subsets of Ω each of size l. Let S be a subset of Ω of size s which is chosen independently and uniformly at random. Then*

$$\mathbf{Pr}\left[\exists j \in [k] \ : \ (W_j \subseteq S)\right] \geq \frac{1}{4} \ .$$

Proof : We first observe that we can focus on the case $k \leq \frac{n}{2}$, because if $k > \frac{n}{2}$, then every set W_i contains exactly one element and then we immediately get $\mathbf{Pr}[\exists j \in [k] \ : \ (W_j \subseteq S)] \geq \frac{1}{2}$. Further, since $k \leq \frac{n}{2}$ yields $l + \frac{n-l}{(2k)^{1/l}} \leq \frac{2n}{(2k)^{1/l}}$, it is sufficient to consider only the case $s = l + \frac{n-l}{(2k)^{1/l}}$. Next, by Boole-Benferoni inequality (see, e.g., [65, Proposition C.2]) we obtain

$$\mathbf{Pr}\left[\exists j \in [k] \ : \ (W_j \subseteq S)\right] \ \geq \ \sum_{j=1}^{k} \mathbf{Pr}[W_j \subseteq S] \ - \sum_{1 \leq i < j \leq k} \mathbf{Pr}[(W_i \cup W_j) \subseteq S] \ .$$

Further, we observe that for every $j \in [k]$ it holds that

$$\mathbf{Pr}[W_j \subseteq S] = \frac{\binom{n-l}{s-l}}{\binom{n}{s}} = \frac{(n-l)!}{(s-l)!\,(n-s)!} \cdot \frac{s!\,(n-s)!}{n!} = \frac{(n-l)!\,s!}{n!\,(s-l)!} = \prod_{r=0}^{l-1} \frac{s-r}{n-r} \ .$$

Similar arguments can be used to show that for every $i, j \in [k]$, if $i \neq j$, then it holds that

$$\mathbf{Pr}[(W_i \cup W_j) \subseteq S] = \frac{\binom{n-2l}{s-2l}}{\binom{n}{s}} = \prod_{r=0}^{2l-1} \frac{s-r}{n-r} = \prod_{r=0}^{l-1} \frac{s-r}{n-r} \cdot \prod_{r=0}^{l-1} \frac{(s-l)-r}{(n-l)-r} \ .$$

Hence, Boole-Benferoni inequality implies that

$$\begin{aligned}
\mathbf{Pr}\left[\exists j \in [k] \ : \ (W_j \subseteq S)\right] \ &\geq \ k \cdot \prod_{r=0}^{l-1} \frac{s-r}{n-r} \ - \binom{k}{2} \cdot \prod_{r=0}^{l-1} \frac{s-r}{n-r} \cdot \prod_{r=0}^{l-1} \frac{(s-l)-r}{(n-l)-r} \\
&= \ k \cdot \prod_{r=0}^{l-1} \frac{s-r}{n-r} \cdot \left(1 - \frac{k-1}{2} \cdot \prod_{r=0}^{l-1} \frac{(s-l)-r}{(n-l)-r}\right) \\
&\geq \ k \cdot \prod_{r=0}^{l-1} \frac{s-l}{n-l} \cdot \left(1 - k \cdot \prod_{r=0}^{l-1} \frac{s-l}{n-l}\right) \\
&= \ k \cdot \left(\frac{s-l}{n-l}\right)^{l} \cdot \left(1 - k \cdot \left(\frac{s-l}{n-l}\right)^{l}\right) \ .
\end{aligned}$$

Now, since we assumed that $s = l + \frac{n-l}{(2k)^{1/l}}$, our calculations above yield $\mathbf{Pr}[\exists j \in [k] : (W_j \subseteq S)] \geq \frac{1}{4}$, and thus the lemma follows. □

The next lemma is useful for certain lower bound constructions. The lemma gives an upper bound on the probability that a sample set of size s chosen uniformly at random contains one of k sets W_i completely.

Lemma 2.4.2 *Let Ω be an arbitrary set of n elements. Let k, l, and s be arbitrary integers (possibly dependent on n). Let $W_1, \ldots, W_k$ be any disjoint subsets of Ω of size l each. Let S be a subset of Ω of size s which is chosen independently and uniformly at random. For every real p, $0 < p < 1$, (possibly depending on other parameters), if $s \leq (n - (l - 1)) \cdot (p/k)^{1/l}$, then*

$$\mathbf{Pr}\left[\exists j \in [k] \ : \ (W_j \subseteq S)\right] \leq p \ .$$

Proof : We use the union bound to obtain that

$$\mathbf{Pr}\left[\exists j \in [k] \ : \ (W_j \subseteq S)\right] \ \leq \ \sum_{j=1}^{k} \mathbf{Pr}[W_j \subseteq S] \ = \ \sum_{j=1}^{k} \prod_{r=0}^{l-1} \frac{s-r}{n-r} \ \leq \ k \cdot \left(\frac{s}{n-(l-1)}\right)^{l} \ .$$

Therefore, if $s \leq (n - (l-1)) \cdot (p/k)^{1/l}$, then the above inequality is upper bounded by p. □

3 Testing Algorithms for Geometric Properties

In this chapter we design and analyze testing algorithms for fundamental problems from the area of Computational Geometry. The problems we consider deal with global properties of geometric objects such as point sets or geometric graphs. We can formulate property testing in the standard model for geometric objects as follows:

Given a geometric object O (for example, a set of points or a geometric graph) and the ability to perform local queries about the object (e.g. queries of the form: 'What is the position of the i-th point of the collection ?', 'What is the i-th edge incident to vertex p ?', or 'What is the position of vertex p ?') decide whether O has a certain (predetermined) property (e.g., whether the set of points is in convex position or whether the geometric graph is a Euclidean minimum spanning tree) or is far away from every object having this property.

We assume that our object representation is in the standard testing model. Point sets are represented in the same way as it has been described in the previous chapter. Other objects we consider are *sets of geometric objects* and *geometric graphs*. The object representation is again in the spirit of the standard testing model. We allow only the most basic queries about the object. It is only required that it is possible to determine the whole structure of the object using these queries. By allowing only basic queries we ensure that our testing algorithm is almost independent of the application (almost every structure supports these basic queries). Later, in Chapter 5 we consider a model where we allow more powerful queries.

The first problem we consider is disjointness of geometric objects. Then we design a property tester for the convex position property of point sets. And finally, we give a property tester for Euclidean minimum spanning trees.

3.1 Intersection of Geometric Objects

The first problem we consider is to test whether a set $\mathcal{O} = \{O_1, \ldots, O_n\}$ of n geometric objects is pairwise disjoint. A geometric object is an (implicitly represented) subset of $\mathbb{R}^d$. Two geometric objects O_1 and O_2 are said to be *disjoint*, if $O_1 \cap O_2 = \emptyset$. A set $\mathcal{O} = \{O_1, \ldots, O_n\}$ of n geometric objects is *pairwise disjoint*, if each pair of objects O_i and O_j, $1 \leq i, j \leq n$, is disjoint. An equivilant formulation of this problem is to test

whether the intersection graph of the objects contains no edges.[1]

We represent sets of geometric objects by a function $f : [n] \to \mathcal{R}$ where $\mathcal{R}$ contains implicit representations of all geometric objects of a certain class of objects, e.g. all line segments, points, or rectangle. For example, when we consider sets of line segments in the $\mathbb{R}^d$ then $\mathcal{R} = \mathbb{R}^d \times \mathbb{R}^d$. The property tester for disjointness of geometric objects is used later in this thesis when we design a property tester for the Euclidean minimum spanning tree. We use the standard distance measure from Definition 2.1.3 which has the following equivalent formulation:

Definition 3.1.1 *A set $\mathcal{O}$ of n objects in the $\mathbb{R}^d$ is ϵ-far from being* pairwise disjoint*, if one has to remove more than ϵn objects from $\mathcal{O}$ to obtain a disjoint set of objects.*

The Testing Algorithm. Like many other property testers the tester for disjointness of geometric objects is very simple. It picks a set S of $8\sqrt{n/\epsilon}$ objects uniformly at random and computes whether these objects are pairwise disjoint. If they are, then the algorithm concludes that in this case either the objects are disjoint or at least 'most of the objects of the input set are disjoint'. Thus in such a case the algorithm can accept the input. If the sample set contains two or more objects that intersect, then we know for sure that the set of objects is not disjoint and the algorithm can reject the input. We prove that the following algorithm is a property tester for disjointness of geometric objects.

DISJOINTNESSTESTER(set of objects $\mathcal{O}$)
Choose a set $S \subseteq \mathcal{O}$ of size $8\sqrt{n/\epsilon}$ uniformly at random
if S is disjoint **then** *accept*
else *reject*

Lemma 3.1.2 *Algorithm* DISJOINTNESSTESTER *is a property tester for disjointness of geometric objects. Its query complexity is $\mathcal{O}(\sqrt{n/\epsilon})$ and its running time is $T(8\sqrt{n/\epsilon}) + \mathcal{O}(1)$, where $T(m)$ is the time to decide whether a set of m input objects is disjoint.*

Proof : We have to prove that (1) algorithm DISJOINTNESSTESTER accepts every set of disjoint geometric objects, and (2) that it rejects every set of geometric objects that is ϵ-far from disjoint with probability at least $2/3$.

Part (1) is immediate: If $\mathcal{O}$ is pairwise disjoint, the every subset of $\mathcal{O}$ is also pairwise disjoint and so algorithm DISJOINTNESSTESTER accepts $\mathcal{O}$.

Thus let us suppose that $\mathcal{O}$ is ϵ-far from disjoint and prove part (2). It is easy to see that we can apply $k = \epsilon n/2$ times the following procedure to $\mathcal{O}$: pick a pair of intersecting objects W_i, $i \in [k]$, and remove it from $\mathcal{O}$. In order to prove that DISJOINTNESSTESTER is a property tester it is sufficient to show that with probability at least $2/3$ at least one of these pairs is in the sample set S. We apply Lemma 2.4.1 and standard amplification

[1] We do not use the formulation as a graph problem here because our distance measure would be based on vertex deletion rather than edge deletion as it is usually in the property testing literature (see Section 2.1).

arguments. We consider the sample set S as four independently selected sets $S_1^*, S_2^*, S_3^*, S_4^*$, each of size $\frac{2n}{(2k)^{1/2}}$ and then apply Lemma 2.4.1 to obtain:

$$\mathbf{Pr}\big[\exists j \in [k] : (W_j \subseteq S)\big] \geq 1 - \prod_{1 \leq i \leq 4} \big(1 - \mathbf{Pr}\big[\exists j \in [k] : (W_j \subseteq S_i^*)\big]\big) \geq 1 - (3/4)^4 \geq 2/3$$

□

A Lower Bound on the Query Complexity. We now want to show that the algorithm DISJOINTNESSTESTER in the general setting of arbitrary geometric objects is optimal with respect to its asymptotic query complexity. For this purpose we consider the problem if a set of unit disks is disjoint. We represent a set of n unit disks by their center points, that is, by a function $f : [n] \to \mathbb{R}^d$. We observe that the disjointness property for unit disks is equivalent to the following *sparseness* property of point sets. A point set P in the $\mathbb{R}^d$ is called *sparse*, if for each pair of points p, q we have $\mathtt{dist}(p, q) > 1$. In order to prove that every property tester for disjointness of geometric objects has query complexity $\Omega(\sqrt{n/\epsilon})$ we show that there is no property tester for sparseness of point sets with query complexity $o(\sqrt{n/\epsilon})$.

Lemma 3.1.3 *Every property tester for disjointness of sets of geometric objects must have query complexity* $\Omega(\sqrt{n/\epsilon})$.

Proof : As already mentioned above sparseness of point sets is a special case of disjointness of geometric objects. Hence, it suffices to show a lower bound for the sparseness property of point sets. Let us assume that there is a property tester $\mathbb{A}$ for sparseness of point sets with query complexity $q(\epsilon, n) = o(\sqrt{n/\epsilon})$.

Claim 3.1.4 *Let* $\mathbb{A}$ *be a property tester for sparseness of point sets with query complexity* $q(\epsilon, n)$. *Then there is a property tester* $\mathbb{A}''$ *for sparseness of point sets that samples a set* S *of* $q(\epsilon, n)$ *points uniformly at random and accepts if and only if the points in* S *are sparse. Such a property tester is called* canonical.

Proof : By Lemma 2.3.1 we know that there is a property tester $\mathbb{A}'$ that samples a set of $q(\epsilon, n)$ points uniformly at random and decides based on S and its internal coin flips. Let $\mathbb{A}''$ denote an algorithm that samples a set S of $q(\epsilon, n)$ points uniformly at random and that accepts if and only if S is sparse. We want to prove that $\mathbb{A}''$ is a property tester. Clearly, algorithm $\mathbb{A}''$ accepts every sparse point set. Thus we have to prove that it rejects every point set that is ϵ-far from being sparse (a point set is ϵ-far from sparse if the corresponding set of unit disks is ϵ-far from disjoint). We show this indirectly by proving that if algorithm $\mathbb{A}'$ rejects a sample set S then so does algorithm $\mathbb{A}''$. By definition a property tester has one-sided error and so $\mathbb{A}'$ must accept if the sample is sparse. But if the sample S is not sparse then $\mathbb{A}''$ rejects and thus it follows that if $\mathbb{A}'$ rejects then so does $\mathbb{A}''$. Since $\mathbb{A}'$ is a property tester it rejects every point set P that is ϵ-far from sparse with probability at least $2/3$. Hence algorithm $\mathbb{A}''$ also rejects such a set P with probability at least $2/3$. Since $\mathbb{A}''$

accepts every sparse point set we have shown that $\mathbb{A}''$ is a property tester for sparseness. This proves Claim 3.1.4. □

We construct a set P of n points in $\mathbb{R}$ that is ϵ-far from sparse but is not rejected by a canonical property tester with query complexity $q(\epsilon, n)$ if n is sufficiently large. P is the union of $\lfloor \epsilon n \rfloor + 1$ pairs W_i of points and a set U of $n - 2(\lfloor \epsilon n \rfloor + 1)$ points such that the following condition is satisfied for all points $p, q \in P$ with $p \neq q$: If there is a W_i with $W_i = \{p, q\}$ then $\mathtt{dist}(p, q) \leq 1$. Otherwise, we have $\mathtt{dist}(p, q) > 1$. We observe that algorithm $\mathbb{A}''$ rejects P if and only if the sample contains one of the sets W_i. We apply Lemma 2.4.2 with $p = 1/2$, $k = \epsilon n + 1$,

$$s = \frac{n-1}{\sqrt{\frac{1}{2}(\epsilon n + 1)}} = \Omega(\sqrt{n/\epsilon})$$

and $l = 2$; thus:

$$\mathbf{Pr}\big[\exists j \in [k] : (W_j \subseteq S)\big] \leq 1/2$$

Since $q(\epsilon, n) = o(\sqrt{n/\epsilon})$ for sufficiently large n we have

$$q(\epsilon, n) \leq \frac{n-1}{\sqrt{\frac{1}{2}(\epsilon n + 1)}} \ .$$

This proves that there is a point set that is ϵ-far from sparse and that is rejected by algorithm $\mathcal{A}''$ with probability at most $1/2$. Since $1/2 < 2/3$ we conclude that $\mathbb{A}''$ is not a property tester. Contradiction. □

We also note that the proof of the lower bound for disjointness of unit disks can be modified to prove lower bounds for other classes of objects including line segments, rectangles, etc. This means that we cannot asymptotically improve the query complexity of a property tester when we use the geometry of the special class of objects.

Summary. We summarize the results from this section in the following theorem:

Theorem 1 *There is a property tester for pairwise disjointness of a set of geometric objects $\mathcal{O}$ with query complexity $\mathcal{O}(\sqrt{n/\epsilon})$ and running time $T(8\sqrt{n/\epsilon}) + \mathcal{O}(1)$ where $T(m)$ denotes the time needed to compute if a set of m objects is disjoint. Every property tester for disjointness of geometric objects has query complexity $\Omega(\sqrt{n/\epsilon})$.*

3.2 Convex Position

In this section we consider a classical property of point sets: Being in *convex position*. Let P denote a set of n points in general position in the $\mathbb{R}^d$. We say that a point set P is in *convex position,* if all points in P are vertices of the convex hull of P:

Definition 3.2.1 *Let P be a point set in the $\mathbb{R}^d$ and let $\mathcal{CH}(P)$ denote the convex hull of P. A point $p \in P$ is called an* extreme point *of P, if p is a vertex of $\mathcal{CH}(P)$. A point set P is in* convex position *if each point in P is an extreme point.*

Using our standard representation of point sets we assume that the input point set P is represented by a function $f : [n] \to \mathbb{R}^d$. We also use the standard distance measure from Definition 2.1.3. In terms of the problem under consideration this can be stated as follows:

Definition 3.2.2 *A set P of n points is ϵ-far from convex position, if no set Q of size (at most) ϵn exists such that $P \setminus Q$ is in convex position.*

In the remainder of this section we prove that the following algorithm is a property tester for convex position:

CONVEXTESTER (P, ϵ)
let $s = 16\left(4\sqrt[d+1]{n^d/\epsilon} + 2d + 2\right)$
Choose a set $S \subseteq P$ of size s uniformly at random
if S is in convex position **then** *accept*
else *reject*

In order to show that CONVEXTESTER is a property tester we have to prove that (1) it accepts every point set in convex position and (2) it rejects every point set that is ϵ-far from convex position with probability at least $2/3$.

We observe that CONVEXTESTER accepts every point set in convex position because every subset of a set in convex position is in convex position, as well.

Thus we only have to prove that a point set that is ϵ-far from convex position is rejected with high probability. We make use of the following theorem by Carathéodory [23]:

Theorem 2 (Carathéodory's Theorem) *If $P \subseteq R^d$ and $p \in \mathcal{CH}(P)$ then $p \in \mathcal{CH}\{p_0, \ldots, p_d\}$ for certain affinely independent vectors $\{p_0, \ldots, p_d\}$ in P.*

Now let us assume that P is ϵ-far from convex position. We want to prove that algorithm CONVEXTESTER rejects P with probability at least $2/3$. It follows from Theorem 2 that every point set that is not in convex position can be rejected by finding a small set of $d + 2$ points that is not in convex position. We refer to such a set as a *counter example*.

The idea of the analysis is now to consider sets W_i, $1 \leq i \leq \frac{\epsilon n}{2(d+1)}$, of $d + 1$ points that can be easily extended to a set of $d + 2$ points that is not in convex position. Easily extendible means that there is a large set U_i of points such that for each $p \in U_i$ the set $W_i \cup \{p\}$ is a counter example. Intuitively, the existence of such a large set means that we get one of the $d+2$ points of the counter example for free because our sample set contains at least one point of U_i with very high probability.

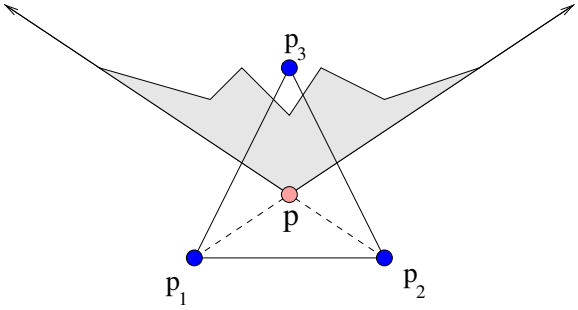

Figure 3.1: Illustration for the proof of Lemma 3.2.3. One of the cones C_i (this one is defined by p_1 and p_2).

Finding Counter Examples. We now want to formalize this idea and prove two technical lemmas. First we prove that for a given point p in the interior of the convex hull of a point set P there exists a set $W \subseteq P$ of d points such that there are at least $\frac{n}{d+1}$ points in P such that any of them can be added to $W \cup \{p\}$ to obtain a point set that is not in convex position.

Lemma 3.2.3 *Let P be a set of n points in $\mathbb{R}^d$ and let $p \in \mathbb{R}^d$ be a point with $p \in \mathcal{CH}_{\text{int}}(P)$ and let $P \cup \{p\}$ be in general position. Then there exist sets $W \subseteq P$ and $U \subseteq P \setminus W$ with $|W| = d$ and $|U| \geq \frac{n}{d+1}$ such that $p \in \mathcal{CH}_{\text{int}}(W \cup \{q\})$ for each $q \in U$.*

Proof : By Theorem 2 and the general position assumption it follows that there is a set $W^* \subseteq P$ of size $d+1$ such that $p \in \mathcal{CH}_{\text{int}}(W^*)$. Let us denote by W_i^*, $1 \leq i \leq d+1$, the subsets of W^* of size d. We show that one of the subsets W_i^* of W^* satisfies Lemma 3.2.3. W.l.o.g., let us assume that $p = (0, \ldots, 0)$. We now want to partition the $\mathbb{R}^d$ into $d+1$ cones. Let W_i^-, $1 \leq i \leq d+1$, denote the set of points $\{(-x_1, \ldots, -x_d) : (x_1, \ldots, x_d) \in W_i^*\}$. The conic combination of the point vectors in the set W_i^- defines a cone C_i ,$1 \leq i \leq d+1$. The union of these cones C_i covers the $\mathbb{R}^d$. Thus there is a cone C_j, $1 \leq j \leq d+1$ that contains at least $\frac{n}{d+1}$ points of P (and also of $P \setminus W_j^*$ since $W_j^* \cap C_j = \emptyset$). We further observe that for each $q \in P \cap C_j$ it holds that $p \in \mathcal{CH}_{\text{int}}(W_j^* \cup \{q\})$. We conclude that the sets $W = W_j^*$ and $U = P \cap C_j$ satisfy the lemma. □

The second technical lemma shows that if a point set is ϵ-far from convex position then there are many sets W_i and U_i such that $W_i \cup \{q\}$ is not in convex position for every $q \in U_i$.

Lemma 3.2.4 *Let P be a set of n points in the $\mathbb{R}^d$ that is ϵ-far from convex position and let $k = \frac{\epsilon n}{d+1}$. Then there exist sets $W_i \subseteq P$, $i \in [k]$, and sets $U_i \subseteq P$, $i \in [k]$, such that the following properties are fulfilled:*

(1) $|W_i| = d+1$, $i \in [k]$,

(2) $W_i \cap W_j = \emptyset$ *for all* $i, j \in [k]$ *with* $i \neq j$,

(3) $W_i \cup \{q\}$ *is not in convex position for every* $q \in U_i$, *and*

(4) $|U_i| \geq \frac{1-\epsilon}{d+1}n$.

Proof : We construct point sets $P_1 = P, P_2, \ldots, P_k$ with $W_i \subseteq P_i$ and $P_{i+1} = P_i \setminus W_i$. Clearly, our construction satisfies that the sets W_i are disjoint. We show how to construct W_i from the set P_i: First of all, observe that for every $i \in [k]$ it holds that $|P_i| = n - (d+1)(i-1)$ and thus $|P_i| \geq n - (d+1)(\frac{\epsilon n}{d+1} - 1)$. This implies that $|P \setminus P_i| < \epsilon n$ and so the set P_i cannot be in convex position (by the definition of ϵ-far we can delete every set of size at most ϵn and do not obtain a point set in convex position). Since P_i is not in convex position there is a point $p \in \mathcal{CH}_{int}(P_i)$. We apply Lemma 3.2.3 with p and $P_i \setminus \{p\}$ and obtain the sets W and U. We choose $W_i = W \cup \{p\}$ and $U_i = U$. By Lemma 3.2.3 we get $|W_i| = d+1$ and $W_i \cup \{q\}$ is not in convex position for each $q \in U_i$. We already observed that $|P_i| \geq n - (d+1)(\frac{\epsilon n}{d+1} - 1)$. Thus we know that $|P_i \setminus \{p\}| \geq n - \epsilon n$ which means that $|U_i| \geq \frac{n-\epsilon n}{d+1} = \frac{1-\epsilon}{d+1}n$. All properties in Lemma 3.2.4 are satisfied and the lemma is proven. □

Lemma 3.2.5 *Algorithm* ConvexTester *is a property tester for the convex position property. Its query complexity is* $\mathcal{O}(\sqrt[d+1]{n^d/\epsilon})$.

Proof : We have to prove that ConvexTester accepts every point set in convex position and rejects every point set that is ϵ-far from convex position with probability $2/3$.

We already observed that algorithm ConvexTester accepts every point set in convex position.

Thus we have to show that every point set that is ϵ-far from convex position is rejected with probability at least $2/3$. Let P be a point set that is ϵ-far from convex position and let $k = \frac{\epsilon n}{d+1}$. Then there exist sets W_i and U_i, $i \in [k]$, with the properties stated in Lemma 3.2.4. Now let $S \subseteq P$ be a point set of size $4\sqrt[d+1]{n^d/\epsilon} + 2(d+1)$ chosen uniformly at random from P. W.l.o.g., let us assume that $\epsilon < 1/2$. We view S as the union of two sets W and U of size $4\sqrt[d+1]{n^d/\epsilon}$ and $2(d+1)$, respectively. Clearly, we can assume that each of them is chosen uniformly (and independently) at random from P. We first observe that for a fixed U_i of size $\frac{1-\epsilon}{d+1}n$:

$$\mathbf{Pr}\big[U_i \cap U \neq \emptyset\big] \geq 1 - \big(1 - \frac{|U_i|}{n}\big)^{|U|} \geq 1 - 1/e \geq 1/2$$

We further know that:

$$\begin{aligned}\mathbf{Pr}\big[P \text{ is rejected}\big] &\geq \mathbf{Pr}\big[\exists i \in [k] : W_i \subseteq S \text{ and } U_i \cap S \neq \emptyset\big] \\ &\geq \mathbf{Pr}\big[\exists i \in [k] : W_i \subseteq W \text{ and } U_i \cap U \neq \emptyset\big] \\ &\geq \mathbf{Pr}\big[\exists i \in [k] : W_i \subseteq W\big] \cdot 1/2 \ .\end{aligned}$$

To derive a bound on $\mathbf{Pr}\big[\exists i \in [k] : W_i \subseteq W\big]$ we apply Lemma 2.4.1 with $k = \frac{\epsilon n}{d+1}$, $l = d+1$, and

$$s = 4\sqrt[d+1]{n^d/\epsilon} \geq \sqrt[d+1]{(2n)^d \cdot (d+1)/\epsilon} \geq \frac{2n}{(2k)^{1/l}}$$

and obtain:

$$\mathbf{Pr}\big[\exists i \in [k] : W_i \subseteq W\big] \geq 1/4 \,.$$

We conclude that

$$\mathbf{Pr}\big[P \text{ is rejected}\big] \geq 1/8 \,.$$

If algorithm CONVEXTESTER chooses a set S of size $16 \cdot \left(4\sqrt[d+1]{n^d/\epsilon} + 2(d+1)\right)$ then

$$\mathbf{Pr}\big[P \text{ is rejected}\big] \geq 1 - (1 - 1/8)^{16} \geq 2/3$$

which proves the query complexity stated in Lemma 3.2.5. □

A Lower Bound on the Query Complexity. We now want to prove that our property tester is optimal with respect to the asymptotic query complexity. The proof follows the same line of thought as the proof of Lemma 3.1.3. We first show that the existence of a property tester with query complexity $q(\epsilon, n)$ implies that there is a canonical property tester with the same query complexity, i.e. a property tester that samples a set S of $q(\epsilon, n)$ points uniformly at random and accepts, if and only if the sample set S is in convex position. Then we show that for sufficiently large n there is a point set P that is ϵ-far from convex position such that

$$\mathbf{Pr}[S \text{ is in convex position }] > 1/3$$

holds, if $S \subseteq P$ is a set of $q(\epsilon, n) = o(\sqrt[d+1]{n^d/\epsilon})$ points chosen uniformly at random from P. This implies that there is no property tester with query complexity $q(\epsilon, n)$.

Lemma 3.2.6 *Every property tester (with one sided error) for convex position has query complexity* $\Omega(\sqrt[d+1]{n^d/\epsilon})$.

Proof : The proof is by contradiction. Assume there is a property tester $\mathbb{A}$ for convex position with query complexity $q(\epsilon, n) = o(\sqrt[d+1]{n^d/\epsilon})$. We first want to prove that there is a canonical tester with the same query complexity:

Claim 3.2.7 *Let $\mathbb{A}$ be a property tester for convex position of point sets with query complexity $q(\epsilon, n)$. Then there is a property tester $\mathbb{A}''$ for convex position of point sets that samples a set S of $q(\epsilon, n)$ points uniformly at random and accepts if and only if the points in S are in convex position.*

Proof : The proof is similar to the proof of Claim 3.1.4. By Lemma 2.3.1 we know that there is a property tester $\mathbb{A}'$ that samples a set of $q(\epsilon, n)$ points uniformly at random and decides based on S and its internal coin flips. Let $\mathbb{A}''$ be an algorithm that samples a set S of $q(\epsilon, n)$ points uniformly at random and that accepts if and only if S is in convex position. We want to prove that $\mathbb{A}''$ is a property tester. Clearly, algorithm $\mathbb{A}''$ accepts every point set in convex position. Thus we have to prove that it rejects every point set that is ϵ-far from convex position. We show this indirectly by proving that if algorithm $\mathbb{A}'$ rejects a sample set S then so does algorithm $\mathbb{A}''$. By definition a property tester has one-sided error and

so $\mathbb{A}'$ must accept if the sample is in convex position. But if the sample S is not in convex position then $\mathbb{A}''$ rejects. It follows that $\mathbb{A}''$ always rejects when $\mathbb{A}'$ rejects. Since $\mathbb{A}'$ is a property tester it rejects every point set P that is ϵ-far from disjoint with probability at least $2/3$. Hence algorithm $\mathbb{A}''$ also rejects such a set P with probability at least $2/3$. Since $\mathbb{A}''$ accepts every point set in convex position we have shown that $\mathbb{A}''$ is a property tester for convex position. This proves Claim 3.2.7. □

Let us denote by $\mathbb{A}''$ the property tester as obtained in Claim 3.2.7. In the remainder of the proof we construct a point set P of n points that is ϵ-far from convex position and show that this point set is accepted by $\mathbb{A}''$ (observe that $\mathbb{A}''$ is uniquely defined by $q(\epsilon, n)$) with probability at least $1/2$. Since P is ϵ-far from convex position this is a contradiction to the fact that $\mathbb{A}''$ is a property tester (which would require that P is rejected with probability at least $2/3 > 1 - 1/2 = 1/2$).

We start with an overview of our construction and then present it in details. The idea is to construct a point set P that consists of $\epsilon n + 1$ sets W_i of size $d+1$ (and some additional points). Each set W_i consists of a set F_i of d extreme points that form a facet of the convex hull and one point q_i that is located infinitesimally close to the facet but in the interior of the convex hull. The point set has the property that every subset $S \subseteq P$ of size more than $d+1$ is in convex position, if and only if there is a set $W_i \subseteq S$.

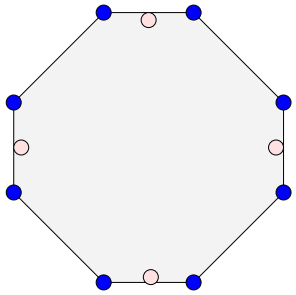

Figure 3.2: An example for the lower bound construction in 2D.

Now let S be a point set of size s chosen uniformly at random from P. We apply Lemma 2.4.2 with $p = 1/2$ and obtain that for

$$s \leq (n-d) \cdot \frac{1}{2(\epsilon n+1)}^{1/(d+1)} = \Theta(\sqrt[d+1]{n^d/\epsilon})$$

our sample set S does not contain one of the sets W_i with probability at least $1/2$. Hence S is in convex position with probability at least $1/2$ and thus algorithm $\mathbb{A}''$ is not a property tester, if $q(\epsilon, n) \leq (n-d) \cdot \frac{1}{2(\epsilon n+1}^{1/(d+1)}$.

The Detailed Construction. For a set X and a point p in $\mathbb{R}^d$ let $dist(X, p) = \min_{x \in X} ||x - p||$. Let P_1 be a set of $n - d \cdot (\epsilon n + 1)$ points in general and convex position and let $\epsilon < 1/d - 1/n$. Further let $P_2 = \{p_1, \ldots, p_k\} \subseteq P_1$ be a subset of size $k = \epsilon n + 1$.

For each point $p \in P_1$ let h_p denote a hyperplane (a tangent to $\mathcal{CH}(P_1)$) such that $h_p \cap \mathcal{CH}(P_1) = p$. Further let $\mathcal{H}_p^+, \mathcal{H}_p^-$ denote the closed halfspaces induced by h_p such that $\mathcal{H}_p^+$ denotes the halfspace containing P_1 (Figure 3.2). Let

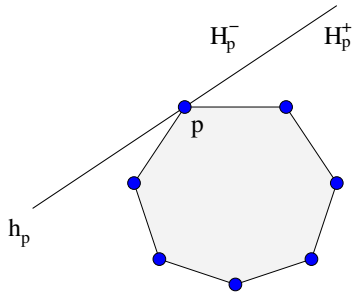

Figure 3.3: Step 1: We start with a point set in convex position.

$$\beta = \min_{p,q \in P_1, p \neq q} dist(h_p, q) \ .$$

Observation 3.2.8 *Every point set $\widetilde{P}_1$ obtained from P_1 by perturbating each of the points in P_1 by a distance of less than $\beta/2$ is in convex position.*

Instead of perturbating the point set we replace each point $p_i \in P_2$ by a set F_i, $i \in [k]$, of d points (Figure 3.4). We do this replacement in such a way that we obtain a point set P_3 in general and convex position. For a point $p \in \mathbb{R}^d$ let $\mathbb{B}(p, \beta/3)$ denote the d-dimensional ball with center p and radius $\beta/3$. We choose each F_i such that

- $q \in h_p \cap \mathbb{B}(p, \beta/3)$ for each $q \in F_i$.
- $P_3 = P_1 \setminus P_2 \cup F$ is in general position, where $F = \bigcup_{i \in [k]} F_i$

Clearly, we can choose the F_i such that P_3 is in general position because we can move the points within $\mathbb{B}(p_i, \beta/3) \cap h_{p_i}$ to destroy every degenerate cases. We observe that P_3 is in convex position since h_{p_i} is a witness for the fact that the points in F_i are extreme points in P_3. We obtain the set P from P_3 by adding a set $Q = \{q_1, \ldots, q_k\}$ of k points to P_3. Let $W_i = F_i \cup \{q_i\}$ for each $q_i \in Q$. We want to choose Q in such a way that:

- $P = P_3 \cup Q$ is ϵ-far from convex position,
- If a subset $S \subseteq P$ contains no set W_i, $i \in [k]$, completely, then S is in convex position.

We now give a construction that satisfies these properties. For each $i \in [k]$ let q_i' denote a point in the interior of the ($(d-1)$-dimensional) convex hull of F_i and let v_i denote a vector pointing from q_i' in the interior of $\mathcal{CH}(P_3)$ (Figure 3.5). Let $q_i(\beta)$ denote the point $q_i' + \beta \cdot v_i$. For $i \in [k]$ and $p \in F_i$ let $h_{F_i,p}(\beta)$ denote the hyperplane induced by $F_i \setminus \{p\} \cup \{q_i(\beta)\}$. Further let $\mathcal{H}^+_{F_i,p}(\beta)$ and $\mathcal{H}^-_{F_i,p}(\beta)$ denote the halfspaces induced by

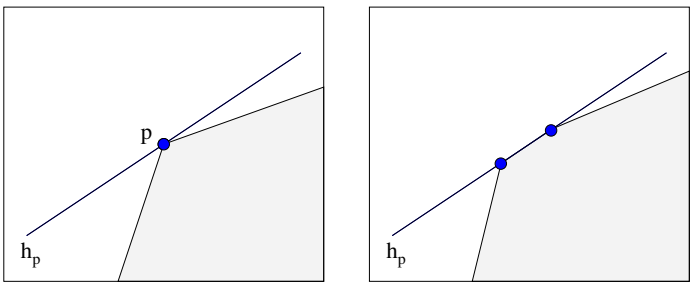

Figure 3.4: Step 2: We replace some points by a set F_i of $d = 2$ points.

$h_{F_i,p}(\beta)$ such that $\mathcal{H}^+_{F_i,p}(\beta)$ contains P for $\beta = 0$. Since P_3 is in general position there is a value β^* such that for every $i \in [k]$ and every $p \in F_i$ the halfspace $\mathcal{H}^+_{F_i,p}(\beta^*)$ contains $P_3 \setminus \{p\}$ and such that $P_3 \cup Q$ is in general position (where Q is defined as follows). We choose $q_i = q_i(\beta^*)$ and $Q = \bigcup_{i\in[k]} q_i$. We now have to prove that Q has the required properties:

Claim 3.2.9 *The set $P = P_3 \cup Q$ is ϵ-far from convex position. If a subset $S \subseteq P$ does not contain a set W_i completely, then S is in convex position.*

Proof : Assume P is ϵ-close to convex position. Then there is a set R of at most ϵn points such that $P \setminus R$ is in convex position. Since $|R| \leq \epsilon n$ there is a set W_i completely contained in $P \setminus R$. Further we know that by the size of P and R there must be at least one further point q in $P \setminus R$. By the choice of Q we know that q is in $\mathcal{H}^+_{F_i,p}(\beta^*)$ for each $p \in F_i$. Hence q_i is in the interior of $\mathcal{CH}(F_i \cup \{q\})$ which contradict the fact that $P \setminus R$ is in convex position.

It remains to prove the second statement. Let $S \subseteq P$ a subset that contains no set W_i completely. W.l.o.g., we assume that it contains exactly d points of each set W_i. If these points are the points in F_i then by our construction the hyperplane h_{p_i} is a witness that the points in $W_i = F_i$ are extreme. Otherwise, one of the points in F_i is not in S. Let is call the missing point p. Then $h_{F_i,p}(\beta^*)$ is a witness that the points are extreme points. Thus for every point $p \in S$ we have a witness that p is extreme. Hence S is in convex position. □

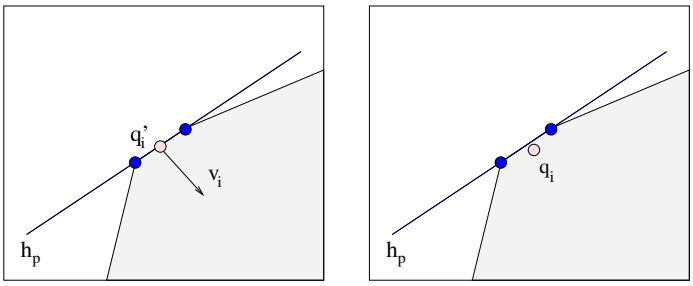

Figure 3.5: Step 3: We place another point in the interior of $\mathcal{CH}(F_i)$ and move it slightly along v_i into the interior of the convex hull.

Now we can apply Lemma 2.4.2 with $p = 1/2$ and obtain that for

$$s \leq (n-d) \cdot \frac{1}{2(\epsilon n+1)}^{1/(d+1)} = \Omega(\sqrt[d+1]{n/\epsilon})$$

a sample set S of size s chosen uniformly at random from P does not contain one of the W_i with probability at least $1/2$. Since $q(\epsilon, n) = o(\sqrt[d+1]{n^d/\epsilon}$ we have for sufficiently large n

$$q(\epsilon, n) \leq (n-d) \cdot \frac{1}{2(\epsilon n+1)}^{1/(d+1)}.$$

It follows that there exists a point set P that is ϵ-far from convex position and that is accepted by algorithm $\mathbb{A}''$ with probability at least $1/2$. Since the existence of a property tester for convex position with query complexity $q(\epsilon, n)$ implies that algorithm $\mathbb{A}''$ is a property tester, it follows that there is no property tester with query complexity $o(\sqrt[d+1]{n^d/\epsilon})$. □

Summary. We summarize the results of this section in the following theorem:

Theorem 3 *Let P be a point set of n points in the $\mathbb{R}^d$. Then there is a property tester for the convex position property with query complexity $\mathcal{O}(\sqrt[d+1]{n^d/\epsilon})$ and running time $\mathcal{O}(T(\sqrt[d+1]{n^d/\epsilon}))$ where $T(n) = \mathcal{O}(n \cdot \log^{\mathcal{O}(1)} h + (nh)^{\frac{\lfloor d/2 \rfloor}{\lfloor d/2 \rfloor + 1}} \cdot \log^{\mathcal{O}(1)} n)$ is the running time of the fastest known algorithm [24] to decide if a point set is in convex position (h denotes the number of extreme points of the set). Every property tester for convex position has a query complexity of $\Omega(\sqrt[d+1]{n^d/\epsilon})$.*

Proof : Follows from Lemma 3.2.5 and Lemma 3.2.6. □

3.3 Euclidean Minimum Spanning Tree

We are given a point set P in the $\mathbb{R}^2$ *and* a geometric graph $G = (P, E)$ with vertex set P and edge set E and we want to design a property tester for the property that G is a *Euclidean Minimum Spanning Tree (EMST)* of the point set P. A graph $G = (P, E)$ is called a Euclidean minimum spanning tree of point set P, if G is a minimum spanning tree of the complete Euclidean graph of P. The complete Euclidean graph is a complete weighted graph where each edge $e = (p, q) \in P \times P$ has weight equal to the Euclidean distance between p and q.

In this section we make slightly different assumptions on the input representation. We assume that our property tester has oracle access to point set P and graph G each represented by a separate function. The point set is represented in a similar way as in the previous subsection. The graph is represented in the unbounded length adjacency list model [67]. Details on the input representation follow in Subsection 3.3.1.

For simplicity we assume throughout this section that P is in general position, i.e., all edge weights are distinct and hence the EMST is unique and its maximum degree is five.

3.3.1 Basic Definitions and Input Representation

Let P denote a set of n points in general position in $\mathbb{R}^2$ (we consider only the problem in two dimensional space) and let $\mathbb{T} = (P, \mathbb{E})$ denote the Euclidean minimum spanning tree of P. We start with some basic definitions needed in this section before we discuss the input representation:

Definition 3.3.1 *A* geometric graph *for P is a weighted graph $G = (P, E)$ with vertex set P and edge set $E \subseteq P \times P$ (the edges can be interpreted as straight-line segments connecting the endpoints). The weight of an edge (p, q) is implicitly given by the Euclidean distance between p and q in $\mathbb{R}^2$.*

Definition 3.3.2 *A geometric graph for P that is the minimum spanning tree of the complete geometric graph for P is called the* Euclidean minimum spanning tree (EMST) *of P.*

In this section we do not use the standard distance measure (though the differences are insignificant), because the input object consists of two parts, the point set and the graph. Further we use the unbounded length adjacency list model which uses a non-functional representation of the graph. Our results also hold in the (weaker) bounded degree adjacency list model.

Typically, a distance measure for graph properties depends on the number of entries in the graph representation that must be changed to obtain a graph that has the tested property. In our case, the size of the graph representation depends on the number of edges in the graph. Since every EMST has $n - 1$ edges we let our distance measure depend on the number of vertices instead of the number of edges:

Definition 3.3.3 *Let $G = (P, E)$ be a geometric graph for P and let $\mathbb{T} = (P, \mathbb{E})$ denote the Euclidean minimum spanning tree of P. We say G is* ϵ-far from being the Euclidean minimum spanning tree of P *(or, in short,* ϵ-far from EMST*), if one has to modify (insert or delete) more than ϵn edges in G to obtain $\mathbb{T}$, that is :*

$$|E \setminus \mathbb{E}| + |\mathbb{E} \setminus E| > \epsilon n \ .$$

We want to design a property tester for the property of being a Euclidean minimum spanning tree of a point set in the plane. Such a property tester takes as input a point set P in general position in $\mathbb{R}^2$ and a geometric graph G for P, and accepts the input if G is the EMST for P and rejects the input with probability at least $\frac{2}{3}$ if G is ϵ-far from being the Euclidean minimum spanning tree of P.

Input Representation. Our property tester has to verify whether a geometric graph is the EMST of a point set P. Hence, it must have access to the graph and the point set. Both the point set and the graph are given as an oracle. Similar to the previous section the point set is represented by a function $f : [n] \to \mathbb{R}^2$. Thus the algorithm may query the oracle for the value of $f(i)$ for some $i \in [n]$ and gets in return the position of the i-th point of P.

The geometric graph is given in the *unbounded length adjacency list representation* introduced in [67]. Let us briefly recall the description of the model as given in Chapter 2: The unbounded length adjacency list model is a general model for sparse graphs. The graph structure is represented by adjacency lists of varying length. Our property tester may query for the degree $\mathtt{deg}(p)$ of a vertex p and for each $i \leq \mathtt{deg}(p)$ it may query for the i-th neighbor of p. We represent the vertex set of our graph by the set of numbers $[n]$. Thus we can easily obtain the position of a vertex p from the point set representation by querying for the value of $f(p)$.

Basic Properties of EMSTs. The following claim states some basic (and well-known) properties about Euclidean minimum spanning trees:

Claim 3.3.4 *Every Euclidean minimum spanning tree of a point set in general position in $\mathbb{R}^2$ has maximum degree less than or equal to five, is connected, and its straight-line embedding is crossing-free.*

Proof : The EMST is connected since it is a spanning tree of a complete graph. Its embedding is crossing-free because it is a subgraph of the Delaunay triangulation; see, e.g., [61]. Assume the EMST has a vertex with degree 6 or larger. Then there are two edges with an angle less than 60 degree since the point set is in general position. But this means that at most one of these edges can be in the EMST. Contradiction. □

Now we want to introduce some additional notation that will be useful to simplify the description of the algorithm and its analysis. Let G denote a geometric graph for P. The *EMST-completion* of G is a geometric graph G' over the same point set that contains all edges from G and all edges of the EMST (that are missing in G).

Definition 3.3.5 *For a given point set P, a geometric graph $G = (P, E)$, and the Euclidean minimum spanning tree $\mathbb{T} = (P, \mathbb{E})$ of P, the* EMST-completion *of G is the geometric graph $G_C = (P, E \cup \mathbb{E})$ that contains all edges that are either in G or in $\mathbb{T}$.*

In the next subsection we present a property tester for EMST that works for a special class of input graphs which we call *well-shaped*. A well-shaped graph is a connected graph with maximum degree 5 that has crossing-free EMST-completion. The restriction to well-shaped graphs simplifies the analysis of the algorithm and it allows a clear view on the important features of the property tester.

Definition 3.3.6 *Let G be a geometric graph for point set P. We call G* well-shaped, *if*

- *it has maximum degree of 5,*
- *it is connected,*
- *and the straight-line embedding of its EMST-completion is crossing-free.*

Later in this section we present a general testing algorithm for EMST. This algorithm first tests if the input graph is far from a well-shaped graph. If this is the case then we can reject the graph by Claim 3.3.4. If the input graph passes the test, then we know that with good probability it is either close to a well-shaped graph or it is well-shaped. If the graph is well-shaped we can use the testing algorithm for the special case of well-shaped graphs. At the end of this section we show that for this algorithm it does not matter if the graph is well-shaped or close to well-shaped. Hence, we can also use the algorithm if the graph is close to a well-shaped graph.

3.3.2 Testing EMSTs in Well-Shaped Graphs

We now design a property tester for EMST for the case that our input graph is well-shaped. First, we give an overview of the algorithm:

Let G denote a well-shaped geometric graph with vertex set P. Our property tester uses the following procedure: We first pick a sample set $S \subseteq P$ using some randomized scheme to be described later. Next, we find the subgraph G_S of G that is induced by the vertex set S. Then we compute the EMST-completion of G_S. If the EMST-completion has a cycle then we reject the input, otherwise we accept. Computing the EMST-completion can be done in time $\mathcal{O}(|S| \cdot \log |S|)$. We just have to compute the EMST of S and insert the missing edges in G_S. The query complexity of the tester is $\mathcal{O}(|S|)$.

We first show that we can always reject the input graph, if the EMST-completion of G_S contains a cycle. We use the following lemma which follows easily from standard theory of minimum spanning trees (see, e.g., [77, Chapter 6]). It implies that if we sample a set of vertices $S \subseteq P$ then an edge e cannot be in the EMST of P if it is not in the EMST of S.

Lemma 3.3.7 *Let $S \subseteq P$ be a subset of P and let $p, q \in S$. If the edge $e = (p, q)$ does not belong to the EMST of S, then e does not belong to the EMST of P.*

Proof : Our proof is by contradiction. Let us suppose that e does not belong to the EMST of S and e belongs to the EMST $\mathbb{T}$ of P. The removal of e in $\mathbb{T}$ cuts $\mathbb{T}$ into two trees. These two trees induce a partition of P into two subsets P_1 and P_2. Since e belongs to the EMST of P, e must also be the shortest edge between these two subsets. Let $S_1 = P_1 \cap S$ and $S_2 = P_2 \cap S$. P_1 and P_2 are not empty since one vertex of e is in each of the sets. Then e is also the shortest edge between S_1 and S_2 and therefore it belongs to the EMST of S; contradiction. □

Lemma 3.3.7 is of major importance for the way we proceed. Therefore we state its consequences in Corollary 3.3.8 below. First of all, it shows that our property tester always accepts the EMST of P (it rejects only, if there is a cycle in the EMST-completion and such a cycle provides a counter example for the EMST property). Further it can be used to show that we can reject the input graph, if all vertices of a cycle in the EMST-completion of G are contained in S:

Corollary 3.3.8 *Let G be a geometric graph for P. Let $S \subseteq P$ be a subset of P and let G_S be the subgraph induced by S.*

- *If the EMST-completion* $G' = (P, E')$ *of* G *contains a cycle* $C = (p_0, \ldots, p_k)$ [2] *of length* $k \geq 3$ *and* $p_i \in S$ *for all* $0 \leq i \leq k$*, then there is a cycle in the EMST-completion of* G_S*.*
- *If the EMST-completion of* G_S *contains a cycle* $C = (p_0, \ldots, p_k)$ *of length* $k \geq 3$*, then* G *is not the EMST of* P*.*

Proof : We have to prove that C is a cycle in the EMST-completion of G_S, i.e., that every edge $(p_{i-1}, p_i + 1)$ for all $i \in [k]$ is in the EMST-completion G'_S of G_S. Every such edge is either an edge of the EMST or an edge of the input graph. Obviously, the edges of the input graph are in G'_S and by Lemma 3.3.8 all EMST edges are also in G'_S.

Now we prove the second part: Since the EMST-completion of G_S contains a cycle there is an edge e in G_S that does not belong to the EMST of S. It follows by Lemma 3.3.7 that e is not in the EMST of P. Hence G is not the EMST of P. □

Now we consider the case when the input graph $G = (P, E)$ is ϵ-far from EMST. Our goal is to design a randomized sampling scheme such that the EMST-completion of the subgraph induced by the sample set contains a cycle with high probability. Let $\mathbb{T} = (P, \mathbb{E})$ denote the EMST of P and let $G_C = (P, \mathbb{E} \cup E)$ denote the EMST-completion of G. In the following we refer with *red* edges to the edges in $\mathbb{E} \setminus E$ and with blue edges to the edges in E. It turns out that it is sufficient to focus on "short" cycles that contain at most two red edges:

Definition 3.3.9 *Let* G *be a geometric graph for* P *and let* C *be a cycle of length* k *in the EMST-completion of* G*. We call* C ϵ-short *if (1) its length* k *is smaller than or equal to* $\frac{72}{\epsilon}$ *and (2) it contains at most two red edges.*

In our algorithm we try to find ϵ-short cycles that satisfy some additional "topological" properties. We want to exploit the fact that G is well-shaped, in particular, that the EMST-completion of G has a crossing-free straight-line embedding. Hence we use a topological-like representation of the geometric graph G to exploit the fact that every minimal cycle in a (well-shaped) planar geometric graph corresponds to a face in its straight-line embedding. In order to use this approach in a formal framework we will consider the geometric graph G not only as an undirected graph, but at the same time also using its "directed" representation by "replacing" each undirected edge (p, q) by two directed edges $[p, q\rangle$ and $[q, p\rangle$.

For every vertex p in G we (cyclically) sort incident outgoing edges in clockwise order around the vertex p with respect to the Euclidean positions of the edges' endpoints. (This sorting is done only implicitly, but since we assume that each vertex has a constant degree, each time we consider a vertex we can in constant time sort its incident edges.) The *successor* of a directed edge $[p, q\rangle$ is the edge $[q, r\rangle$ where r is the vertex adjacent to q that precedes p in the adjacency list of q (sorted in clockwise order around q). With this definition cycles of succeeding edges correspond to faces of the straight-line embedding.

[2] Here, $C = (p_0, \ldots, p_k)$ is a cycle (of length k) if $p_i \in P$ for all $i \in [k]$, $p_0 = p_k$, $(p_{i-1}, p_i) \in E'$ for all $i \in [k]$ and $p_i \neq p_j$ for all $i, j \in [k]$, $i \neq j$.

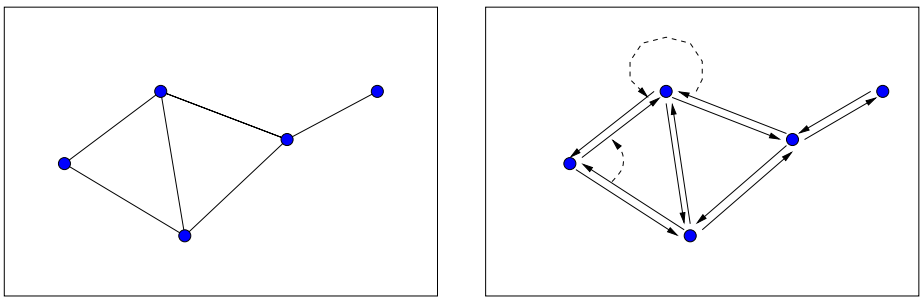

Figure 3.6: A straight-line embedding and its planar map representation.

Such a representation of G is called a *planar map* for the straight-line embedding of G (see figure 3.6). We denote it $\widetilde{G}$.

Definition 3.3.10 *Let G be a well-shaped geometric graph for P and let $\widetilde{G}$ be the corresponding planar map. For each edge $e = [p, q\rangle$ in $\widetilde{G}$ the* successor *of e is the edge $[q, r\rangle$ in $\widetilde{G}$ with r being the vertex adjacent to q that precedes p in the adjacency list of q (sorted in clockwise order around q); if q has degree one, then $r = p$.*

For edge $e = [p, q\rangle$ in G the kth successor, $k \geq 0$, *is defined recursively as follows: the 0th successor of $[p, q\rangle$ is $[p, q\rangle$ itself, and for $k > 0$, the kth successor of $[p, q\rangle$ is the successor of the $(k-1)$st successor of $[p, q\rangle$.*

Similarly, edge e is the predecessor *of edge e' if edge e' is the successor of edge e, and e is the* kth predecessor *of e' if e' is the kth successor of e.*

We have introduced the planar map representation of a graph G because it describes the faces of the corresponding embedding in a simple way (using succeeding edges in $\widetilde{G}$). We observe that the correspondence between the faces in the embedding of G and the cycles of succeeding edges in $\widetilde{G}$ is one to one. We also note that each (directed) edge is contained in exactly one cycle of succeeding edges.

Definition 3.3.11 *Let G be a well-shaped geometric graph for P and let $C = (p_0, \ldots, p_k)$ be a cycle in the planar map of the EMST-completion of G. Then C is called* topological, *if for every two consecutive edges on the cycle $[p_i, p_{i+1}\rangle$ and $[p_{i+1}, p_{i+2}\rangle$, $[p_{i+1}, p_{i+2}\rangle$ is the successor of $[p_i, p_{i+1}\rangle$. We also call the corresponding cycle in G topological.*

The following key lemma shows that every well-shaped geometric graph that is far from EMST must contain many short topological cycles in its EMST-completion.

Lemma 3.3.12 *Let $G = (P, E)$ be a well-shaped geometric graph for P. If G is ϵ-far from EMST, then there are at least $\frac{\epsilon n}{100}$ ϵ-short topological cycles in the EMST-completion of G.*

Proof : Let $\mathbb{T} = (P, \mathbb{E})$ denote the EMST of P and let $E_B = E \setminus \mathbb{E}$ denote the blue edges of the EMST-completion of G that are not in $\mathbb{E}$. Further let $E_R = \mathbb{E} \setminus E$ denote the red edges

of the EMST-completion of G. Since G is ϵ-far from EMST we have $|E_B| + |E_R| > \epsilon n$ by definition of ϵ-far.

Now let H denote the EMST-completion of G and let $\widetilde{H}$ denote the planar map of its straight-line embedding. Let us denote by ρ the number of faces in the straight-line embedding of H. Hence $\widetilde{H}$ has ρ (disjoint) topological cycles since each face is bounded by a unique cycle of succeeding edges (which by definition is called topological). Since H is planar and connected and has more than $n - 1 + \epsilon n/2$ edges we can apply Euler's formula to deduce that $\rho \geq \epsilon n/2$.

Now let $s(f)$ denote the number of (directed) edges in the topological cycle bounding face f. Since $\sum_f s(f) \leq 6n$ by Euler's formula there can be at most $\frac{\rho}{4}$ faces f with $s(f) \geq \frac{48}{\epsilon}$. Thus $\frac{3\rho}{4}$ faces f have $s(f) < \frac{48}{\epsilon}$.

Since $|E_R| \leq \rho$ the number of directed red edges is at most 2ρ. Hence the number of topological cycles with 3 or more red edges can be at most $\frac{2\rho}{3}$. Since we have shown that there are at least $\frac{3\rho}{4}$ topological cycles having less than $\frac{48}{\epsilon}$ edges, at least $\frac{\rho}{12} \geq \frac{\epsilon n}{100}$ of them have at most two red edges. □

Definition 3.3.13 *Let G be a well-shaped geometric graph for P. For every vertex $p \in P$, we define its* topological k-neighborhood *as the set of vertices that are the endpoints of the edges that are either the ith successor, $0 \leq i \leq k$, of any edge incident to p, or are the jth predecessor, $0 \leq j \leq k$, of any edge incident to p. The topological k-neighborhood of a vertex p is denoted $\mathcal{N}_G^{top}(p, k)$.*

The following claim follows easily from the fact that our input graph has maximum degree of 5.

Claim 3.3.14 *Let G be a well-shaped geometric graph for P. For every vertex $p \in P$, we can find its topological k-neighborhood in time $\mathcal{O}(k)$.* □

Now, provided that G is ϵ-far from EMST (but well-shaped), our first approach is to sample uniformly at random a sufficiently large set Q of points in P. Then we add for every point in Q its topological $\frac{72}{\epsilon}$-neighborhood to the sample set. Provided that the set Q is sufficiently large, we prove in Lemma 3.3.18 that if G is ϵ-far from EMST, then the so obtained set will contain all vertices from a certain ϵ-short topological cycle in the EMST-completion of G with probability at least $2/3$. Therefore, this will certify that G is not an EMST by Corollary 3.3.8. In Lemma 3.3.24, we tune the sketched algorithm to slightly improve the complexity bound.

3.3.3 A Simple Property Tester in Well-Shaped Graphs

Now we discuss our first property tester for EMST for well-shaped input graphs. We follow the approach sketched in the beginning of this subsection. According to that description the key issue is to describe the algorithm that finds the sample set S and to prove that with probability at least $\frac{2}{3}$ the input is rejected if G is ϵ-far from EMST. In particular, by our discussion above we know from Lemma 3.3.12 that if a well-shaped graph G is ϵ-far

from EMST, then there are many ϵ-short topological cycles in the EMST-completion of G. Further, by Corollary 3.3.8, in order to reject G it is sufficient to prove that with probability at least $\frac{2}{3}$ the sample S contains all vertices from a certain ϵ-short topological cycle in the EMST-completion of G. We provide a precise description of this algorithm and its analysis below. We assume that G is well-shaped. Every ϵ-short topological cycle either

1. is a cycle consisting of at most $\frac{72}{\epsilon}$ blue edges, or
2. is a path consisting of at most $\frac{72}{\epsilon}$ blue edges having the endpoints connected by a red edge, or
3. is a path consisting of at most $\frac{72}{\epsilon}$ blue edges having the endpoints connected by a path of two red edges, or
4. consists of two paths containing at most $\frac{72}{\epsilon}$ blue edges that are connected to each other by two red edges.

Our first observation is that if there are many ϵ-short topological cycles of type (1) or (2), then we can easily spot them. Indeed, if there are at least $\frac{\epsilon n}{200}$ ϵ-short topological cycles of type (1) or (2) in the EMST-completion of G, then it is enough to take a random subset Q of P of size $\Theta(1/\epsilon)$ to ensure that at least one vertex from any ϵ-short topological cycle will be in Q with probability at least $\frac{2}{3}$.

Lemma 3.3.15 *Let $G = (P, E)$ be a well-shaped geometric graph. If the EMST-completion of G contains at least $\frac{\epsilon n}{200}$ ϵ-short topological cycles of type (1) or (2), then a set $Q \subseteq P$ of size $4000/\epsilon$ chosen uniformly at random contains at least one vertex from any ϵ-short topological cycle.*

Proof : Since G is well-shaped it has a maximum degree of 5 and so the EMST-completion of G has a maximum degree of 10. We conclude that every vertex $p \in P$ is contained in at most 10 ϵ-short topological cycles. Furthermore, the set P_C of all vertices that are contained in at least one ϵ-short topological cycle (of type (1) or (2)) has cardinality at least $\frac{\epsilon n}{2000}$. Now let $Q \subseteq P$ denote a set of size $\frac{4000}{\epsilon}$ taken uniformly at random from P. Then

$$\mathbf{Pr}\big[Q \cap P_C = \emptyset\big] \leq (1 - \frac{|P_C|}{n})^{|Q|} \leq (1 - \frac{\epsilon}{2000})^{|Q|} \leq 1/3 \ .$$

Therefore,

$$\mathbf{Pr}\big[Q \cap P_C \neq \emptyset\big] \geq 2/3 \ .$$

□

Now, we explore the key feature of ϵ-short topological cycles of type (1) or (2): For every vertex v from such a cycle all other vertices from the cycle belong to the topological $\frac{72}{\epsilon}$-neighborhood of v. This motivates us to define the sample set S as the topological $\frac{72}{\epsilon}$-neighborhood of all vertices in Q. Since the set Q contains at least one vertex from any ϵ-short topological cycle of type (1) or (2) with probability at least $2/3$, we can conclude that S contains all vertices from a particular ϵ-short topological cycle of type (1) or (2)

with probability at least $\frac{2}{3}$. Since every vertex of the topological cycle is contained in our sample set we know by Corollary 3.3.8 that the EMST-completion of the subgraph induced by our sample contains a cycle. Thus our property tester rejects the input with probability $\frac{2}{3}$ if it is ϵ-far from EMST.

It is easy to see that the procedure for cycles of type (1) and (2) described above has a query complexity of $\mathcal{O}(\frac{|Q|}{\epsilon}) = \mathcal{O}(\frac{1}{\epsilon^2})$.

Lemma 3.3.16 *Let $G = (P, E)$ be a well-shaped geometric graph and let $Q \subseteq P$ be a set of size $4000/\epsilon$ chosen uniformly at random from P. If the EMST-completion of G contains at least $\frac{\epsilon n}{200}$ ϵ-short topological cycles of type (1) or (2), then the set*

$$S = \bigcup_{p \in Q} \mathcal{N}_G^{top}(p, \frac{72}{\epsilon})$$

contains all vertices of at least one ϵ-short topological cycle with probability at least $2/3$.

Proof : Follows from Lemma 3.3.15 and the observation that a cycle of type (1) and (2) is completely contained in S, if one of its vertices is contained in Q. □

The ϵ-short topological cycles of type (3) and (4) are a little bit more difficult to detect. However, we can still use a very similar approach as for cycles of type (1) or (2), but this time we must find two vertices that belong to the same ϵ-short topological cycle. Suppose that there are at least $\frac{\epsilon n}{200}$ ϵ-short topological cycles of type (3) or (4) in the EMST-completion of G. As before, we first take a random subset Q of P, but this time the size of Q is $\Theta(\sqrt{n/\epsilon})$. Then, we define the sample set S to be the union of the topological $\frac{72}{\epsilon}$-neighborhood of all vertices in Q. We show now that the so defined sample set is sufficient to certify that G (if it is ϵ-far from EMST) is not an EMST by proving an analogous statement to Lemma 3.3.16 for cycles of type (3) and (4):

Lemma 3.3.17 *Let $G = (P, E)$ be a well-shaped geometric graph and let $Q \subseteq P$ be a set of size $80\sqrt{n/\epsilon}$ chosen uniformly at random from P. If the EMST-completion of G contains at least $\frac{\epsilon n}{200}$ ϵ-short topological cycles of type (3) or (4), then the set*

$$S = \bigcup_{p \in Q} \mathcal{N}_G^{top}(p, \frac{72}{\epsilon})$$

contains all vertices of at least one ϵ-short topological cycle with probability at least $2/3$.

Proof : For every ϵ-short topological cycle C of type (3) let us define the set W_C to contain two vertices: one vertex on the blue path in C and the vertex incident to the two red edges in C. Similarly, for every ϵ-short topological cycle C of type (4) let us define the set W_C to contain one vertex from the first blue path in C and one vertex from the second blue path in C.

Since each vertex $p \in P$ belongs to at most 10 ϵ-short topological cycles we can select from the sets W_C the sets W_i, $1 \leq i \leq \frac{\epsilon n}{2000}$, such that the sets W_i are disjoint and for each

i, $1 \leq i \leq \frac{\epsilon n}{2000}$, there is an ϵ-short topological cycle C with $W_i = W_C$. Then we apply Lemma 2.4.1 with $k = \frac{\epsilon n}{2000}$, $l = 2$, and $s = \frac{20n}{\sqrt{\epsilon n}} = 20\sqrt{n/\epsilon}$ and obtain

$$\mathbf{Pr}\left[\exists j \in [k] : (W_j \subseteq Q)\right] \geq 1 - (1 - 1/4)^4 \geq 2/3 \ .$$

Now observe that all vertices of a cycle C are in S, if $W_C \subseteq Q$. Therefore, the lemma follows. □

We prove that the following algorithm is a property tester for EMST:

EMST-TEST-SIMPLE(G, ϵ)
  $s = 80\sqrt{n/\epsilon} + 4000/\epsilon$
  choose a set $Q \subseteq P$ of size s uniformly at random
  $S = \bigcup_{q \in Q} \mathcal{N}_G^{top}(q, \frac{72}{\epsilon})$
  compute the subgraph G_S induced by S
  compute the EMST-completion G_C of G_S
  if G_C contains a cycle **then** reject
  else accept

Lemma 3.3.18 *Let G be a well-shaped geometric graph for P. Then there is a property tester that in time $\mathcal{O}\left(\sqrt{n/\epsilon^3} \cdot \log(n/\epsilon)\right)$ and with query complexity $\mathcal{O}(\sqrt{n/\epsilon^3})$ accepts the input if G is an EMST of P and rejects the input with probability at least $\frac{2}{3}$, if G is ϵ-far from EMST.*

Proof : We claim the algorithm EMST-TEST-SIMPLE is a property tester for EMST. By Corollary 3.3.8 we know that EMST-TEST-SIMPLE accepts, if the input graph $G = (P, E)$ is the EMST.

Now let us consider the case when G is ϵ-far from EMST. Then by Lemma 3.3.12 we know that there are $\epsilon n/100$ ϵ-short topological cycles in the EMST completion of G. It follows that there are $\epsilon n/200$ cycles of type (1) and (2) or $\epsilon n/200$ cycles of type (3) or (4). By Lemma 3.3.16 and 3.3.17 we know that the sample taken by EMST-TEST-SIMPLE contains an ϵ-short topological cycle with probability at least $2/3$. By Corollary 3.3.8 we know that then there is a cycle in the EMST-completion of the subgraph induced by our sample. Hence the algorithm rejects in such a case.

The query complexity of the algorithm is immediate. Its running time follows from Claim 3.3.14 and the fact that the EMST completion of a graph with m vertices can be computed in time $\mathcal{O}(m \log m)$. □

3.3.4 An Improved Property Tester in Well-Shaped Graphs

Now we slightly improve the complexity of the property tester described in Lemma 3.3.18. In our analysis above we were always trying to catch one initially fixed single vertex from each blue path although an ϵ-short topological cycle can contain as many as $\frac{72}{\epsilon}$ vertices.

We now want to take the length of the topological cycles into consideration. Further, we were always taking topological $\frac{72}{\epsilon}$-neighborhoods of all vertices. This strategy should be applied to the cycles that have as many as $\frac{72}{\epsilon}$ edges, but it is not necessary for shorter cycles. Our approach now is to improve the complexity of the property tester by combining these two observations. We want to show that the following algorithm is a property tester for EMST, if the input graph is well-shaped:

EMSTTEST(G, ϵ)
  $s = 1700\sqrt{n/\epsilon} + 192000/\epsilon + 4000/\epsilon$
  $S =$FINDCYCLE(G, s, ϵ)
  compute the subgraph G_S induced by S
  compute the EMST-completion G_C of G_S
  if G_C contains a cycle **then** reject
  else accept

Where the procedure FINDCYCLE is the following:

FINDCYCLE(G, s, ϵ)
  $\mathcal{S}^{(0)} = \emptyset$
  for $i = 1$ **to** $2s$ **do**
    $j = 0$
    pick a vertex $p^{(i)} \in P$ uniformly at random
    while $j \leq \log \frac{72}{\epsilon}$ **do**
      $j = j + 1$
      flip a coin
      if head **then exit**
    $S^{(i)} = S^{(i-1)} \cup \mathcal{N}_G^{top}(p^{(i)}, 2^j)$
  return $S^{(2s)}$

First of all, we observe that algorithm EMSTTEST accepts every Euclidean minimum spanning tree by Corollary 3.3.8.

Thus we have to prove that the algorithm rejects, if the input graph is ϵ-far from EMST. Let us assume that G is well-shaped and ϵ-far from EMST. Then, by Lemma 3.3.12, there are at least $\frac{\epsilon n}{100}$ ϵ-short topological cycles in the EMST-completion of G. Let $\mathfrak{C}_j$, $j = 1, 2, 3, 4$, denote the set of all ϵ-short topological cycles of type (j) in the EMST-completion of G. Now we consider separately cycles in $\mathfrak{C}_1 \cup \mathfrak{C}_2$ and cycles in $\mathfrak{C}_3 \cup \mathfrak{C}_4$. By our discussion above we have either $|\mathfrak{C}_1 \cup \mathfrak{C}_2| \geq \frac{\epsilon n}{200}$ or $|\mathfrak{C}_3 \cup \mathfrak{C}_4| \geq \frac{\epsilon n}{200}$.

Cycles of type (1) and (2). Suppose that G is a geometric graph for P with maximum degree 5 and there are at least εn ϵ-short topological cycles of type (1) or (2) in the EMST-completion of G for $\varepsilon = \frac{\epsilon}{200}$. We first consider the probability that a fixed ϵ-short topological cycle $C \in \mathfrak{C}_1 \cup \mathfrak{C}_2$ is contained in the sample set. Let ℓ denote the number of vertices in cycle C. Then the probability that in round i of the FINDCYCLE procedure

vertex $p^{(i)}$ is one of the ℓ vertices of cycle C is $\frac{\ell}{n}$. Further the probability that the topological neighborhood of $p^{(i)}$ is chosen large enough to contain all vertices of C is at least $\frac{1}{2\ell}$. Overall, for a fixed cycle C the probability that a vertex of C is chosen in round i and that the topological neighborhood of the vertex is large enough is at least $\frac{1}{2n}$. If the cycles are vertex disjoint then it is simple to prove that after $\mathcal{O}(\frac{1}{\epsilon})$ rounds at least one cycle is completely contained in the sample set with constant probability. Unfortunately, in the general case the cycles are not vertex disjoint. To overcome this technical problem we use the planar map representation of G and the following trick for the analysis: Instead of taking the whole topological 2^j-neighborhood of vertex $p^{(i)}$ we assume that our algorithm selects only one of the outgoing edges (in its planar map representation) uniformly at random. Then it includes only the 2^j successors and predecessors of the chosen edge in the planar map representation of G. Clearly, this procedure considers only a subset of the vertices considered in the original procedure. Nevertheless, we can show that the set of vertices we pick using this procedure is still sufficiently large. We can now use the fact that the (directed) cycles are edge disjoint. Assume that we pick a vertex that belongs to a cycle C. Provided that the chosen neighborhood is large enough we still have to choose the correct outgoing edge to have all vertices of C in the sample set. Since our graph has a degree bound of 5 the probability that this edge is chosen is at least $1/5$. Since type (2) cycles consist of a path of $\ell - 1$ (directed) blue edges the probability that $p^{(i)}$ is one of the $\ell - 1$ origins of these edges is $\frac{\ell-1}{n} \geq \frac{\ell}{2n}$ (a directed edge points from its *origin* to its *destination*). Hence the probability that a cycle C of type (1) or (2) is completely contained in the sample set is at least $\frac{1}{20n}$. We know that the cycles are disjoint and so the probability that at least one cycle is completely contained in the sample set in round i is at least $\frac{\varepsilon n}{20n} = \frac{\varepsilon}{20}$. Let X_C denote the indicator random variable for the event that cycle $C \in \mathfrak{C}_1 \cup \mathfrak{C}_2$ is completely contained in the sample set. Then we have for $s \geq 20/\varepsilon = 4000/\epsilon$:

$$\mathbf{Pr}\big[\forall C \in \mathfrak{C}_1 \cup \mathfrak{C}_2 : X_C = 0\big] \leq \big(1 - \frac{\varepsilon}{20}\big)^{2s} \leq 1/3$$

and hence

$$\mathbf{Pr}\big[\exists C \in \mathfrak{C}_1 \cup \mathfrak{C}_2 : X_C = 1\big] \geq 2/3$$

and so we have just proved:

Lemma 3.3.19 *Let G be a geometric graph for P with maximum degree 5 that has at least $\varepsilon n = \frac{\epsilon n}{200}$ topological ϵ-short cycles of type (1) or (2). If algorithm* FINDCYCLE(G, s, ϵ) *is invoked with $s \geq 4000/\epsilon$ then the set $S^{(2s)}$ returned by the algorithm contains an ϵ-short topological cycle with probability at least $2/3$.* □

Cycles of type (3) and (4). Let G be a geometric graph with maximum degree 5. Let us further assume that there are at least εn topological ϵ-short cycles of type (3) and (4) in the EMST-completion of G, for $\varepsilon = \frac{\epsilon}{200}$. We want to show that the sample set computed by algorithm FINDCYCLE contains every vertex of at least one ϵ-short topological cycle with good probability.

Recall that cycles of type (4) consist of 2 paths of blue edges connected by two red edges. Cycles of type (3) are a special case of type (4) cycles: The shorter path has length

0. For each cycle $C \in \mathfrak{C}_3 \cup \mathfrak{C}_4$ let $X_C^{(i)}$ denote the indicator random variable for the event that all vertices of the longer (blue) path of cycle C are in $S^{(i)}$. Let $Y_C^{(i)}$ denote the indicator random variable for the event that all vertices of the shorter (blue) path of cycle C are in $S^{(i)}$. Further let $\Delta^{(i+1)}$ denote the indicator random variable for the event that there is a cycle $C' \in \mathfrak{C}_3 \cup \mathfrak{C}_4$ with $X_{C'}^{(i)} = 0$ and $X_{C'}^{(i+1)} = 1$. We say that a cycle $C \in \mathfrak{C}_3 \cup \mathfrak{C}_4$ is *half-contained* in $S^{(i)}$, if $X_C^{(i)} = 1$. Cycle C is *contained* in $S^{(i)}$, if $X_C^{(i)} = 1$ and $Y_C^{(i)} = 1$.

We analyze the algorithm in two steps. We first show that with high probability many (at least $\varepsilon s/80$) topological ϵ-short cycles are half-contained in the set $S^{(s)}$. Then we show that the set $S^{(2s)}$ contains at least one cycle $C \in \mathfrak{C}_3 \cup \mathfrak{C}_4$ with high probability.

Claim 3.3.20 *Let the outcome of the random choices in round* 1 *to* i *of the* **for***-loop of* FINDCYCLE *be fixed. Then it holds that, if*

$$\sum_{C \in \mathfrak{C}_3 \cup \mathfrak{C}_4} X_C^{(i)} < \frac{\varepsilon s}{2} \tag{3.1}$$

then

$$\mathbf{Pr}\left[\Delta^{(i+1)} = 1\right] \geq \varepsilon/40 \ . \tag{3.2}$$

Proof : Let us assume that Equation (3.1) is true. Then we observe that:

$$\sum_{C \in \mathfrak{C}_3 \cup \mathfrak{C}_4} X_C^{(i)} < \frac{\varepsilon s}{2} \leq \frac{\varepsilon n}{2}$$

since $s \leq n$. We conclude that we have more than $\varepsilon n/2$ cycles in $\mathfrak{C}_3 \cup \mathfrak{C}_4$ that are not half-contained in $S^{(i)}$. If $p^{(i+1)}$ is one of the vertices of the longer path of one of these cycles and if the topological neighborhood included in FINDCYCLE is large enough then we have $\Delta^{(i+1)} = 1$. To estimate the probability for $\Delta^{(i+1)} = 1$ we apply the same trick as in the analysis for the case of type (1) and (2) cycles. This yields immediately (observing the fact that we have $\varepsilon n/2$ cycles instead of εn):

$$\mathbf{Pr}\left[\Delta^{(i+1)} = 1\right] \geq \frac{1}{2\ell} \cdot \frac{\ell}{2n} \cdot \frac{1}{5} \cdot \frac{\varepsilon n}{2} = \varepsilon/40$$

. □

Our next goal is to show that there are at least $\varepsilon s/80$ cycles that are half-contained in $S^{(s)}$.

Claim 3.3.21

$$\mathbf{Pr}\Big[\sum_{C \in \mathfrak{C}_3 \cup \mathfrak{C}_4} X_C^{(s)} \leq \varepsilon s/80\Big] \leq e^{-\varepsilon s/300} \ .$$

Proof :

$$\begin{aligned} & \mathbf{Pr}\Big[\sum_{C\in\mathfrak{C}_4\cup\mathfrak{C}_4} X_C^{(s)} \le \frac{\varepsilon s}{80}\Big] \\ \le\ & \mathbf{Pr}\Big[\sum_{1\le i\le s} \Delta^{(i)} \le \frac{\varepsilon s}{80}\Big] \\ \le\ & \mathbf{Pr}\Big[\sum_{1\le i\le s} B^{(i)} \le \frac{\varepsilon s}{80}\Big] \\ \le\ & \mathbf{Pr}\Big[\sum_{1\le i\le s} B^{(i)} \le (1-\frac{1}{2})\cdot\frac{\varepsilon s}{40}\Big] \end{aligned}$$

where $B^{(i)}$ are independent $0-1$ variables with $\mathbf{Pr}[B^{(i)}=1]=\varepsilon/40$. The latter inequality follows from Claim 3.3.20. We apply a Chernoff bound [56, Inequality (7)] and it follows that

$$\le\ e^{-(\frac{1}{2})^2\frac{\varepsilon s}{40\cdot 2}} = e^{-\varepsilon s/320}\ .$$

□

Let $W^{(i+1)}$ denote the indicator random variable for the event that there exists $C\in\mathfrak{C}_3\cup\mathfrak{C}_4$ with $X_C^{(i)}=1$ and $Y_C^{(i)}=0$ and $Y_C^{(i+1)}=1$.

Claim 3.3.22 *Let the outcome of the random choices in round 1 to i of the procedure* FINDCYCLE *be fixed. If*

$$\sum_{C\in\mathfrak{C}_3\cup\mathfrak{C}_4} X_C^{(i)} > \varepsilon s/80$$

then

$$\mathbf{Pr}\big[W^{(i+1)}\big] \ge \frac{\varepsilon s}{1600\cdot n}\ .$$

Proof :

We assume that there are more than $\varepsilon s/80$ cycles that are half-contained in $S^{(i)}$. Again we use essentially the same trick as in the case of type (1) and (2) cycles. We observe that there is a little problem with cycles of type (3). Since the length of the shorter path is 0 there is no directed edge in this path. Thus we have to modify our approach slightly. We use the following sampling scheme for the analysis: Instead of taking the whole topological 2^j-neighborhood of $p^{(i)}$ we choose a number k between 1 and 6 uniformly distributed. If k is between 1 and 5 we include the 2^j predecessors and successors of the k-th edge incident to $p^{(i)}$. In the case $k=6$ we only include the vertex $p^{(i)}$ in the sample. Then we get that the probability that a cycle C is contained in the sample is at least $\frac{1}{2\ell}\cdot\frac{\ell}{2n}\cdot\frac{1}{6}=\frac{1}{24n}$. We have more than $\varepsilon s/80$ cycles that are half-contained in $S^{(i)}$. Therefore we obtain that:

$$\mathbf{Pr}\big[W^{(i+1)}\big] \ge \frac{\varepsilon s}{1920\cdot n}\ .$$

□

Lemma 3.3.23 *Let* G *be a geometric graph for* P *with maximum degree* 5 *that has at least* $\varepsilon n = \frac{\epsilon n}{200}$ *topological* ϵ*-short cycles of type (3) or (4). Then* FINDCYCLE *is an algorithm with (expected) query complexity* $\mathcal{O}(\sqrt{n/\epsilon}\log(n/\epsilon))$ *that samples a set* $S \subseteq P$*,* $|S| \geq 1700\sqrt{n/\epsilon} + 192000/\epsilon$*, such that the EMST-completion of the subgraph* $G_{S^{(2s)}}$ *induced by* $S^{(2s)}$ *has an* ϵ*-short topological cycle.*

Proof : Let $\varepsilon = \epsilon/200$ and let G be a geometric graph for P with maximum degree 5 that has at least εn topological ϵ-short cycles of type (3) or (4).

$$\begin{aligned} & \mathbf{Pr}\big[\text{There is a cycle in the EMST-completion of the subgraph induced by } S^{(2s)}\big] \\ \geq\ & 1 - \Big(\mathbf{Pr}\big[\frac{1}{s}\sum_{C\in\mathfrak{C}_3\cup\mathfrak{C}_4} X_C^{(s)} \leq \frac{\varepsilon}{80}\big] + \Big(1 - \frac{\varepsilon s}{1920n}\Big)^s\Big) \end{aligned}$$

by Claim 3.3.22. Choosing $s \geq 1700\sqrt{n/\epsilon} + 192000/\epsilon$ it follows together with Claim 3.3.21 for $n \geq 4$:

$$\geq\ 1 - (e^{-2} + e^{-5}) \geq \frac{4}{5}\ .$$

□

Obtaining Deterministic Query Complexity. It is easy to modify the algorithm such that the query complexity has an upper bound of $\mathcal{O}(\sqrt{n/\epsilon}\log(n/\epsilon))$ by insignificantly increasing the error of the algorithm. We can do this in the following way: We run algorithm EMSTTEST and stop, if it either accepts or rejects or if the sample size becomes too large. Let X_S denote the random variable for the size of $S^{(2s)}$. We stop the algorithm and accept the input, if we find out that the size of $S^{(2s)}$ becomes larger than $10 \cdot \mathbf{E}[X_S]$. By Markov inequality we have:

$$\mathbf{Pr}\big[X_S \geq 10\,\mathbf{E}[X_S]\big] \leq 1/10\ .$$

Hence it follows that our new algorithm rejects a geometric graph that is ϵ-far from EMST with probability $4/5 - 1/10 \geq 2/3$. Thus it is a property tester with a deterministic bound of $\mathcal{O}(\log(n/\epsilon)\cdot\sqrt{n/\epsilon})$ on the query complexity of our algorithm (rather than expected query complexity).

Lemma 3.3.24 *Let* G *be a well-shaped geometric graph for* P*. Then there is a property tester that in time* $\mathcal{O}(\log^2(n/\epsilon)\cdot\sqrt{n/\epsilon})$ *and with query complexity of* $\mathcal{O}(\log(n/\epsilon)\cdot\sqrt{n/\epsilon})$ *accepts the input* G*, if* G *is an EMST of* P *and that rejects the input with probability at least* $\frac{2}{3}$ *if* G *is* ϵ*-far from EMST.*

Proof : Follows from Lemma 3.3.12, Lemma 3.3.19 and Lemma 3.3.23. □

3.3.5 A Property Tester in Graphs with Maximum Degree 5

Now we want to remove the well-shaped condition for input graphs. In this subsection we do the first step towards that goal. We develop property tester for connectivity and crossing-free EMST-completions. Then we replace the well-shaped condition for EMSTTEST by the assumption that the input graph has maximum degree 5. Before EMSTTEST is invoked we test if the input graph is $\epsilon/200$-far from connected and if its EMST-completion has a crossing-free straight line embedding. Thus we may assume that EMSTTEST gets an input graph that is $\epsilon/200$-close to connected and $\epsilon/200$-close to having an EMST-completion with a crossing-free straight line embedding (if this is not the case, the property testers for connectivity and crossing-free EMST-completions reject). This way we develop a property tester for graphs with maximum degree 5. The degree bound is then removed in the last subsection.

Testing Connectivity. In the first phase we test whether the input graph is connected.

Definition 3.3.25 *A geometric graph* G *for* P *is* ϵ-far from connected *if one has to add more than* $\epsilon \cdot n$ *edges to* G *to obtain a connected graph. If* G *is not* ϵ*-far from connected, then we call it* ϵ-close *to connected.*

Remark 1 *Let us notice here the equivalent characterization of geometric graphs for* P *that are* ϵ*-far from connected — these are geometric graphs for* P *having more than* $\epsilon \cdot n+1$ *connected components.*

Since the property of being connected does not depend on the positions of the input points in P, we can use a property tester for connectivity in graphs. In [52] it has been proven that connectivity of bounded degree graphs can be tested efficiently:

Lemma 3.3.26 *[52] Let* G *be a graph with degree bound* d*. Connectivity of* G *can be tested with* $\mathcal{O}(\frac{\log^2(1/\epsilon d)}{\epsilon d})$ *time and query complexity in the bounded length adjacency list model (see Chapter 2)* [3].

We can immediately apply this result to geometric graphs:

Corollary 3.3.27 *Let* G *be a geometric graph for* P *with maximum degree 5. There is a property tester that in time* $\mathcal{O}(\frac{\log^2(1/\epsilon)}{\epsilon})$ *and with a query complexity of* $\mathcal{O}(\frac{\log^2(1/\epsilon)}{\epsilon})$ *accepts the input if* G *is connected and rejects the input with probability at least* $\frac{2}{3}$ *if* G *is* ϵ*-far from connected.*

Proof : We run the tester from [52] with $d = 5$ and $\epsilon' = \epsilon/5$. □

[3] In the bounded degree graph model a graph is ϵ-far from connected if one has to add more than $\epsilon d n$ edges to obtain a connected graph.

Testing Crossing-Free EMST-Completions. Next, we design a tester that accepts the input graph, if it is the EMST and rejects it if the straight-line embedding of its EMST-completion is ϵ-far from crossing-free. We proceed in two steps. First, our property tester checks for pairs of intersecting blue edges and then it tries to find intersections between blue and red edges; red edges cannot intersect because they are edges of an EMST. We use the following distance measure.

Definition 3.3.28 *The straight-line embedding of a geometric graph* G *for* P *is* ϵ-far from crossing-free *if one has to remove more than* $\epsilon \cdot n$ *edges in* G *to obtain a crossing-free straight-line embedding. If the straight-line embedding of* G *is not* ϵ*-far from crossing-free, then we call it* ϵ-close *to crossing-free.*

We first use the tester DISJOINTNESS developed in Section 3.1 to find intersections between blue segments (induced by blue edges). Since G has maximum degree 5 it has at most $5n$ edges. Therefore, since one can verify in time $\mathcal{O}(n \log n)$ if a geometric graph with n vertices has crossing-free straight-line embedding [16] (the number of edges must be $\mathcal{O}(n)$; otherwise there must be a crossing), Theorem 1 implies the following result.

Lemma 3.3.29 *Let* G *be a geometric graph with maximum degree 5. There is a property tester that in time* $\mathcal{O}(\sqrt{n/\epsilon}\log(n/\epsilon))$ *and with the query complexity of* $\mathcal{O}(\sqrt{n/\epsilon})$ *accepts the input if the straight-line embedding of* G *is crossing-free and rejects the input with probability at least* $\frac{2}{3}$ *if the straight-line embedding of* G *is* ϵ*-far from crossing-free.* □

It remains to design a property tester for red-blue intersections in the EMST-completion of G [4]. A geometric graph with red and blue edges has a straight-line embedding without red-blue intersections, if there is no intersection between the corresponding red and blue segments. Similarly, the straight-line embedding of a geometric graph whose edges are colored blue and red is ϵ*-far from having no red-blue intersections*, if one has to delete more than an ϵ-fraction of its edges to remove all red-blue intersections.

The main difficulty with testing for red-blue intersections in the EMST-completion of G is that the red edges are only defined implicitly. The following lemma shows a relation between the endpoints of red and blue edge intersecting each other.

Lemma 3.3.30 *Let* $\overline{pq}$ *be a red and* $\overline{xy}$ *be a blue segment in the EMST-completion of* G. *If* $\overline{pq}$ *and* $\overline{xy}$ *intersect each other, then either* $\overline{pq}$ *is not in the EMST of every set containing* $\{p, q, x\}$ *or it is not in the EMST of every set containing* $\{p, q, y\}$.

Proof : The points p, q, x, y are in convex position because the segments $\overline{pq}$ and $\overline{xy}$ intersect. We consider the quadrilateral $pxqy$ (see figure 3.7). Let us call the inner angles in the quadrilateral at vertices p, q, x, y to be α, β, γ, and δ, respectively. Let us recall that the longest edge of a triangle is opposite of the largest angle.

[4] More precisely, we do not design a property tester for the property of having no red-blue intersections. Our algorithm might reject an input graph if its EMST-completion has no red-blue intersections. However, if the input graph is the EMST then it is always accepted by our algorithm.

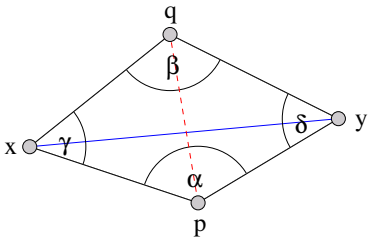

Figure 3.7: A red blue intersection. The red edge is dotted.

If $\alpha < \frac{\pi}{2}$ and $\beta < \frac{\pi}{2}$ then γ or δ is larger than $\frac{\pi}{2}$ because $\alpha+\beta+\gamma+\delta = 2\pi$. Without loss of generality, let $\gamma > \frac{\pi}{2}$. Then, segment $\overline{pq}$ is the longest edge of triangle pqx and thus is cannot be the EMST of $\{p, q, x\}$. By Lemma 3.3.7 this is a contradiction to the fact that (p, q) is an EMST edge. Hence we must have either $\alpha \geq \frac{\pi}{2}$ or $\beta \geq \frac{\pi}{2}$.

If $\alpha \geq \frac{\pi}{2}$ then segment $\overline{xy}$ is the longest edge in triangle pxy. Hence edge (x, y) is not contained in the EMST of p, x, y. By Lemma 3.3.7 it is also not contained in every EMST of a subset of P that contains p, x, y.

If $\beta \geq \frac{\pi}{2}$ then segment $\overline{xy}$ is the longest edge in triangle qxy. Hence edge (x, y) is not contained in the EMST of q, x, y. By Lemma 3.3.7 it is also not contained in every EMST of a subset of P that contains q, x, y. □

This lemma shows that each red-blue intersection has a "witness" consisting of one point and one edge of the input graph. Our property tester for red-blue intersections is similar to the DISJOINTNESS tester. We modify the disjointness property in the following way: We say that two points $p, q \in P$ intersect, if there is an (blue) edge $e = (p, x)$ (or $e = (q, x)$) incident to p (or q) such that the other point is a witness that e is not in the EMST of P, that is e is not in the EMST of p, q, x. The following algorithm is a property tester for red-blue intersections:

REDBLUETEST(G, ϵ)
Choose a set $S' \subset P$ of size $16\sqrt{5n/\epsilon}$ uniformly at random
Let $S = S' \cup \mathcal{N}(S')$ where $\mathcal{N}(S')$ denotes the set of neighbors of points in S'
Let G_S denote the subgraph induced by S
if the EMST-completion of G_S has a cycle **then** *reject*
else *accept*

The analysis of the algorithm is similar to the analysis of the algorithm DISJOINTNESS.

If the input graph is the EMST then there are no intersections among the points in P. If the EMST-completion of the input graph is ϵ-far from having no red-blue intersections then by Lemma 3.3.30 it follows that the point set P is ϵ-far from being disjoint, if the EMST-completion is ϵ-far from having no red-blue intersections. Thus we have:

Lemma 3.3.31 *Let G be a geometric graph for P with maximum degree 5. Algorithm* RedBlueTest *runs in time $\mathcal{O}(\sqrt{n/\epsilon}\log n)$ and with the query complexity of $\mathcal{O}(\sqrt{n/\epsilon})$ and accepts the input graph G if it is the Euclidean Minimum Spanning Tree and rejects the input with probability at least $\frac{2}{3}$ if the straight-line embedding of the EMST-completion of G is ϵ-far from having no red-blue intersection.*

Proof : If $G = (P, E)$ is the EMST then its EMST-completion is crossing-free. Thus algorithm RedBlueTest accepts G.

Let G_C denote the EMST-completion of G and let us assume that G_C is ϵ-far from having no red-blue intersections. By Lemma 3.3.30 we can apply the following procedure $k = \epsilon n/20$ times: pick a pair of intersecting (with the definition from above) points $\{p, q\} = W_i$, $i \in [k]$, and remove all edges incident to p and q from G_C. By the degree bound, we have to remove at most 10 edges for each of the two vertices. Therefore, we can apply this procedure at least k times.

In order to prove that RedBlueTest rejects G with probability at least $2/3$ if G_C is ϵ-far from having no red-blue intersections we show that with probability at least $2/3$ one of the pairs W_i, $i \in [k]$, is in S'. We apply Lemma 2.4.1 and obtain:

$$\mathbf{Pr}\left[\exists j \in [k] : (W_j \subseteq S')\right] \geq 1 - (1 - 3/4)^4 \geq 2/3$$

It remains to show that the algorithm rejects, if there is a pair $W_i \subseteq S'$. If $W_i = \{p, q\} \subseteq S'$ then there exists $e = (p, x)$ (or $e = (q, x)$) such that e is not in the EMST of p, q, x. By Lemma 3.3.7 e is not in the EMST of S. Hence S must have a cycle and RedBlueTest rejects. □

Finally, we can combine Lemma 3.3.29 and Lemma 3.3.31 to obtain.

Lemma 3.3.32 *Let G be a geometric graph for P with maximum degree 5. There is an algorithm that in time $\mathcal{O}(\sqrt{n/\epsilon}\log(n/\epsilon))$ and with a query complexity of $\mathcal{O}(\sqrt{n/\epsilon})$ accepts G, if G is the EMST of P and rejects G with probability at least $\frac{2}{3}$ if the straight-line embedding of its EMST-completion is ϵ-far from crossing-free.*

Proof : Let G be ϵ-far from having a crossing-free EMST-completion. Then, either the straight-line embedding of G is $\frac{\epsilon}{2}$-far from crossing-free or the EMST-completion of G is $\frac{\epsilon}{2}$-far from having no red-blue intersections. Applying Lemma 3.3.29 and Lemma 3.3.31 with $\epsilon = \frac{\epsilon}{2}$ shows that G is rejected with probability at least $\frac{2}{3}$. Since the tester for blue-blue intersection and the tester for red-blue intersections both accept the EMST we have completed the proof of Lemma 3.3.32. □

Extension to Degree Bounded Graphs. Now we want to design a property tester for arbitrary degree bounded input graphs. We first extend the planar map representation for well-shaped graphs to a *planar map like* representation for general graphs in the following straightforward way:

For every vertex p in G we (cyclically) sort incident outgoing edges in clockwise order around the vertex p with respect to the Euclidean positions of the edges endpoints. For sake of completeness we restate Definitions 3.3.10, 3.3.11, and 3.3.13 for general graphs (these generalizations are immediate):

Definition 3.3.33 *Let* G *be a geometric graph for* P *and let* $\widetilde{G}$ *be the corresponding planar map like representation. For each edge* $e = [pq\rangle$ *in* $\widetilde{G}$ *the* successor *of* e *is the edge* $[q, r\rangle$ *in* $\widetilde{G}$ *with* r *being the vertex incident to* q *that precedes* p *in the adjacency list of* q *(sorted in clockwise order around* q*); if* q *has degree one, then* $r = p$.

For edge $e = [p, q\rangle$ *in* G *the* kth successor, $k \geq 0$, *is defined recursively as follows: the* 0*th successor of* $[p, q\rangle$ *is* $[p, q\rangle$ *itself, and for* $k > 0$, *the* k*th successor of* $[p, q\rangle$ *is the successor of the* $(k-1)$*st successor of* $[p, q\rangle$.

Similarly, edge e *is the* predecessor *of edge* e' *if edge* e' *is the successor of edge* e, *and* e *is the* kth predecessor *of* e' *if* e' *is the* k*th successor of* e.

Definition 3.3.34 *Let* G *be a geometric graph for* P *and let* $C = (p_0, p_1, \ldots, p_k)$ *be a cycle in the planar map of the EMST-completion of* G. *Then* C *is called* topological, *if for any two consecutive edges on the cycle* $[p_i, p_{i+1}\rangle$ *and* $[p_{i+1}, p_{i+2}\rangle$, $[p_{i+1}, p_{i+2}\rangle$ *is the successor of* $[p_i, p_{i+1}\rangle$. *We also call the corresponding cycle in* G *topological.*

Definition 3.3.35 *Let* G *be a geometric graph for* P. *For any vertex* $p \in P$, *we define its* topological k-neighborhood *as the set of vertices that are the endpoints of the edges that are either the* i*th successor,* $0 \leq i \leq k$, *of any edge incident to* p, *or are the* j*th predecessor,* $0 \leq j \leq k$, *of any edge incident to* p. *The topological* k*-neighborhood of a vertex* v *is denoted* $\mathcal{N}_G^{top}(v, k)$.

Then we recall that Lemmas 3.3.19 and 3.3.23 do not require that G is connected not that the EMST-completion. Hence it suffices to show that the bound of Lemma 3.3.12 carries over to general graphs:

Lemma 3.3.36 *Let* $G = (P, E)$ *be a geometric graph for* P *that is* $\epsilon/200$*-close to connected and* $\epsilon/200$*-close to having a crossing-free straight-line embedding. If* G *is* ϵ*-far from EMST, then there are at least* $\frac{\epsilon n}{100}$ ϵ*-short topological cycles in the EMST-completion of* G.

Proof : Let G_C denote the EMST-completion of $G = (V, E)$. Since G_C is $\epsilon/200$-close to crossing-free we can delete a set E_D of at most $\epsilon n/200$ edges from G_C to make it crossing-free. Then we can insert a set E_I of at most $\epsilon n/100$ edges to make G connected (and still keep its EMST-completion crossing-free). This is always possible since we can insert EMST edges to connect the disconnected components. Let us denote the resulting graph $G'_C = (V, E \setminus E_D \cup E_I)$.

Using the same arguments as in the proof of Lemma 3.3.12 we obtain that there are at least $\epsilon n/24$ ϵ-short topological cycles in G'_C. Now we reverse our modifications of the EMST-completion and we remove the edges E_I from G'_C. We observe that the removal of a single edge can destroy at most two cycles since the cycles are disjoint (in their planar map representation). Hence the removal of E_D destroys at most $\epsilon n/50$ ϵ-short topological cycles. Then we reinsert the removed edges E_D. Now each edge can destroy at most 2 cycles. Hence the re-insertion of E_D destroys at most $\epsilon n/100$ ϵ-short topological cycles. Counting the remaining cycles we get that at least $\epsilon n/100$ ϵ-short topological cycles are in G_C. □

We conclude:

Lemma 3.3.37 *Let* G *be a geometric graph for* P *with maximum degree 5. Then there is a property tester that in time* $\mathcal{O}(\log^2(n/\epsilon) \cdot \sqrt{n/\epsilon})$ *and with query complexity of* $\mathcal{O}(\log(n/\epsilon) \cdot \sqrt{n/\epsilon})$ *accepts the input if* G *is an EMST of* P *and rejects the input with probability at least* $\frac{2}{3}$ *if* G *is* ϵ*-far from EMST.*

Proof : Follows from Lemma 3.3.19, Lemma 3.3.23, and Lemma 3.3.36. □

3.3.6 A Property Tester in General Graphs

It remains to remove the degree bound condition for the input graph. In order to do this, we first describe a low degree tester:

A property tester for low degree. We first define when a graph is far from having low degree vertices:

Definition 3.3.38 *A geometric graph* $G = (P, E)$ *for* P *is* ϵ-far from having low degree *if one has to remove more than* $\epsilon \cdot n$ *edges in* G *to obtain a graph having maximum degree smaller than or equal to five.*

If G *is not* ϵ*-far from having low degree, then we call it* ϵ-close *to having small degree.*

It is easy to see that if G is ϵ-far from having small degree, then there are at least $\sqrt{\epsilon \cdot n}$ vertices in G either having degree greater than five or having an adjacent vertex with degree greater than five. Therefore the simple algorithm that picks a random set S of $4\sqrt{n/\epsilon}$ points in P and tests if every point $p \in S$ has the degree smaller than or equal to 5 and if so then it tests if every neighbor of $p \in S$ has degree smaller than or equal to 5 [5], will detect with probability greater than or equal to $\frac{2}{3}$ every geometric graph G that is ϵ-far from having small degree.

Lemma 3.3.39 *Let* G *be a geometric graph for* P*. There is a property tester that in time* $\mathcal{O}(\sqrt{n/\epsilon})$ *and with the query complexity of* $\mathcal{O}(\sqrt{n/\epsilon})$ *accepts the input if* G *has the maximum degree smaller than or equal to* 5 *and rejects the input with probability at least* $\frac{2}{3}$ *if* G *is* ϵ*-far from having small degree.*

Proof : Clearly, our algorithm accepts every graph having maximum degree of five. Let us assume that G is ϵ-far from having small degree. A vertex that has degree more than five or that is adjacent to a vertex with degree more than 5 is called a *heavy vertex*. Let S denote a sample of size $4\sqrt{\epsilon n}$ chosen uniformly at random from P.

$$\mathbf{Pr}\big[S \text{ contains no heavy vertex}\big] \leq (1 - 1/\sqrt{n/\epsilon})^{4\sqrt{n/\epsilon}} \leq 1/3 \ .$$

It follows that

$$\mathbf{Pr}\big[S \text{ contains a heavy vertex}\big] \geq 2/3$$

Hence our algorithm rejects every graph that is ϵ-far from having small degree with probability at least $2/3$. Thus it is a property tester. □

[5] Since the degree of every $p \in S$ is less than or equal to 5, such a test can be performed in a constant time per vertex p.

The Tester for General Graphs. To obtain a tester for general graphs we have to modify the tester for graph with degree bound of 5 in the following way: We first test whether the input graph is $\epsilon/2$-far from having low degree (using the tester described below). Then we run the property tester for graphs with maximum degree of 5 after applying the following modifications and with distance parameter $\varepsilon = \epsilon/2$:

- If during the course of the algorithm we encounter a vertex with degree more than 5, we immediately reject.
- For each vertex $v \in S$ we also include every neighbor of v in G into the sample set. This can be done without asymptotically increasing the running time of the algorithm (because we reject, if we encounter a vertex with degree more than 5).

Clearly, the above modifications do not affect the case when the input graph is the EMST of the point set: The algorithm will still accept the input graph. Thus let us consider the case when the input graph G is ϵ-far from EMST. If the low degree tester rejects the input graph, we are done. Thus let us assume that the input graph passes this test (and thus is $\epsilon/2$-close to having low degree):

Let us assume that G is $\epsilon/2$-close to having low degree but ϵ-far from EMST. We call a vertex of G a *distinguisher*, if it either has degree more than 5 or if it has a neighbor whose degree is greater than 5. Now we define the graph G' to be a graph obtained from G by deleting a minimal set of edges such that G' has a maximum degree of 5. Since we deleted less than $\epsilon n/2$ edges from G to obtain G' we conclude that G' is $\epsilon/2$-far from EMST.

In order to analyze the behavior of the modified algorithm we consider the (unmodified) algorithm for graphs with maximum degree 5. First of all, we observe that the modified algorithm always rejects, if there is a distinguisher in the sample chosen by the unmodified algorithm. But if there is no distinguisher in the sample chosen by the unmodified algorithm then the graph 'looks' like the graph G' which has maximum degree 5 and is $\epsilon/2$-far from EMST. If we run the unmodified algorithm on input G' it rejects with probability $2/3$. Thus the modified algorithm always rejects when the unmodified algorithm rejects G'. We conclude that the modified algorithm rejects G with probability at least $2/3$ because it either rejects or it behaves like the unmodified algorithm with input G'. Hence we proved:

Theorem 4 *There is a property tester for the EMST property with a running time of* $\mathcal{O}(\sqrt{\epsilon n}) \cdot \log^2(n/\epsilon))$ *and with a query complexity of* $\mathcal{O}(\sqrt{\epsilon n} \cdot \log(n/\epsilon))$. □

4 Efficient Property Testers

In the previous chapter we have seen some property testers and their analyses. This leads to the general question if it is possible to characterize all properties of a certain class of objects (e.g., all properties of point sets) that can be tested efficiently. First of all, it is necessary to specify what is meant by "efficiently testable": A property Π is *efficiently testable*, if there is a property tester for Π that has query complexity independent of the size of the input (for constant distance parameter ϵ). One of the major open questions in the field is to characterize for certain classes of objects all properties that are efficiently testable and there has been some effort to achieve this goal: For graph properties it is known that all first order graph properties that do not have a $\forall\exists$ quantification are efficiently testable in the adjacency matrix model [5]. In the same model it has also been proven that a large class of graph partitioning problems can be tested efficiently [51]. For matrix properties it is known that certain monotonicity properties can be tested efficiently [45].

In this chapter we try to find a characterization of efficiently testable properties for functions from an arbitrary finite domain $\mathcal{D}$ into an arbitrary range $\mathcal{R}$. Using the very general formulation of a property (see Definition 2.1.1) we can use our framework for a large class of objects and - as shown in this chapter - also for a large class of properties. In particular, we show that our framework can be used to prove that certain clustering properties over point sets, a reversal distance property over permutations, and the k-coloring property over ℓ-uniform hypergraphs can be tested efficiently. For some of these problems a property testing algorithm has been analyzed before [4]. Our goal is here to show that our framework is fairly general and that it leads to elegant proofs that highlight the combinatorial features of the problems. We can also show that every hereditary graph property (every graph property that is closed under taking induced subgraphs) is efficiently testable, if and only if there is a proof of its testability using our framework.

We emphasize that our results only hold for property testers with *one-sided error*. Our result for hereditary graph properties is existential. It shows that our framework can be applied to a large variety of problems. It would be very interesting to show that our characterization can be used to show that certain large classes of functions are efficiently testable. This is one of the major open questions regarding our framework.

4.1 Abstract Combinatorial Programs

In this section we introduce *abstract combinatorial programs*. An abstract combinatorial program (ACP) is a combinatorial structure defined over a *ground set* of *atom items*. In

this combinatorial structure atom items may be arranged into sets. These sets describe possible "basic" configurations and they are called the *bases* of the abstract combinatorial program. Further there is a relationship between atom items and bases: Each atom item is either *consistent* with a given basis or it *violates* it. This relationship is described by a violation function.

ACPs can be used to highlight combinatorial features of property testing problems. In typical applications the ground set depends on the problem under consideration and corresponds in a natural way to the basic items of the considered problem. When we want to apply ACPs to graph problems then the ground set may be the set of vertices of the graph and when we consider properties of point sets the ground set may be the set of points.

The set of *bases* describes possible "basic" configurations of the corresponding problem. If we consider a graph coloring problem then a basis may correspond to a subset of vertices W together with an associated coloring of W. For technical reasons we define every basis as a pair (W, ℓ), where W is a subset of the ground set and ℓ is an index describing a configuration of W (for example, in the graph-coloring example above, it is a coloring of vertices in W).

The *violation function* describes for each basis $\mathfrak{b}$ and each atom item $\mathfrak{x}$ if $\mathfrak{x}$ is consistent with $\mathfrak{b}$ or not. If $\mathfrak{x}$ is not consistent with $\mathfrak{b}$ then we say $\mathfrak{x}$ *violates* $\mathfrak{b}$. If every atom item is consistent with a basis then we call this basis *feasible*. Normally, the violation function depends on the input instance. For the graph-coloring example above one could define the violation function such that a vertex v violates a basis (colored vertex set W) if and only if in the input graph the k-coloring of W cannot be extended to a proper k-coloring of $W \cup \{v\}$.

Formally, we define an abstract combinatorial program in the following way.

Definition 4.1.1 *An* abstract combinatorial program *(ACP) is a triple* $(\mathcal{C}, \mathcal{B}, \varpi)$*, where*

- $\mathcal{C}$ *is a finite set called* ground set*,*
- $\mathcal{B} \subseteq \{(W, \ell) : W \subseteq \mathcal{C}, \ell \in \mathbb{N}\}$ *is a set of* bases*, and*
- $\varpi : \mathcal{B} \to 2^{\mathcal{C}}$ *is a* violation function*. For each basis* $\mathfrak{b} \in \mathcal{B}$ *the set* $\varpi(\mathfrak{b})$ *is the set of atom items violating* $\mathfrak{b}$*.*

A basis $\mathfrak{b}$ *is* feasible *if* $\varpi(\mathfrak{b}) = \emptyset$*. An abstract combinatorial program is* feasible *if it has a feasible basis.*

We also would like to remark here that in the context of randomized incremental algorithms *configuration spaces* have been introduced as an analysis tool (e.g., see [35]). Although the structure of a configuration space is similiar to that of an ACP it would be misleading to describe our framework in terms of configuration spaces as the intuitive meaning of the corresponding components of the definitions differs widely.

We are now interested in the problem of testing the feasibility of ACPs. In order to do so we need some further definitions. The first definition introduces the term *dimension* to denote the maximum number of atom items involved in a basis. Then we denote by the width of an ACP the maximum number of different bases with the same set of atom items:

Definition 4.1.2 (ACP Dimension) *Let $\mathcal{P}$ be an abstract combinatorial program. The* dimension *of $\mathcal{P}$ (denoted $\dim(\mathcal{P})$) is defined as $\max\{|W| : (W,\ell) \in \mathcal{B}\}$. The* width *of $\mathcal{P}$ denoted (width($\mathcal{P}$)) is defined as $\max\{\ell : (W,\ell) \in \mathcal{B}\}$.*

Since we are interested in property testing we want to investigate in "local" properties of ACPs to conclude if the ACP as a whole is feasible or not. For this purpose we need some further definitions. We say that a basis is *feasible* for a set of atom items, if none of these items violates the basis. A basis is *covered* by a set of atom items, if the basis contains only atom items from this set. And a set of atom items contains a *self-feasible basis*, if there is a feasible basis that is covered by the set:

Definition 4.1.3 (Self-feasible bases) *Let $\mathcal{P} = (\mathcal{C}, \mathcal{B}, \varpi)$ be an abstract combinatorial program. We say a basis $\mathfrak{b} = (W,\ell) \in \mathcal{B}$ is* covered *by a subset $\mathrm{C}^* \subseteq \mathcal{C}$ if $W \subseteq \mathrm{C}^*$. We say that a basis $\mathfrak{b}$ is* feasible *for a subset $\mathrm{C}^* \subseteq \mathcal{C}$, if no $\mathrm{c} \in \mathrm{C}^*$ violates $\mathfrak{b}$. We say a subset $\mathrm{C}^* \subseteq \mathcal{C}$* contains a self-feasible basis *if there is a basis $\mathfrak{b}$ that is covered by C^* and that is feasible for C^*.*

In the next section we want to design a property tester for feasibility of ACPs. We assume that the algorithm has the possibility to determine for a set $S \subseteq \mathcal{C}$ if S has a self-feasible basis or not. The size of the set S should be as small as possible (the size of this set could be seen as the query complexity of the algorithm). Since a property tester has 1-sided error it is necessary that we can always determine if the input ACP is feasible. But then this would require that an ACP that consists of a single feasible basis is always accepted. The only chance to ensure this is to set $S = \mathcal{C}$. Therefore, we consider only *monotone* ACPs. If a monotone ACP $\mathcal{P}$ with dimension $\dim(\mathcal{P})$ is feasible then every $S \subseteq \mathcal{C}$ with $|S| \geq \dim(\mathcal{P})$ has a self-feasible basis.

Definition 4.1.4 (Monotonicity) *Let $\mathcal{P} = (\mathcal{C}, \mathcal{B}, \varpi)$ be an abstract combinatorial program with dimension $\dim(\mathcal{P})$. $\mathcal{P}$ is called* monotone *if it is either not feasible, or if (it is feasible and) every subset $S \subseteq \mathcal{C}$ with $|S| \geq \dim(\mathcal{P})$ contains a self-feasible basis.*

4.1.1 Testing Abstract Combinatorial Programs

In this section we design a property tester for monotone ACPs. We first have to define when an ACP is far from feasible. We do this directly without specifying a distance measure between ACPs:

Definition 4.1.5 *An abstract combinatorial program is* ϵ-far from feasible *if every basis is violated by more than $\epsilon \cdot |\mathcal{C}|$ objects from the ground set $\mathcal{C}$.*

A property tester for ACPs is an algorithm that (i) accepts every feasible ACP and (ii) rejects with probability at least $\frac{2}{3}$ every ACP that is ϵ-far from feasible. In the following theorem we analyze a property tester for certain ACPs:

Theorem 5 (Testing ACPs) *Let $\mathcal{P} = (\mathcal{C}, \mathcal{B}, \varpi)$ be an abstract combinatorial program with dimension at most δ and width at most ρ. Then there exists s with*

$$s = \Theta(\epsilon^{-1} \cdot (\delta \cdot \ln(\delta/\epsilon) + \ln \rho))$$

such that the following algorithm

> ACP-TESTER($\mathcal{P}, \epsilon$)
> *Sample a set S of s objects from $\mathcal{C}$ uniformly at random*
> **if** *S contains a self-feasible basis* **then** *accept $\mathcal{P}$*
> **else** *reject $\mathcal{P}$*

1. *accepts $\mathcal{P}$, if $\mathcal{P}$ is monotone and feasible,*

2. *and rejects $\mathcal{P}$ with probability at least $2/3$, if $\mathcal{P}$ is ϵ-far from feasible .*

Proof : Let $\mathcal{P} = (\mathcal{C}, \mathcal{B}, \varpi)$ be an ACP that is ϵ-far from feasible. Further let $\mathcal{P}$ have dimension at most δ and width at most ρ. For a basis $b = (W, \ell)$ let $\mathcal{E}_b$ be the random event (with respect to the random choice of S) that $W \subseteq S$ and that none of the elements from $\varpi(b)$ is in S. Now, in order to prove the theorem it is sufficient to show that with the probability larger than or equal to $\frac{2}{3}$ for none of $b \in \mathcal{B}$ the event $\mathcal{E}_b$ holds.

For every r, $0 \leq r \leq \delta$, let Δ_r be the set of all $b = (W, \ell) \in \mathcal{B}$ with $|W| = r$. Let us fix an arbitrary $b \in \Delta_r$. Then we have

$$\mathbf{Pr}[\mathcal{E}_b] \;\leq\; \frac{\binom{(1-\epsilon)n-r}{s-r}}{\binom{n}{s}} \;.$$

Since $\mathcal{P}$ has dimension at most δ and width at most ρ, we have $|\Delta_r| \leq \rho \cdot \binom{n}{r}$ for every $r \geq 0$. Furthermore, we have $|\Delta_r| = 0$ for all $r > \delta$. Therefore, by the union bound we obtain

$$\begin{aligned}
\mathbf{Pr}[\exists b \in \mathcal{B} : \mathcal{E}_b] \;&\leq\; \sum_{b \in \mathcal{B}} \mathbf{Pr}[\mathcal{E}_b] \;=\; \sum_{r=0}^{\delta} \sum_{b \in \Delta_r} \mathbf{Pr}[\mathcal{E}_b] \\
&\leq\; \rho \cdot \sum_{r=0}^{\delta} \binom{n}{r} \cdot \frac{\binom{(1-\epsilon)n-r}{s-r}}{\binom{n}{s}} \\
&=\; \rho \cdot \sum_{r=0}^{\delta} \frac{\binom{n}{r}}{\binom{n}{s}} \cdot \binom{(1-\epsilon)n-r}{s-r} \\
&=\; \rho \cdot \sum_{r=0}^{\delta} \binom{s}{r} \frac{(n-s)!}{(n-r)!} \cdot \frac{((1-\epsilon)n-r)!}{((1-\epsilon)n-s)!}
\end{aligned}$$

$$
\begin{aligned}
&= \rho \cdot \sum_{r=0}^{\delta} \binom{s}{r} \cdot \frac{((1-\epsilon)n-r)\cdots((1-\epsilon)n-s+1)}{(n-r)\cdots(n-s+1)} \\
&\leq \rho \cdot \sum_{r=0}^{\delta} s^r \cdot (1-\epsilon)^{s-r} \leq \rho \cdot \sum_{r=0}^{\delta} s^r \cdot e^{-\epsilon(s-r)} \\
&\leq \rho \cdot \delta \cdot s^{\delta} \cdot e^{-\epsilon(s-\delta)} \leq \rho \cdot \delta \cdot s^{\delta} \cdot e^{-\epsilon(s-s/2)} \ ,
\end{aligned}
$$

where we assume in the last inequality that $s \geq 2\delta$. Then we set $s' := (\delta\epsilon^{-1}\ln(3\delta\cdot\rho))^3$ and

$$s = 2\epsilon^{-1}(\delta \ln s' + \ln(3\delta\cdot\rho)) \ .$$

With these choices we have

$$
\begin{aligned}
s &= 2\epsilon^{-1}(\delta \ln s' + \ln(3\delta\cdot\rho)) \\
&\leq 2\epsilon^{-1}\cdot\delta\cdot\ln s'\cdot\ln(3\delta\cdot\rho) \\
&\leq 2(\epsilon^{-1}\delta\ln(3\delta\cdot\rho))^2 \\
&\leq (\delta\epsilon^{-1}\ln(3\delta\cdot\rho))^3 = s'
\end{aligned}
$$

We further conclude

$$
\begin{aligned}
\rho\cdot\delta\cdot s^{\delta}\cdot e^{-\epsilon(s-s/2)} &\leq \rho\cdot\delta\cdot s^{\delta}\cdot(s')^{-\delta}\cdot(3\delta\cdot\rho)^{-1} \\
&\leq 1/3
\end{aligned}
$$

Hence, with probability at least $\frac{2}{3}$ all $b = (W,\ell) \in \mathcal{B}$ with $W \subseteq S$ are violated by S, which completes the proof of the first part of the theorem. If $\mathcal{P}$ is monotone and feasible then every set $X \subseteq \mathcal{C}$ of size at least $\dim(\mathcal{P})$ contains a self-feasible basis. Therefore, S must contain a self-feasible basis because $s \geq \dim(\mathcal{P})$. Hence the tester accepts the input. It remains to show that

$$s = \Theta(\epsilon^{-1}\cdot(\delta\cdot\ln(\delta/\epsilon) + \ln\rho))$$

We have

$$
\begin{aligned}
s &= 2\epsilon^{-1}(\delta\ln s' + \ln(3\delta\cdot\rho)) \\
&= \Theta(\epsilon^{-1}(\delta(\ln(\delta\epsilon^{-1}) + \ln\ln(\delta\rho)) + \ln(\delta\rho)) \\
&= \Theta(\epsilon^{-1}(\delta(\ln(\delta\epsilon^{-1}) + \ln\ln\rho) + \ln\rho)) \\
&= \Theta(\epsilon^{-1}\cdot(\delta\cdot\ln(\delta/\epsilon) + \ln\rho))
\end{aligned}
$$

by the observation that for $\delta \geq \sqrt{\ln\rho}$ we have $\delta\ln\ln\rho = \mathcal{O}(\delta\ln(\delta/\epsilon))$ and for $\delta < \sqrt{\ln\rho}$ we have $\delta\ln\ln\rho = o(\ln\rho)$. □

Corollary 4.1.6 *Algorithm* ACP-TESTER *is a property tester for monotone abstract combinatorial programs.*

Proof : Follows immediately from Theorem 5. □

4.2 Property Testing vs. Testing Abstract Combinatorial Programs

Our motivation to introduce abstract combinatorial programs was to study its relation to property testing problems. We now prove a theorem that shows how we can use abstract combinatorial programs to prove for certain properties that there is an efficient property tester. Roughly speaking, a property can be tested with small query complexity, if for every problem instance there is an equivalent (in the sense of property testing) abstract combinatorial program of small dimension and width.

We now present a first (simple) variant of the main theorem of this chapter. Then we give some examples and discuss in detail how our theorem can be used to prove the existence of a property tester with small query complexity. In most examples the obtained algorithm has also a small running time.

Our approach of using the framework of abstract combinatorial programs to study property testers of functions $f \in \mathcal{F}$ is to reduce testing of f to testing certain ACPs. In this section we consider only ACPs whose ground set $\mathcal{C}$ is the domain $\mathcal{D}$ of the function f. Later in Section 4.5 we show how to deal with other cases. In order to show that a property Π can be tested with low query complexity we construct for every $f \in \mathcal{F}$ an ACP $\mathcal{P}_f$. $\mathcal{P}_f$ must satisfy the following constraints: If f is ϵ-far from Π then $\mathcal{P}_f$ must be ϵ-far from feasible. The second constraint requires that if the function values of f on a set X can be extended to a function in Π (and if X has a certain size) then X contains a feasible basis. We now want to prove a special case of the main theorem of this chapter. We only consider ACPs whose ground set $\mathcal{C}$ is the domain $\mathcal{D}$ of the tested function f.

Theorem 6 *Let $\mathcal{F}$ be a set of functions from a finite set $\mathcal{D}$ to a set $\mathcal{R}$ and let Π be a property of $\mathcal{F}$. Let $0 < \epsilon < 1$ and let $\delta, \rho \in \mathbb{N}$. If for every $f \in \mathcal{F}$ there exists an ACP $\mathcal{P}_f$ with $\dim(\mathcal{P}_f) \leq \delta$ and width$(\mathcal{P}_f) \leq \rho$ such that the following two conditions are satisfied:*

(Distance Preserving) *if f is ϵ-far from Π then $\mathcal{P}_f$ is ϵ-far from feasible and*

(Feasibility Preserving) *for every $X \subseteq \mathcal{C}$ with $|X| \geq \delta$: If there exists $g \in \Pi$ with $f_{|X} = g_{|X}$, then X contains a self-feasible basis,*

then there exists $s = \Theta(\epsilon^{-1} \cdot (\delta \cdot \ln(\delta/\epsilon) + \ln \rho)))$ such that the following algorithm is a property tester for property Π:

```
TESTER(f, ε)
    Sample a set S of s elements in D uniformly at random
    if f_|S = g_|S for some g ∈ Π then accept f
    else reject f
```

.

Proof : In order to show that TESTER(f, ϵ) is a property tester for Π, we have to prove that every function having property Π is accepted by the tester, and every function that is

ϵ-far from having property Π is rejected with probability at least $\frac{2}{3}$. If $f \in \Pi$ then for every $X \subseteq \mathcal{C}$ we have $f_{|X} = g_{|X}$ with $g = f \in \Pi$. This immediately implies that every $f \in \Pi$ is accepted by TESTER(f, ϵ). Therefore, it remains to prove that if f is ϵ-far from Π, then the algorithm rejects the input with probability greater than or equal to $\frac{2}{3}$. We prove this by relating ACP-TESTER$(\mathcal{P}, \epsilon)$ to TESTER(f, ϵ) and by applying Theorem 5.

By the Distance Preserving property, if f is ϵ-far from Π then $\mathcal{P}_f$ is ϵ-far from feasible. Furthermore, by Theorem 5, if $\mathcal{P}_f$ is ϵ-far from feasible then ACP-TESTER$(\mathcal{P}_f, \epsilon)$ rejects $\mathcal{P}_f$ with probability greater than or equal to $\frac{2}{3}$. $\mathcal{P}_f$ is rejected by ACP-TESTER$(\mathcal{P}_f, \epsilon)$ only if the chosen sample set S contains no self-feasible basis. But now the Feasibility Preserving property implies that if there is $g \in \Pi$ that agrees with f on the sample set then every set $X \subseteq \mathcal{C}$ with $|X| \geq \delta \geq \dim(\mathcal{P}_f)$ contains a self-feasible basis. By the fact that $|S| \geq \delta$ we can conclude that S contains a self-feasible basis, if there exists a $g \in \Pi$ that agrees with f on the sample set S. Therefore, we can conclude that if f is ϵ-far from Π then with probability at least $\frac{2}{3}$ there is no such $g \in \Pi$ with $f_{|S} = g_{|S}$. Hence, f is rejected by TESTER(f, ϵ) with probability at least $\frac{2}{3}$. This implies that TESTER(f, ϵ) is a property tester for Π. □

4.3 Clustering Problems

So far we have introduced a framework that can be used to prove that certain properties can be tested efficiently. Although in principle we can model many different property testing problems in terms of our framework, it is not clear that we can use our framework to prove that these properties have efficient property testers. For this reason we now consider problems from different fields and show that our framework can be used to design and analyze efficient property testers.

The first class of problems we consider are clustering problems. Clustering deals with the problem to partition a set of items into different groups called *clusters* such that a given optimization function is minimized. If we consider the corresponding decision problem we want to know if a clustering with a given optimization value exists.

We consider two clustering problems of point sets in the $\mathbb{R}^d$. The first problem is called *radius* k*-clustering*. Here the goal is to partition a point set in the $\mathbb{R}^d$ into k different clusters such that the maximum cost of the k clusters is minimized. The cost of a cluster is given by the *radius* of the smallest ball enclosing the cluster. The second problem is called *diameter* k*-clustering*. Again the goal is to minimize the maximum cost among a set of k clusters but this time the cost of a cluster is given by the maximum distance between any two points within a cluster.

Both problems have been analyzed in the context of property testing and it is known that there are efficient property testers [4]. We merely want to show that these results can also be achieved using our framework. The proofs we present are in the spirit of the proofs from [4]. Yet we think that our proofs might give a clearer view of the important combinatorial aspects that make these problems testable. In case of the diameter clustering problem we also slightly improve the query complexity.

4.3.1 Radius clustering

The decision version of the *radius* k-*clustering* problem in the Euclidean space $\mathbb{R}^d$ [4, 41] [57, p. 325] (sometimes also called the Euclidean k-center problem) is to verify whether a given set P of n points in $\mathbb{R}^d$ can be partitioned into k sets such that the points in each set are contained in a unit ball. If such a partition exists, we say that P is k-*clusterable*. We assume that P is in general position and, as usual, that the point set P is represented as a function $f : [n] \to \mathbb{R}^d$. Let $\mathcal{F}$ denote the set of functions representing point sets of size n and let $\Pi \subseteq \mathcal{F}$ denote the set of functions representing point sets that are k-clusterable. The distance between two point sets is given by the standard distance measure between functions (see Definition 2.1.3) which is consistent with the following definition:

Definition 4.3.1 *A set P of n points in $\mathbb{R}^d$ is* ϵ-far from being k-clusterable *if more than $\epsilon\, n$ points must be deleted from P to obtain a point set that is k-clusterable.*

Now, in order to use our framework from Theorem 6 we have to describe for every input point set P in $\mathbb{R}^d$ an ACP $\mathcal{P} = (\mathcal{C}, \mathcal{B}, \varpi)$ with $\mathcal{C} = [n]$ that satisfies the two conditions of the theorem. Before we start thinking about the construction of $\mathcal{P}$ we observe that the radius k-clustering property is a combinatorial property. This means that the 'number' of a point in the representation is irrelevant for the property. Thus we can identify the ground set $\mathcal{C}$ of the ACP with the point set P. Hence a basis of such an ACP consists of a small set of points from P (formally, of their corresponding indices) and some additional information (formally encoded as an integer number).

Now we are ready to talk about how to construct the ACPs. Usually, the hard part is to find the right set of bases. Typically, a basis is used to represent implicitly a "possible solution" to the problem. Usually, it should be the case that the set of bases corresponds to the set of possible solutions of the problem. Additionally, it is required that this implicit representation can be done in the form of some atom items from the ground set of the ACP (in our case these items correspond to points from P) and some "additional information".

For simplicity, let us first consider the radius 1-clustering problem. If the point set P is 1-clusterable then by definition it is contained in some unit ball. Vice versa, we can describe every 'possible solution' of the problem implicitly by the position of such a unit ball. In a similar way we can consider every ball with radius at most 1 as a possible solution. Our goal is to describe a subset of these balls implicitly in terms of atom items (points). If P is 1-clusterable then this subset must contain a ball that contains every point in P.

In the following we use the fact that every finite point set is contained in a unique (closed) ball of smallest radius (see, e.g., [78]). We denote this ball by $\mathit{sball}(P)$. With every subset $W \subseteq P$ we associate its smallest enclosing ball. If the radius of $\mathit{sball}(W)$ is at most 1 then this ball can be interpreted as a 'possible solution' of the problem and W (formally, the pair $(W, 1)$) as a basis. So we could say that for each $W \subseteq P$ the pair $(W, 1)$ is a basis, if and only if the radius of $\mathit{sball}(W)$ is at most 1. But to get a query complexity independent of n from our framework we need to reduce the number of atom items involved in a basis. To do this we use the fact that there is always a subset $W \subseteq P$ of cardinality at most $d + 1$ such that $\mathit{sball}(W) = \mathit{sball}(P)$ [78].

We say that $(W, 1)$ is a basis, if the following two conditions are satisfied:

- $|W| \leq d + 1$,
- the radius of $sball(W)$ is at most 1.

We can now define a natural violation function in the following form: A basis $(W, 1)$ is violated by all points that are not contained in $sball(W)$. Using this definition we know (a) that $\mathcal{P}$ has a feasible basis if P is 1-clusterable and (b) that every basis in $\mathcal{P}$ is violated by more than ϵn points, if P is ϵ-far from 1-clusterable.

We can easily extend this definition of a basis to k clusters: A basis for the radius k-clustering problem consists of k bases for single clusters. Formally, a basis consists of at most $k(d + 1)$ points from P and an integer number that encodes a partition of the $k(d + 1)$ points into k sets (of size at most $d + 1$). The violation function is defined in the straightforward way: A point violates a basis, if it is not contained in any of the smallest enclosing balls defined by the bases. We have now done the "tricky part" of the proof - the design of the bases - what remains is straightforward verification of the requirements of our framework.

Bases for radius clustering: We define the bases to specify all possible representations of feasible k-clusterings. Formally, we define the set of bases $\mathcal{B} = \{(W, \ell) : W \subseteq [n], |W| \leq (d + 1)k, 1 \leq \ell \leq k^{(d+1)k}\}$, where the pair $(W, \ell) \in \mathcal{B}$ should be interpreted as follows:

- W is the set of points defining k smallest enclosing balls (clusters), and
- ℓ is represented as a vector $\langle \nu_1, \ldots, \nu_{(d+1)k} \rangle$ of length $(d + 1)k$ such that for $1 \leq i \leq |W|$, the ith point in W defines the smallest enclosing ball containing cluster number $\nu_i \in [k]$.

We say a basis $b \in \mathcal{B}$ is *valid* if every set W' of points defining one of the k smallest enclosing balls in b satisfies that the radius of $sball(W')$ is at most 1.

It is easy to see that with such a definition of the bases, the abstract combinatorial program designed for the radius clustering problem has $\dim(\mathcal{P}) = (d + 1)k =: \delta$ and $width(\mathcal{P}) = k^{(d+1)k} =: \rho$.

Violation Function for Radius Clustering. We say $p \in \mathcal{C} = [n]$ *violates a basis* $b \in \mathcal{B}$ if b is not valid or the point p (located at $f(p)$) is not contained in any of the k balls defined by b. Furthermore, all non-valid bases are violated by all ground set elements.

Now, once we have defined formally the abstract combinatorial program $\mathcal{P}$ for every instance of the radius clustering problem, we have to verify the prerequisites of Theorem 6: The Distance Preserving and the Feasibility Preserving properties.

Distance Preserving Property. If P is ϵ-far from being k-clusterable, then for every set of k balls in $\mathbb{R}^d$ of radius at most one there is always a set of more than ϵn points in P that are not contained in any of the balls. By definition every basis corresponds to such a set of smallest enclosing balls and the violation function is defined according to these balls. Thus every basis $b \in \mathcal{B}$ must be violated by more than ϵn elements from $\mathcal{C}$. This implies the Distance Preserving property.

Feasibility Preserving Property. Every set S, $|S| \geq \delta$, that is k-clusterable is contained in k unit balls. Thus we can partition S into k clusters $S_1, \ldots, S_k$ each of which is contained in a unit ball. Further there exists sets $W_i \subseteq S_i$ with $\mathit{sball}(W_i) = \mathit{sball}(S_i)$ and $|W_i| \leq d+1$. The sets W_i define a certain basis in $\mathcal{B}$ which is covered by S and is feasible for S. This implies the Feasibility Preserving property. Therefore, we can apply Theorem 6 to obtain a property tester for the radius clustering problem with a query complexity of $\widetilde{\mathcal{O}}(d\,k/\epsilon)$.

Implementation. We also observe that we can implement the second statement of the algorithm TESTER(f, ϵ) efficiently in the following way: We compute whether the sample set S is k-clusterable. If it is, we accept the input. If it is not we reject. The correctness follows immediately from the fact that if a point set S is k-clusterable then there exists a superset of S with size n that is also k-clusterable. We summarize our discussion in this section with the following theorem.

Theorem 7 *There is a property tester for the radius clustering problem with query complexity of*

$$\mathcal{O}(d\,k\,\epsilon^{-1}\,\ln(d\,k/\epsilon)) = \widetilde{\mathcal{O}}(d\,k/\epsilon)\ .$$

□

Finally, let us discuss what we learned from this example with respect to our framework. We have seen that the bases of the constructed ACP correspond to possible solutions of the problem. If we generalize this observation we can say, roughly speaking, that properties are testable if every possible (global) solution to the problem can be encoded using a few "atom items" (of the problem instance) and some additional information. Unfortunately, things are not always that simple. In this first example we have a one-to-one correspondence between the clustering problem and the ACP, i.e., a sample $S \subseteq \mathcal{C}$ has a self-feasible basis if and only if the corresponding point set is k-clusterable. This is not always the case as we see in the next section.

4.3.2 Diameter clustering

The decision version of the diameter k-clustering problem ([4], [9, Problem ND54], [48, Problem MS9], [57, p.326]) is defined as follows: Given a point set P in $\mathbb{R}^d$ and a positive integer k, can P be partitioned into k disjoint sets (clusters) $C_1, \ldots, C_k$ such that for every i, $1 \leq i \leq k$, and every $x, y \in C_i$ it holds that $\mathtt{dist}(x, y) \leq 1$. If such a partition exists, we say that P is k-clusterable. As always, we assume in the property testing setting that the

point set is represented by a function $f : [n] \to \mathbb{R}^d$. In this case we *do not use* the standard distance measure. It has been shown in [4] that under the standard distance measure every property tester must have a query complexity of $\Omega(\sqrt{n})$. Instead we use the bicriteria distance measure proposed in [4] which is defined in the following way:

Definition 4.3.2 [4] *Let P be a point set in $\mathbb{R}^d$ and k be a positive integer. We say P is (ϵ, β)*-far from being k-clusterable *if for every partition of P into sets $C_0, C_1, \ldots, C_k$ satisfying $\mathtt{dist}(x, y) \leq 1 + \beta$ for all $1 \leq i \leq k$ and $x, y \in C_i$, it holds that $|C_0| > \epsilon \cdot |P|$.*

It is known that under this distance measure there is a property tester with query complexity $\widetilde{\mathcal{O}}\left(\frac{k^2}{\epsilon} \cdot \left(\frac{2}{\beta}\right)^{2d}\right)$ for the diameter k-clustering problem [4]. Again our goal in this section is to show that we can prove this result using our framework and improve the query complexity of $\widetilde{\mathcal{O}}\left(\frac{k}{\epsilon} \cdot \left(\frac{2}{\beta}\right)^{d}\right)$. Our proof uses combinatorial arguments that appear implicitly in the proof presented in [4]. Our goal is to design an efficient property tester that for given k, ϵ and $\beta > 0$ (i) accepts every point set that is k-clusterable and (ii) rejects with probability at least $\frac{2}{3}$ every input that is (ϵ, β)-far from being k-clusterable. Again we denote by $\mathcal{F}$ all functions representing point sets of size n and by $\Pi \subseteq \mathcal{F}$ all functions representing point sets that are k-clusterable.

As in the case of the radius clustering problem we begin our discussion with the construction of bases for a single cluster. We will see that we can generalize this definition to the case of k-clusters in a similar way as in the radius clustering problem. We use the following notation: A *cluster* C is a non-empty set of points in $\mathbb{R}^d$ with $\mathtt{dist}(x, y) \leq 1$ for every $x, y \in C$.

Definition 4.3.3 *Let C be a cluster. The* kernel $\mathtt{kern}(C)$ of C *is defined as the intersection of unit balls with centers at the points in C.*

We use the following simple properties of a kernel:

Claim 4.3.4 *Let C be a cluster. Then we have:*

1. $C \subseteq \mathtt{kern}(C)$.
2. *There exists a unit ball containing all the points in C.*
3. *If $p \in \mathtt{kern}(C)$ then $\mathtt{dist}(x, y) \leq 1$ for every $x, y \in C \cup \{p\}$.*

Proof : (1) Since $\mathtt{dist}(x, y) \leq 1$ for every $x, y \in C$, each point $x \in C$ is contained in every unit ball with the center at any other point $y \in C$, and hence, x is contained in $\mathtt{kern}(C)$. (2) Since $C \neq \emptyset$, from the previous property we get $\mathtt{kern}(C) \neq \emptyset$. Let us pick any point $x \in \mathtt{kern}(C)$. Since x is contained in all unit balls with centers at the points in C, it is at the distance at most 1 from every point in C. Therefore all points in C are contained in the unit ball with the center at x. (3) If $p \in \mathtt{kern}(C)$, then p is contained in all unit balls with centers at the points in C and thus its distance to every point in X is at most 1. □

Now, in order to use our framework from Theorem 6 we have to describe for every input set P of n points in $\mathbb{R}^d$ an ACP $\mathcal{P} = (\mathcal{C}, \mathcal{B}, \varpi)$ with $\mathcal{C} = [n]$ that satisfies the preconditions of the theorem. Following the arguments from the radius clustering case we can again identify the ground set $\mathcal{C}$ of the ACP with the input point set P. Thus we can again construct our bases using points from P as atom items. We first consider the case $k = 1$:

We start by making an observation about the kernel of a cluster C: If we add a point p to C then C remains a cluster (i.e., the pairwise distance between points from C is at most 1) if and only if p is in the kernel of C. Thus a good basis would contain exactly those elements from C that define the boundary of $kern(C)$. Unfortunately, it might be the case that almost every element of C defines the boundary of C. So this approach is doomed to failure. But we can do the following: We can find a small set of points W in C that approximates the kernel of C. We choose a basis for a cluster C to be every maximal subset $W \subseteq C$ with the property that all points in W have a mutual pairwise distance of more than β. This way, we ensure that (a) the kernel is well approximated and (b) that the number of points defining the kernel of a cluster is small. We now prove that every basis for a single cluster is defined by no more than $(1 + \frac{2}{\beta})^d$ points.

Lemma 4.3.5 *Let C be a cluster and let W be a subset of C such that for every $p, q \in W$ $dist(p, q) > \beta > 0$. Then $|W| \leq (1 + 2/\beta)^d$.*

Proof : Let C and W be as stated in the lemma. If we draw balls of radius $\beta/2$ centered at every point in W, then all these balls are pairwise disjoint. By Property 2 in Claim 4.3.4, all points in W are contained in some unit ball because $W \subseteq C$ and C is a cluster. Therefore, if we draw balls of radius $\beta/2$ centered at every point in W, then all these balls are contained in a ball of radius $1 + \beta/2$. Since all these small balls are disjoint, we have the following upper bound for the size of W:

$$|W| \cdot Vol(\text{ball of radius } \beta/2) \leq Vol(\text{ball of radius } 1 + \beta/2) \ ,$$

where $Vol()$ denotes the volume of the object. Since the volume of a d-dimensional ball of radius r is equal to[1] $\frac{r^d \cdot \pi^{d/2}}{\Gamma(1+d/2)}$, we obtain an upper bound for the size of W of $(1 + (2/\beta))^d$. □

To formalize our definition of bases we need:

Definition 4.3.6 *Let P be a point set in $\mathbb{R}^d$, and β a positive real. Let C be a cluster. We say a point $p \in P$ is* β-covered *by C if $p \in kern(C)$ and there is $q \in C$ such that $dist(p, q) \leq \beta$.*

Similar to the radius clustering problem we want to represent every possible solution to the diameter 1-clustering problem by a basis. As already mentioned we can do this

[1] Here, $\Gamma()$ is Euler's Gamma (factorial) function, that is formally defined as $\Gamma(x) = \int_0^\infty t^{x-1}\, e^{-t}\, dt$ for all positive x. It is well known that $\Gamma(x+1) = x\,\Gamma(x)$ and that for integer $x \geq 0$ we have $\Gamma(x+1) = x!$ and $\Gamma(x + \frac{1}{2})! = \sqrt{\pi} \cdot ((2x)!)/(x! \cdot 4^x)$.

only approximately. Therefore we say that a pair $b = (W, \ell)$ is a basis for the diameter 1-clustering problem if W is a subset of P of size at most $(1 + \frac{2}{\beta})^d$ and $\ell = 1$. We say that a basis is *valid*, if for all $p, q \in W$ we have $\beta \leq dist(p, q) \leq 1$. It remains to define the violation function. If a basis is not valid it is violated by every point in P. If a basis is valid, we have two different types of violation: First of all a point p violates a basis $b = (W, 1)$, if $p \notin kern(W)$. This is the straightforward type of violation. If a point is not in the kernel of W then it cannot belong to the same cluster. The second type of violation is different. A point $p \in kern(W)$ violates b, if p is not β-covered by W. Here a point violates a basis if it is consistent with the current basis of the cluster, but it changes the kernel significantly. But if the kernel is changed significantly then the basis b is no longer "a good implicit description of the solution of the clustering problem". In this case, we can obtain a new basis $b' = (W \cup \{p\}, 1)$ from b that approximates the new kernel.

Bases for diameter clustering. We can extend our definition of bases for the diameter 1-clustering problem to arbitrary k in the following way: A basis for the diameter k-clustering problem is an encoding of k sets $W_1, \ldots, W_k \subseteq P$ each of size at most $(1+\frac{2}{\beta})^d$. A basis is *valid* if for every $p, q \in W_i$, $1 \leq i \leq k$, it holds that $\beta \leq dist(p, q) \leq 1$. Formally, a basis is a set $W = \bigcup_i W_i$ with an integer encoding the partition of W in the sets W_i.

Lemma 4.3.7 *Any ACP $\mathcal{P}$ for the diameter k-clustering problem has dimension at most $k \cdot (1 + 2/\beta)^d$ and width at most $k^{k\cdot(1+2/\beta)^d}$.*

Proof : The dimension of the ACP follows immediately from the definition of the bases. The width follows from the fact that every point of a basis can belong to one of k sets, that is, we have (at most) k choices for each point of the basis. □

Violation function for diameter clustering. A basis b that is an encoding of the sets $W_1, \ldots, W_k$ is violated by a point $p \in P$, if b is not valid or if p violates every W_i (seen as a basis for the 1-clustering problem), $1 \leq i \leq k$.

Feasibility Preserving property. In order to show the Feasibility preserving property we have to show that every k-clusterable set $S \subseteq P$ of size at least $\delta := k\cdot(1+2/\beta)^d \geq \dim(\mathcal{P})$ has a self-feasible basis. If S is k-clusterable then there exists a partition of S into k clusters $C_1, \ldots, C_k$. Since for every $W_i \subseteq C_i$ it holds that $kern(C_i) \subseteq kern(W_i)$ we know that there exists sets W_i with the property that for each $p, q \in W_i$ we have $\beta \leq dist(p, q) \leq 1$ and for each $p \in C_i$ and $q \in W_i$ we have $dist(p, q) \leq \beta$. By Lemma 4.3.5 and the size of the bases the Feasibility Preserving property follows.

Distance Preserving property. We prove the Distance Preserving property by contradiction. Let us assume P is (ϵ, β)-far from being k-clusterable and suppose there is a basis b encoding the sets $W_1, \ldots, W_k$ that is violated by less than ϵn points. We delete all points in P that violate b and let P^* be the remaining point set. Since all the points in

P^* are β-covered by some W_i, for each point $p \in P^*$ there is a W_i with $p \in kern(W_i)$ and for which there exists $q_p \in W_i$ with $dist(p, q_p) \leq \beta$. We assign each such a point p to the cluster corresponding to W_i. Observe that all points in the cluster are contained in $kern(W_i)$. Furthermore, for every point $r \in kern(W_i)$ the distance between p and r is not larger than the distance from p to q_p plus the distance from q_p to r. Hence, we can conclude that the distance between two points in the cluster (both of which must be contained in $kern(W_i)$) is at most $1 + \beta$. This implies that P^* can be partitioned into k clusters of diameter at most $1 + \beta$ each, which is a contradiction.

Implementation. Similar to the radius clustering problem we can implement the second statement of the algorithm TESTER(f, ϵ) by checking if the sample set S is k-clusterable. This follows from the fact that every set k-clusterable set S of size at most n can be extended to a set P of size n that is clusterable. On the other hand, every set S that is not k-clusterable cannot be extended to a set P that is k-clusterable. Now, by our discussion above, we can apply Theorem 6 to obtain the following result.

Theorem 8 *There is a property tester for the diameter* k*-clustering problem with a query complexity of*

$$\widetilde{\mathcal{O}}(k \cdot \epsilon^{-1} \cdot (1 + (2/\beta))^d)$$

4.4 Reversal Distance

In the previous section we have seen how to apply our framework to certain clustering problems. We have seen that a clustering problem is efficiently testable, if we have an implicit characterization of the cluster consisting of a small number of input objects and some additional information. We now want to consider a different problem which is called the *reversal distance problem*. Determining the reversal distance between two permutations is a fundamental problem in computational biology. It has been introduced in the pioneering works by Sankoff and later on many researchers have investigated in this problem (see, e.g., the survey in [71, Chapter 10]). The problem to compute the reversal distance between two permutations can be reduced to the problem of computing the distance between one permutation and the identity permutation (details below). Therefore, the problem is also called '*sorting* by reversals'.

In sorting by reversals one is asked to compute the shortest sequence of (interval) *reversals* that transforms a given permutation π into the identity permutation. The number of reversals that are necessary is called the *reversal distance* between π and the identity permutation. Because of its applications in computational biology, sorting by reversals has been widely studied in the recent years (see, e.g., [10, 17, 22, 59, 71, 72]). It is known that sorting by reversals is $\mathcal{NP}$-hard [22], its optimization version is MAX-SNP-hard [18], and that there exits a polynomial-time 1.375-approximation algorithm [17] (see also [10, 59]).

We now introduce the reversal distance problem, formally. Let $\mathcal{S}_n$ denote the set of all permutations of $[n]$.

Definition 4.4.1 *A reversal $\rho\langle i,j\rangle$ of an interval $[i,j]$, $1 \leq i \leq j \leq n$, is the permutation*

$$\begin{pmatrix} 1 & 2 & \dots & i-1 & i & i+1 & \dots & j-1 & j & j+1 & \dots & n \\ 1 & 2 & \dots & i-1 & j & j-1 & \dots & i+1 & i & j+1 & \dots & n \end{pmatrix}.$$

That is, for a permutation $\pi = (\pi_1, \dots, \pi_n) \in \mathcal{S}_n$, $\rho\langle i,j\rangle$ has the effect of reversing the order of $(\pi_i, \pi_{i+1}, \dots, \pi_j)$ and transforming π into

$$\pi \cdot \rho\langle i,j\rangle = (\pi_1, \dots, \pi_{i-1}, \pi_j, \pi_{j-1}, \dots, \pi_i, \pi_{j+1}, \dots, \pi_n)$$

.

The reversal distance between two permutations π and σ is the minimum number of reversals that is necessary to transform π into σ.

Definition 4.4.2 *Given a pair of permutations $\pi = (\pi_1, \dots, \pi_n), \sigma = (\sigma_1, \dots, \sigma_n) \in \mathcal{S}_n$, the* reversal distance $d_{rev}(\pi, \sigma)$ *between π and σ is the minimum number of reversals needed to transform π into σ (that is, the minimum number k such that there exists a sequence of reversals $\rho_1, \rho_2, \dots, \rho_k$ with $\pi \cdot \rho_1 \cdot \rho_2 \cdots \rho_k = \sigma$).*

Equivalently, we can compute the number of reversals necessary to transform $\sigma^{-1}\pi$ into the identity permutation id. Therefore, we are interested in the reversal distance between a given permutation π and the identity permutation. We want to consider the decision version of the reversal distance problem.

Definition 4.4.3 *The* reversal distance *problem is to decide for a given permutation $\pi \in \mathcal{S}_n$ and an integer k if the reversal distance $d_{rev}(\pi, \mathrm{id})$ between π and the identity permutation id is at most k.*

We now want to formulate the problem as a property testing problem that fits into our framework. The set of functions $\mathcal{F}$ we want to consider is the set of permutations of $[n]$, that is, $\mathcal{F} = \mathcal{S}_n$. We are interested in all permutations that have a reversal distance of at most k to the identity permutation. We can write this as a property Π as follows:

$$\Pi = \{\pi \in \mathcal{S}_n : d_{rev}(\pi, \mathrm{id}) \leq k\}.$$

Now that we have defined the property we need a distance measure between permutations. Here we can use the standard distance measure from Definition 2.1.3. Equivalently, we can use the following definition:

Definition 4.4.4 *A permutation $\pi \in \mathcal{S}_n$ is ϵ-far from having reversal distance smaller than or equal to k if for every sequence of k reversals $\rho_1, \rho_2, \dots, \rho_k$ the permutation $\pi \cdot \rho_1 \cdot \rho_2 \cdots \rho_k$ disagrees with the identity permutation on more than $\epsilon \cdot n$ places, that is, if $\pi \cdot \rho_1 \cdot \rho_2 \cdots \rho_k = (\sigma_1, \dots, \sigma_n)$ then $|\{i \in \{1, 2, \dots, n\} : \sigma_i \neq i\}| > \epsilon \cdot n$.*

In contrast to the clustering problems the reversal distance property is obviously not combinatorial. We conclude that we have to take the domain of the function into account. For a permutation $\pi = (\pi_1, \cdots, \pi_n)$ we denote atom items by π_i. This notion covers the fact that the value of domain element i is $\pi_i = \pi(i)$. Hence it captures also the domain of a value of f.

Let us notice that we can encode an interval $[i, j]$ by the two domain elements π_i and π_j (using the fact that $\pi^{-1}(\pi_i) = i$ and $\pi^{-1}(\pi_j) = j$). If we apply a reversal ρ to π then π_i and π_j *induce* the interval $\left[((\pi \cdot \rho)^{-1})(\pi_i), ((\pi \cdot \rho)^{-1})(\pi_j)\right]$ (for this reason we want to work with π_i rather than with i). We denote the interval induced by two elements π_i and π_j by $[\pi_i, \pi_j]$.

We say a reversal $\rho\langle r, s\rangle$ *splits* an interval $[\pi_i, \pi_j]$, if $i < r \leq j$ or $i \leq s < j$ (or both).

Definition 4.4.5 *Let $\pi = (\pi_1, \ldots, \pi_n)$ be a permutation and let $[\pi_i, \pi_j]$ denote an interval. We say that a reversal $\rho\langle k, \ell\rangle$* splits *an interval $[\pi_i, \pi_j]$, if $i < k \leq j$ or if $i \leq \ell < j$.*

We generalize this notion to k-reversals:

Definition 4.4.6 *Let $\pi = (\pi_1, \ldots, \pi_n)$ be a permutation. A k-reversal $\rho = \rho_1 \cdot \rho_2 \cdots \rho_k$* splits *an interval $[\pi_i, \pi_j]$, if there exists ℓ, $0 \leq \ell < k$, such that $\rho_{\ell+1}$ splits*

$$[(\pi \cdot \rho_1 \cdots \rho_\ell)^{-1}(\pi_i), (\pi \cdot \rho_1 \cdots \rho_\ell)^{-1}(\pi_j)] \ .$$

If ρ does not split $[\pi_i, \pi_j]$ then we say ρ is safe *for $[\pi_i, \pi_j]$.*

Notice that if $\rho_1, \ldots, \rho_k$ is *safe* for $[\pi_i, \pi_j]$ then each of the reversals $\rho_1, \ldots, \rho_k$ either entirely contains $[\pi_i, \pi_j]$ or it does not contain any $\pi_\ell \in [\pi_i, \pi_j]$. Therefore, in this case, after applying $\rho_1, \ldots, \rho_k$ the positions of $\pi_{i+1}, \ldots, \pi_{j-1}$ are determined by the position of π_i and π_j.

Bases for the k-reversal problem: The idea is now to define a basis as a set of $2k+1$ intervals induced by pairs of the atom items of the basis. For each such set we then consider only reversal that are safe for these intervals. We say that a set of $2k+1$ intervals is a basis if a sequence of k reversals $\rho_1, \ldots \rho_k$ exists such that for each atom item π_i of the basis it holds that $(\pi \cdot \rho_1 \cdots \rho_k)^{-1}(\pi_i) = \pi_i$ and if $\rho_1, \ldots \rho_k$ is safe for all intervals induced by the elements involved in the basis.

Definition 4.4.7 (Bases for k-reversal distance) *Let $\pi = (\pi_1, \cdots, \pi_n) \in \mathcal{S}_n$. A set $\mathcal{I}$ of $2k+1$ intervals is a valid basis for the reversal distance problem if there is a sequence $\rho_1, \ldots, \rho_k$ of k reversals such that*

- *$(\pi \cdot \rho_1 \cdots \rho_k)^{-1}(\pi_i) = \pi_i$ and $(\pi \cdot \rho_1 \cdots \rho_k)^{-1}(\pi_j) = \pi_j$ for each interval $[\pi_i, \pi_j] \in \mathcal{I}$, and*
- *no interval $[\pi_i, \pi_j] \in I$ is split by $\rho_1, \ldots, \rho_k$.*

If the set of intervals is a basis b, then we associate with it the k-reversal $\rho_b = \rho_1 \cdots \rho_k$ (in case that there are different sequences that witness the basis property we choose an arbitrary one).

Formally, a basis consists of $4k+2$ atom items and an integer number encoding the $2k+1$ pairs of atom items (an atom item may be paired with itself). The integer number can be seen as a vector of length $2k+1$ having entries with values from $[4k+2] \times [4k+2]$ to specify each of the $2k+1$ pairs.

Lemma 4.4.8 *For each instance of the reversal distance problem the corresponding ACP* $\mathcal{P}$ *has* $\dim(\mathcal{P}) \leq 4k+2$ *and width*$(\mathcal{P}) \leq (4k+2)^{4k+2}$.

Proof : By definition a basis consists of $4k+2$ atom items. Thus we have $\dim(\mathcal{P}) \leq 4k+2$. Each vector of length $2k+1$ with values from $[4k+2] \times [4k+2]$ can be encoded as an integer number between 1 and $(4k+2)^{4k+2}$. Thus we have *width*$(\mathcal{P}) \leq (4k+2)^{4k+2}$. □

Violation function for k-reversal distance: Let b be a basis and let $\rho_b = \rho_1 \cdots \rho_k$ be the k-reversal associated with basis b. We say b is violated by $\pi_i \in \mathcal{C}$, if $(\pi \cdot \rho_b)^{-1}(\pi_i) \neq \pi_i$, that is, π_i is not moved to position π_i when ρ_b is applied to π.

Distance Preserving Property: We have associated a k-reversal ρ_b to each basis. An atom item violates a basis, if it is not at the correct position when ρ_b is applied to π. If a permutation is ϵ-far from having reversal distance smaller than or equal to k then every k-reversal puts more than ϵn elements to the wrong position. Hence every basis is violated by more than $\epsilon\, n$ elements. Therefore, the distance preserving property is satisfied.

Feasibility Preserving Property: The hard part in this case is to prove the Feasibility Preserving property. Let $S \subseteq \mathcal{C}$ be a set of atom items and let $\rho = \rho_1 \cdots \rho_k$ be a k-reversal with $(\pi \cdot \rho)^{-1}(\pi_i) = \pi_i$ for each $\pi_i \in S$. We show that in this case S has a self-feasible basis. First we want to construct a set of $2k+1$ intervals that are safe for $\rho_1, \ldots \rho_k$. We start with the set of $s-1$ intervals induced by S. now we observe that each reversal ρ_i can split at most 2 of these intervals. We conclude that at most $2k$ of these intervals are split by $\rho_1, \ldots \rho_k$. We can merge adjacent intervals not split by $\rho_1, \ldots \rho_k$ and obtain a set of $2k+1$ intervals that are not split by $\rho_1, \ldots \rho_k$. Hence there exists a sequence of k reversals that is safe for each of our intervals. Thus these intervals form a basis b. It remains to prove that this basis is not violated (the k-reversal associated with the basis must not be the k-reversal ρ). By our construction of the intervals each $\pi_i \in S$ is contained in a safe interval. Therefore, its position after applying the reversal is uniquely determined by the positions of the endpoints of the interval. Let $S_I \subseteq S$ denote the set of endpoints of intervals of the basis b. Since b is a basis there is a k-reversal ρ_b with

$$(\pi \cdot \rho_b)^{-1}(\pi_i) = \pi_i = (\pi \cdot \rho)^{-1}(\pi_i) \text{ for each } \pi_i \in S_I$$

Since the endpoints are mapped to the same places when ρ_b and ρ are applied to π we can conclude that each other point in S is also mapped to the same place. Hence no $\pi_i \in S$ violates b and we have shown the Feasibility Preserving property.

Once we have proven the Distance and the Feasibility preserving property, Lemma 4.4.8 and Theorem 6 allow us to conclude with the following theorem.

Theorem 9 *There exists a property tester for the* k*-reversal distance property with query complexity* $\widetilde{\mathcal{O}}(k/\epsilon)$. □

4.5 Property Testing vs. Testing Abstract Combinatorial Programs *(continued)*

In Section 4.2, we described a framework for testing problems via testing abstract combinatorial programs. We only considered ACPs whose ground set is equivalent to the domain of the tested function. Although we showed that this framework can be applied to different problems, it is not always powerful enough. In some cases it is necessary to consider ground sets different from the domain of the tested function. For example, when we consider graph problems we want to identify the ground set with the set of vertices of the graph. If we consider the adjacency matrix model using the approach from Section 4.2 and we represent a graph by a function $f : V \times V \to \{0,1\}$ then the ground set would be a set of entries in the adjacency matrix. But sampling entries of the adjacency matrix results in a disconnected set of edges, if the size of the sample set is $o(\sqrt{n})$ [51]. In order to have a more flexible model we introduce *interpretations*.

Interpretations. Interpretations are functions that map each subset of the ground set of the ACP to a subset of the domain of the tested function. Given a sample set S of ground set items we use the interpretation to determine a set of domain elements $\mathcal{D}_S$. Then we query for the value $f(x)$ for each $x \in \mathcal{D}_S$.

Definition 4.5.1 *An* interpretation *of* $\mathcal{C}$ *in* $\mathcal{D}$ *is a function* $I : 2^{\mathcal{C}} \to 2^{\mathcal{D}}$.

To investigate quantitative properties of the reduction we need the following definition.

Definition 4.5.2 *For a function* $h : \mathbb{N} \to \mathbb{N}$*, we say an interpretation* I *of* $\mathcal{C}$ *in* $\mathcal{D}$ *is* h-bounded *if for every* $X \subseteq \mathcal{C}$ *it holds* $|I(X)| \leq h(|X|)$. *(We write in that case that* I *is* $h(N)$*-bounded, with* N *being the formal input variable.)*

The main idea behind introducing these notions is to allow a more general analysis of algorithm TESTER(f, ϵ) from Section 4.2. Similarly to the proof of Theorem 6, we want to test an input function $f \in \mathcal{F}$ via testing a related ACP $\mathcal{P} = (\mathcal{C}, \mathcal{B}, \varpi)$. Since $\mathcal{P}$ is now allowed to be an ACP with arbitrary ground set $\mathcal{C}$, we use the interpretation I of $\mathcal{C}$ in $\mathcal{D}$ to link the domains of f and $\mathcal{P}$ in the reduction. The notion of h-bounded functions in Definition 4.5.2 is used to describe the size of the random sample in the tester. That is, if the interpretation I is $h(N)$-bounded and if our algorithm samples a set $S \subseteq \mathcal{C}$ then we query for the value of $f(x)$ for every $x \in I(S) \subset \mathcal{D}$ and the restriction $|I(S)| \leq h(s)$ yields an upper bound on the query complexity.

Distance Preserving Property. In Theorem 6 we used the Distance Preserving property that requires that if a function f is ϵ-far from property Π then the ACP is ϵ-far from feasible. In general, however, one can parameterize this property and require the (ϵ, λ)-**Distance Preserving** property: *if f is ϵ-far from property Π then the ACP is λ-far from feasible.*

Now, in the framework defined above, it is easy to see that Theorem 6 can be generalized to the following theorem, which describes our framework in its full generality.

Theorem 10 *Let $\mathcal{F}$ be a set of functions from a finite set $\mathcal{D}$ to a set $\mathcal{R}$, and let Π be a property of $\mathcal{F}$. Let $0 < \epsilon, \lambda < 1$ and let $I : 2^{\mathcal{C}} \to 2^{\mathcal{D}}$ be an h-bounded interpretation of $\mathcal{C}$ in $\mathcal{D}$. If for every $f \in \mathcal{F}$ there exists an ACP $\mathcal{P}_f$ with $\dim(\mathcal{P}_f) \leq \delta$ and width$(\mathcal{P}_f) \leq \rho$ such that:*

((ϵ, λ)-Distance Preserving) *if f is ϵ-far from Π then every basis in $\mathcal{P}_f$ is λ-far from feasible, and*

(Feasibility Preserving) *For every $X \subseteq \mathcal{C}$ with $|X| \geq \delta$: If there exists $g \in \Pi$ with $f_{|I(X)} = g_{|I(X)}$ then X contains a self-feasible basis,*

then there exists $s = \Theta(\lambda^{-1} \cdot (\delta \cdot \ln(\delta/\lambda) + \ln \rho))$ such that the following algorithm is a property tester for Π with query complexity $h(s)$:

TESTER(f, ϵ)
Sample a set S of s elements in $\mathcal{C}$ uniformly at random
if $f_{|I(S)} = g_{I(S)}$ *for some* $g \in \Pi$ **then** *accept* f
else *reject* f

Proof : In order to show that TESTER(f, ϵ) is a property tester for Π, we have to prove that every function having property Π is accepted by the tester, and every function that is ϵ-far from having property Π is rejected with probability at least $\frac{2}{3}$. If $f \in \Pi$ then for every $X \subseteq \mathcal{C}$ we have $f_{|I(X)} = g_{|I(X)}$ with $g = f \in \Pi$. This immediately implies that every $f \in \Pi$ is accepted by TESTER(f, ϵ). Therefore, it remains to prove that if f is ϵ-far from Π, then the algorithm rejects the input with probability greater than or equal to $\frac{2}{3}$. We prove this by relating ACP-TESTER$(\mathcal{P}, \epsilon)$ to TESTER(f, ϵ) and by applying Theorem 5.

By the Distance Preserving property, if f is ϵ-far from Π then $\mathcal{P}_f$ is ϵ-far from feasible. Furthermore, by Theorem 5, if $\mathcal{P}_f$ is ϵ-far from feasible then ACP-TESTER$(\mathcal{P}_f, \epsilon)$ rejects $\mathcal{P}_f$ with probability greater than or equal to $\frac{2}{3}$. $\mathcal{P}_f$ is rejected by ACP-TESTER$(\mathcal{P}_f, \epsilon)$ only if the chosen sample set S contains no self-feasible basis. But now the Feasibility Preserving property implies that if there is $g \in \Pi$ that agrees with f on the interpretation of the sample set then every set $X \subseteq \mathcal{C}$ with $|X| \geq \delta \geq \dim(\mathcal{P}_f)$ contains a self-feasible basis. By the fact that $|S| \geq \delta$ we can conclude that S contains a self-feasible basis, if there exists a $g \in \Pi$ that agrees with f on the sample set S. Therefore, we can conclude that if f is ϵ-far from Π then with probability at least $\frac{2}{3}$ there is no such $g \in \Pi$ with $f_{|I(S)} = g_{|I(S)}$. Hence, f is rejected by TESTER(f, ϵ) with probability at least $\frac{2}{3}$. This implies that TESTER(f, ϵ) is a property tester for Π. □

4.6 Graph Coloring

In this section we apply Theorem 10 to graph coloring. A k-coloring of a graph $G = (V, E)$ is an assignment $\chi : V \to \{1, \ldots, k\}$ of colors to the vertices of the graph. A coloring is *proper* if there is no edge $e = (v, u) \in E$ such that $\chi(v) = \chi(u)$. If G has a proper k-coloring, then G is k-*colorable*. The graph k-coloring problem is to decide whether a given graph is k-colorable. It is a classical problem in algorithmic graph theory. It is known that for $k \geq 3$ the problem of verifying if an input graph is k-colorable is $\mathcal{NP}$-complete (see, e.g., [48, Problem GT4] or [9, Problem GT5]). It is also well known that this problem is very hard to approximate, and so, for example, it is $\mathcal{NP}$-hard to 4-color 3-colorable graphs and it is hard to color k-colorable graphs with approximation within $n^{1-\epsilon}$, and even within $n^{1-\mathcal{O}(1/\sqrt{\log \log n})}$ [38] where n denotes the number of vertices in the graph. The best known approximation bound for arbitrary k is of $\mathcal{O}(n\,(\log \log n)^2/\log^3 n)$ [9, Problem GT5].

We want to consider the graph coloring problem in the adjacency matrix model. That is, the input graph $G = (V, E)$ is given as a function $V \times V \to \{0, 1\}$ representing the adjacency matrix of the graph. Thus we have $f(u, v) = 1$, if and only if $(u, v) \in E$. W.l.o.g. we assume that $V = [n]$. Let Π denote all n-vertex graphs that have a proper k-coloring. We use the standard distance measure between graphs (see Definition 2.1.3) which is equivalent to the following definition:

Definition 4.6.1 *A graph* G *is* ϵ-far from being k-colorable *if in order to transform* G *into a* k*-colorable graph one has to modify more than* $\epsilon\, n^2$ *entries in the adjacency matrix of* G.

It is known that graph coloring in the adjacency model can be tested efficiently [51] and the proof we present uses combinatorial arguments that essentially appear in the proof presented in [51]. The achievement of our framework is again a clear and elegant proof that highlights the combinatorial aspects of the problem and a slight improvement in the query complexity that matches for $k > 2$ the bounds from [6] in the $\widetilde{\mathcal{O}}$-notation.

For the k-coloring problem, we identify the ground set $\mathcal{C}$ with the set of vertices V of the input graph $G = (V, E)$. Since in our framework G can be viewed as given by its adjacency matrix representation, we define the interpretation I to map each set of vertices to the submatrix induced by these vertices. That is, for every $W \subseteq V$, we have $I(W) = W \times W$. Clearly, the interpretation is N^2-bounded.

The bases of the coloring problem are formed by some properly colored sets of vertices. That is, every basis corresponds to a pair (W, χ^*), where $W \subseteq V = \mathcal{C}$ and χ^* is an encoding of a proper k-coloring of W (here, one can think that a k-coloring of W is represented by a vector of length $|W|$ with values in k^W). Notice that with this definition, if for each $(W, \chi^*) \in \mathcal{B}$ we have $|W| \leq \delta$, then so defined abstract combinatorial program $\mathcal{P}$ has $\dim(\mathcal{P}) \leq \delta$ and $\mathit{width}(\mathcal{P}) \leq k^\delta$.

Before we define the bases formally, we need some more definitions:

Definition 4.6.2 *Let* $S \subseteq V$ *be a set of vertices and let* χ *be a proper* k*-coloring of* S. *Let*

$$V_i = \left\{ v \in V : \exists u \in S \text{ with } \chi(u) = i \text{ and } (v, u) \in E \right\}$$

be the set of vertices that cannot be properly colored in color i using any extension of χ to V. We call a vertex $v \in V \setminus S$ heavy *for $\langle S, \chi \rangle$ if there is a proper extension of χ to $S \cup \{v\}$, but every extension increases the number of vertices in certain V_i, $1 \leq i \leq k$, by at least $\epsilon\, n/3$ (that is, (i) there exists $1 \leq j \leq k$, with $v \notin V_j$ and (ii) $\forall_{1 \leq j \leq k}$ if $v \notin V_j$ then $\left|\{w \notin V_j : (v, w) \in E\}\right| \geq \epsilon\, n/3$).*

A vertex $v \in V \setminus S$ that cannot be properly colored by any extension of the coloring χ (that is, $v \in \bigcap_{i=1}^{k} V_i$) is called a conflict *vertex for $\langle S, \chi \rangle$.*

Bases for k-coloring: For the graph coloring problem we define the bases inductively:

- $\{\emptyset, 1\}$ is a basis (where 1 is the encoding of the coloring of the empty set of vertices) and
- if $b = (K, \chi)$ is a basis, v is a *heavy* vertex for b and χ^* is an encoding of the previous coloring χ of K extended by a proper coloring of v, then $(K \cup \{v\}, \chi^*)$ is a basis.

Our next step is to show that the ACP we define this way has small dimension and width:

Lemma 4.6.3 *For every input instance of the graph coloring problem the corresponding abstract combinatorial program $\mathcal{P}$ has* $\dim(\mathcal{P}) \leq 3k/\epsilon$ *and width*$(\mathcal{P}) \leq k^{3k/\epsilon}$.

Proof : Since every k-coloring of a set of r vertices can be encoded using an integer number between 1 and k^r, it is enough to show that $|K| \leq 3k/\epsilon$ for every basis $b = (K, \chi)$.

Let $b = (K, \chi)$ be a basis with $|K| = r$. Since b is a basis, there must exist a sequence of bases (K_0, χ_0), (K_1, χ_1), ..., (K_r, χ_r) with $(K_0, \chi_0) = (\emptyset, 1)$, $(K_r, \chi_r) = (K, \chi)$, and such that for every $1 \leq i \leq r$ we have $|K_i| = i$ and the only vertex in $K_i \setminus K_{i-1}$ is heavy for $\langle K_{i-1}, \chi_{i-1} \rangle$.

Let $V_1^{(i)}, V_2^{(i)}, \ldots, V_k^{(i)}$, $0 \leq i \leq r$, be the sets of vertices such that each vertex $v \in V_j^{(i)}$ cannot be properly colored in color j if we want to extend coloring χ_i to the set $K_i \cup \{v\}$. That is,

$$V_j^{(i)} = \left\{ v \in V : \exists u \in K_i \text{ with } \chi_i(u) = j \text{ and } (v, u) \in E \right\} .$$

It is easy to see that for every i, j we have $V_j^{(i)} \subseteq V_j^{(i+1)}$. Furthermore, by the definition of heavy vertices, we know that for every i, $1 \leq i \leq r$, there is certain j, $1 \leq j \leq k$, such that $|V_j^{(i)}| \geq |V_j^{(i-1)}| + \epsilon\, n/3$. Therefore, since we have $V_j^{(0)} = \emptyset$ for every j, $1 \leq j \leq k$, it must hold that $\sum_{j=1}^{k} |V_j^{(i)}| \geq i\, \epsilon\, n/3$ for every i, $1 \leq i \leq r$. Finally, since $|V_j^{(r)}| \leq n$ for every j, $1 \leq j \leq k$, we conclude that $r \leq 3k/\epsilon$. □

Violation function for k-coloring: *A basis $b = (K, \chi)$ is* violated *by a vertex $v \in V$ if either (i) v is a heavy vertex for $\langle K, \chi \rangle$ or (ii) v is a conflict vertex for $\langle K, \chi \rangle$.*

Once we have described k-coloring in our framework, in order to apply Theorem 10, we must show that the (ϵ, λ)-Distance Preserving and the Feasibility Preserving properties hold. We begin with the proof of the following lemma that implies the (ϵ, λ)-Distance Preserving property with $\lambda = \epsilon/3$.

Lemma 4.6.4 ($(\epsilon, \epsilon/3)$-Distance Preserving property) *Let $G = (V, E)$ be a graph that is ϵ-far from being k-colorable and let $S \subseteq V$ be any set of properly k-colored vertices with a proper coloring χ. Then, V contains more than $\epsilon n/3$ conflict vertices for $\langle S, \chi \rangle$ or V has more than $\epsilon n/3$ heavy vertices for $\langle S, \chi \rangle$.*

Proof : Our proof is by contradiction. Assume G is ϵ-far from having a proper k-coloring and there are less than $\epsilon n/3$ conflict vertices for $\langle S, \chi \rangle$ and less than $\epsilon n/3$ heavy vertices for $\langle S, \chi \rangle$. Then, we show that there is a k-colorable graph G^* that is obtained from G by removing less than ϵn^2 edges. This is a contradiction.

Let X be the set of all conflict vertices for $\langle S, \chi \rangle$, let Y be the set of all heavy vertices for $\langle S, \chi \rangle$, and let Z be the set of remaining uncolored vertices (i.e., $Z = V \setminus (S \cup X \cup Y)$). For every i, $1 \leq i \leq k$, let V_i be the set of vertices in V such that for every $v \in V_i$ the extension of χ by coloring v with color i is not a proper coloring (cf. Definition 4.6.2).

We first construct a graph G' by removing all edges incident to the vertices in $X \cup Y$ and extend the coloring χ of S to a k-coloring of $S \cup X \cup Y$ by coloring the vertices in $X \cup Y$ arbitrarily. Since $|X \cup Y| < 2\epsilon n/3$, less than $2\epsilon n^2/3$ edges are removed from G in this way. Furthermore, since all vertices in $X \cup Y$ are isolated, the obtained coloring χ' is a proper k-coloring of $S \cup X \cup Y$.

Now, we modify G' to extend the coloring χ' to all vertices in Z. For each $v \in Z$, let $\tau(v)$ be a color that satisfies the following two constraints:

- If we extend χ' to $S \cup \{v\}$ and we color v with $\tau(v)$ then the resulting k-coloring is proper, and
- If we extend χ' to $S \cup \{v\}$ and we color v with $\tau(v)$ then the absolute increase in the size of $V_{\tau(v)}$ is minimal (among all possible choices that satisfy the first constraint).

In other words, $v \notin V_{\tau(v)}$ and for every i, $1 \leq i \leq k$, if $v \notin V_i$ then $|\{w \notin V_i : (v, w) \in E\}| \geq |\{w \notin V_{\tau(v)} : (v, w) \in E\}|$.) Since v is not a conflict vertex for $\langle S, \chi \rangle$, such a color $\tau(v)$ always exists (but is possibly not unique). By the fact that v is not a heavy vertex for $\langle S, \chi \rangle$, we know that $|\{w \notin V_{\tau(v)} : (v, w) \in E\}| < \epsilon n/3$. Therefore, if we remove from G' for every $v \in Z$ all edges $(v, w) \in E$ with $w \notin V_{\tau(v)}$, then the resulting graph G^* is obtained from G by removal of less than ϵn^2 edges. It remains to show that the following coloring of G^* is proper: We color each vertex $v \in S \cup X \cup Y$ with color $\chi'(v)$. Each vertex $v \in Z$ is colored with color $\tau(v)$. Now assume that this coloring is not proper. Then there must be an edge (v, u) such that v and u are colored in the same color. Since χ is proper, all vertices in X and Y are isolated, and by the definition of $\tau(v)$ it is immediate that such an edge can only be between vertices in Z. But then we have that $u \notin V_{\tau(v)}$ and this implies that (v, u) has been removed from G'. Contradiction. □

It remains to prove the Feasibility Preserving property:

Lemma 4.6.5 (Feasibility Preserving property) *If for some $S \subseteq V$ every basis covered by S is violated by certain $v \in S$ then the subgraph of G induced by vertices in S cannot be properly k-colored.*

Proof : The proof is by contradiction. Let us suppose there is a proper k-coloring χ of the subgraph of G induced by the vertices in S. For every $U \subseteq S$, let χ_U denote the coloring χ restricted to vertex set U.

Let us observe that the set of bases covered by S is not empty, because it contains the "empty set" basis $(\emptyset, \chi_\emptyset)$. Therefore, there exists a basis (possibly one of many) $b = (U, \chi_U)$ covered by S having the maximum size of set U. Since b is violated by certain $v \in S$, either v is a conflict vertex for $\langle U, \chi_U \rangle$ or v is a heavy vertex for $\langle U, \chi_U \rangle$. Furthermore, since χ was assumed to be a proper k-coloring of S, v cannot be a conflict vertex for $\langle U, \chi_U \rangle$ and therefore it must be a heavy vertex for $\langle U, \chi_U \rangle$. But this implies that $(U \cup \{v\}, \chi_{U \cup \{v\}})$ is a basis that is moreover covered by S. This yields a contradiction, because we assumed that there is no basis (K, χ_K) covered by S having $|K| > |U|$. □

Therefore, we summarize our discussion in this section with the following theorem that follows directly from Theorem 10 and Lemmas 4.6.4 and 4.6.5.

Theorem 11 *There is a property tester for the graph* k*-coloring property with a query complexity of*

$$\mathcal{O}\left((k\,\epsilon^{-2}\ln(k/\epsilon^2))^2\right) = \widetilde{\mathcal{O}}(k^2/\epsilon^4)\ . \qquad \square$$

4.7 Hypergraph Coloring

We now want to extend the analysis from Section 4.6 to obtain an efficient *property tester for hypergraph coloring*. A *hypergraph* is a pair $\mathcal{H} = (V, E)$ with a finite vertex set V and the edge set $E \subseteq 2^V$. A hypergraph $\mathcal{H}$ is ℓ*-uniform* if $|e| = \ell$ for all $e \in E$. (Notice that a 2-uniform hypergraph is a graph.) A k*-coloring* of a hypergraph $\mathcal{H}$ is an assignment $\chi : V \to \{1, \ldots, k\}$ of colors to the vertices of the hypergraph. A coloring is *proper* if no edge in E is *monochromatic*, that is, if for every edge $e \in E$ there are $v, u \in e$ with $\chi(v) \neq \chi(u)$. If $\mathcal{H}$ has a proper k-coloring, then $\mathcal{H}$ is k*-colorable*. The k-coloring problem for hypergraphs is to decide whether a given hypergraph is k-colorable.

Hypergraph coloring is a well studied problem in the literature in discrete mathematics, combinatorics, and computer science. In contrast to graphs, where one can decide in linear time if a graph is 2-colorable (or equivalently, bipartite), deciding if a given hypergraph is 2-colorable is $\mathcal{NP}$-complete even for 3-uniform hypergraphs [64]. In [63], it is shown that unless $\mathcal{NP} \subseteq \mathcal{ZPP}$, for every fixed $\ell \geq 3$, it is impossible to approximate in polynomial time the chromatic number of ℓ-uniform hypergraphs within a factor $n^{1-\varepsilon}$ for every constant $\varepsilon > 0$. See also [55, 60] for further inapproximability results. The property of hypergraph 2-colorability has been also extensively studied in combinatorics (see, e.g., [28, 29, 39, 73]), and for example, the study of this problem led to the discovery of the celebrated Lovász Local Lemma [39].

We want to design a property tester for the k-coloring problem in ℓ-uniform hypergraphs. An ℓ-uniform hypergraph can be represented by a function $f : V^\ell \to \{0, 1\}$ that encodes its adjacency matrix. We use the standard distance measure (Definition 2.1.3) to measure the distance between hypergraphs. In terms of hypergraphs we can express this distance measure as follows:

Definition 4.7.1 *An ℓ-uniform hypergraph $\mathcal{H} = (V, E)$ is ϵ-far from having a proper k-coloring, if we have to remove more than ϵn^ℓ edges from $\mathcal{H}$ to obtain a hypergraph that has a proper k-coloring.*

Now, we discuss how to apply our framework to hypergraph coloring. Similarly to the graph coloring problem, we identify the ground set $\mathcal{C}$ with the set of vertices V of the input hypergraph $\mathcal{H} = (V, E)$. Since in our representation $\mathcal{H}$ can be viewed as given by its adjacency matrix representation, we define the interpretation I to map each set of vertices to the submatrix induced by these vertices. That is, for every $S \subseteq V$, we have $I(S) = S \times S \times \cdots \times S$. Clearly, the interpretation is N^ℓ-bounded. Let $\langle S, \chi \rangle$ be a pair with $S \subseteq V$ and χ a proper k-coloring of vertices in S.

Definition 4.7.2 *We say a vertex v is* i-colorable *with respect to $\langle S, \chi \rangle$ if for every $e \in E$ with $v \in e$, either (i) there exists a vertex $u \in (S \cap e)$ with $\chi(u) \neq i$ or (ii) there exists a vertex $w \in e \setminus (S \cup \{v\})$.*

We want to extend our approach for graph coloring to hypergraphs. Again we use a recursive definition of bases that is based on *heavy* vertices and *conflict* vertices. In order to define heavy vertices we introduce a potential function. A vertex is a heavy vertex if its coloring increases the potential significantly (very much in the spirit of graph coloring; further motivation behind the definition of the potential function can be found in the proof of Lemma 4.7.7.)

Definition 4.7.3 *Let $\mathcal{H} = (V, E)$ be a hypergraph. Let $S \subseteq V$ and let χ be a proper k-coloring of vertices in S. The* potential *of $\langle S, \chi \rangle$ is defined as*

$$\Phi_{\mathcal{H}}(\langle S, \chi \rangle) := \sum_{i=1}^{k} \sum_{j=1}^{\ell-1} n^{j-1} \cdot |\varphi(\langle S, \chi \rangle, i, j)|$$

where

$$\varphi(\langle S, \chi \rangle, i, j) = \left\{ W \subseteq V : |W| = \ell - j \ \& \ \exists e \in E \ (W \subseteq e \ \& \ \forall_{v \in e \setminus W} \ \chi(v) = i) \right\} .$$

Hence if $W \in \varphi(\langle S, \chi \rangle, i, j)$, then coloring all vertices in X with color i creates a monochromatic edge. Our next step is to extend the notion of heavy and conflict vertices to hypergraphs:

Definition 4.7.4 *A vertex $v \in V \setminus S$ is* heavy *with respect to $\langle S, \chi \rangle$ if (i) there is an i, $1 \leq i \leq k$, such that v is i-colorable and (ii) for every i, $1 \leq i \leq k$, if v is i-colorable and χ' is the extension of χ to $S \cup \{v\}$ by coloring v with color i then*

$$\Delta\Phi_{\mathcal{H}}(\langle S, \chi \rangle, v, i) := \Phi_{\mathcal{H}}(\langle S \cup \{v\}, \chi' \rangle) - \Phi_{\mathcal{H}}(\langle S, \chi \rangle) > \frac{\epsilon n^{\ell-1}}{3} .$$

Definition 4.7.5 *A vertex $v \in V \setminus S$ is a* conflict vertex *with respect to $\langle S, \chi \rangle$ if for every i, $1 \leq i \leq k$, v is* not *i-colorable.*

The bases and the violation function are defined in the same way as for graph coloring:

Bases for k-coloring:

- $\{\emptyset, 1\}$ is a basis (where 1 is the encoding of the coloring of the empty set of vertices) and
- if $b = (K, \chi)$ is a basis, v is a *heavy* vertex for b and χ' is an encoding of the previous coloring χ of K extended by a proper coloring of v, then $(K \cup \{v\}, \chi')$ is a basis.

Violation function for k-coloring: *A basis $b = (K, \chi)$ is* violated *by a vertex $v \in V$ if either (i) v is a heavy vertex for $\langle K, \chi \rangle$ or (ii) v is a conflict vertex for $\langle K, \chi \rangle$.*

In a similar way as for the graph coloring problem we can give an upper bound for the dimension of the constructed ACP:

Lemma 4.7.6 *For every problem instance of the hypergraph k-coloring problem the corresponding abstract combinatorial program $\mathcal{P}$ has $\dim(\mathcal{P}) \leq 3k\ell/\epsilon$ and width$(\mathcal{P}) \leq k^{3k\ell/\epsilon}$.*

Proof : Since every k-coloring of a set of r vertices can be encoded using an integer in the range between 1 and k^r, it is enough to show that $|K| \leq 3\,k\,\ell/\epsilon$ for every basis $b = (K, \chi)$. To show this, let us recall that the bases are defined inductively by adding a heavy vertex to another basis. By definition, a heavy vertex increases the potential of the corresponding basis by more than $\frac{1}{3}\,\epsilon\,n^{\ell-1}$. The maximum potential of every basis is less than $k\,\ell\,n^{\ell-1}$ and the starting potential is 0. Thus, it follows for every basis $b = (K, \chi)$ that $|K| \leq 3\,k\,\ell/\epsilon$. □

Our next step is to prove the Distance Preserving property.

Lemma 4.7.7 ($(\epsilon, \epsilon/3)$-Distance Preserving property) *Let $\mathcal{H} = (V, E)$ be a hypergraph that is ϵ-far from being k-colorable and let $S \subseteq V$ be an arbitrary set of vertices colored according to a proper k-coloring χ. Then, either V contains more than $\epsilon\,n/3$ conflict vertices with respect to $\langle S, \chi \rangle$ or V has more than $\epsilon\,n/3$ heavy vertices for $\langle S, \chi \rangle$.*

Proof : Our arguments are similar to those used in the proof of Lemma 4.6.4. The proof is by contradiction. Let us assume there are less than or equal to $\epsilon\,n/3$ heavy vertices and less than or equal to $\epsilon\,n/3$ conflict vertices with respect to $\langle S, \chi \rangle$. Then, we show that it is possible to extend coloring χ of S to a coloring χ^* of V that has at most $\epsilon\,n^\ell$ monochromatic (violating) edges in $\mathcal{H}$. This will yield contradiction.

We define χ^* as follows:

$$\chi^*(v) = \begin{cases} \chi(v) & \text{for every } v \in S \\ 1 & \text{if } v \in V \setminus S \text{ and } v \text{ is a heavy vertex or a conflict vertex with respect to } \langle S, \chi \rangle \\ i & \text{if } v \in V \setminus S \text{ is } i\text{-colorable with respect to } \langle S, \chi \rangle \text{ and } i \text{ minimizes (over all possible choices of proper coloring } i) \text{ the increase in the potential, that is, } \Delta\Phi_{\mathcal{H}}(\langle S, \chi \rangle, v, i) \leq \Delta\Phi_{\mathcal{H}}(\langle S, \chi \rangle, v, j) \text{ for every proper coloring } j \text{ of } v \end{cases}$$

Now, we give an upper bound on the number of monochromatic edges in coloring χ^* of $\mathcal{H}$. Let us first consider heavy and conflict vertices. By our assumption, the number of such vertices is upper bounded by $\frac{2}{3}\,\epsilon\,n$. Therefore, the number of edges incident to these vertices is upper bounded by $\frac{2}{3}\,\epsilon\,n^{\ell}$. Hence, it is sufficient to show that there are at most $\frac{1}{3}\,\epsilon\,n^{\ell}$ monochromatic edges in $\mathcal{H}$ that are not incident to heavy or conflict vertices. We show this indirectly by defining a set $E_D \subseteq E$ of at most $\frac{1}{3}\,\epsilon\,n^{\ell}$ edges. Then we prove that this set contains all monochromatic edges for the coloring χ^*. Let V_{light} denote the set of all vertices in $V \setminus S$ that are neither heavy nor conflict vertices for $\langle S, \chi\rangle$.

For a vertex $v \in V_{light}$ let us define

$$\Delta\varphi(\langle S,\chi\rangle, v, i, j) := \varphi(\langle S \cup \{v\}, \chi'\rangle, i, j) \setminus \varphi(\langle S,\chi\rangle, i, j)$$

where χ' is the extension of χ to $S \cup \{v\}$ by coloring vertex v with color i. We further denote by $E(X, v) = \{e \in E : v \in e \;\&\; X \subseteq e\}$ all edges of the hypergraph that contain $X \cup \{v\}$. Now we make a simple but important observation:

Claim 4.7.8 *If* $E(X, v) = \emptyset$ *then* $X \notin \Delta\varphi(\langle S,\chi\rangle, v, \chi^*(v), j)$.

Proof : Follows immediately from the definition of $\Delta\varphi(\langle S,\chi\rangle, v, \chi^*(v), j)$. □

We want to define the set

$$E_D := \bigcup_{v \in V_{light}} \bigcup_{j \in [\ell-1]} E_{D,j}^{(v)}$$

using sets $E_{D,j}^{(v)}$ that determine for each vertex v a set of edges that is responsible for the sets in $\Delta\varphi(\langle S,\chi\rangle, v, \chi^*(v), j)$. As we will see later these edges are the only edges that may possibly be monochromatic in χ^*. We define the set $E_{D,j}^{(v)}$ as follows:

$$E_{D,j}^{(v)} := \bigcup_{X \in \Delta\varphi(\langle S,\chi\rangle, v, \chi^*(v), j)} E(X, v)$$

Claim 4.7.9 *Let* $e \in E$ *be a monochromatic edge in the coloring* $\langle V, \chi^*\rangle$. *Then* $e \in E_D$.

Proof : If e is monochromatic then there is $i \in [k]$ such that $\chi^*(u) = i$ for all $u \in e$. Further we conclude that for each $u \in e$ we have $\{u\} \in \varphi(\langle V,\chi\rangle, i, \ell-1)$. Now we distinguish between two cases. First let us consider the case when there is a $u \in e$ with $\{u\} \in \varphi(\langle S,\chi\rangle, i, \ell-1)$. In this case $\chi^*(u) \neq i$ by the definition of χ^*. In the second case such a u does not exists but we have $\{u\} \in \varphi(\langle V,\chi\rangle, i, \ell-1)$. But - as can be seen from Claim 4.7.8 - we defined E_D in such a way that there cannot be an $X \in \varphi(\langle V,\chi\rangle, i, \ell-1)$ that is not contained in $\varphi(\langle S,\chi\rangle, i, \ell-1)$. Hence, if e is not in E_D it cannot be monochromatic. □

It remains to show that the size of E_D is small enough. By definition we have

$$\left|E_D\right| = \sum_{v \in V_{light}} \sum_{j=1}^{\ell-1} \left|E_{D,j}^{(v)}\right|$$

and it is easy to see that for the $E^{(\nu)}_{D,j}$ it holds

$$\left|E^{(\nu)}_{D,j}\right| \leq \left|\Delta\varphi(\langle S,\chi\rangle,\nu,\chi^*(\nu),j)\right| \cdot n^{j-1}$$

because of the fact that $|E(X,\nu)| \leq n^{\ell-(|X|+1)}$. We conclude that we have

$$\left|E_D\right| \leq \sum_{\nu\in V_{light}} \sum_{j=1}^{\ell-1} \left|\Delta\varphi(\langle S,\chi\rangle,\nu,\chi^*(\nu),j)\right| \cdot n^{j-1}$$

Since all considered vertices are light we also have for every $\nu \in V_{light}$

$$\Delta\Phi_{\mathcal{H}}(\langle S,\chi\rangle,\nu,\chi^*(\nu)) = \sum_{j=1}^{\ell-1} \left|\Delta\varphi(\langle S,\chi\rangle,\nu,\chi^*(\nu),j)\right| \cdot n^{j-1} \leq \frac{1}{3}\epsilon n^{\ell-1} \ .$$

This finally gives us

$$\left|E_D\right| \leq \frac{1}{3}\epsilon n^{\ell} \ .$$

Together with Claim 4.7.9 we get a contradiction to the fact that $\mathcal{H}$ is ϵ-far from being k-colorable. This proves the lemma. □

The proof of the following lemma is essentially the same as the proof of Lemma 4.6.5 and we present it here for the sake of completeness only.

Lemma 4.7.10 (Feasibility Preserving property) *If for some $S \subseteq V$ every basis covered by S is violated by certain $\nu \in S$, then the sub-hypergraph of $\mathcal{H}$ induced by vertices in S cannot be properly k-colored.*

Proof : The proof is by contradiction. Let us suppose there is a proper k-coloring χ of the subgraph of $\mathcal{H}$ induced by the vertices in S. For every $U \subseteq S$, let $\chi_{|U}$ denote the coloring χ restricted to vertex set U.

Let us observe that the set of bases covered by S is not empty, because it contains the "empty set" basis $(\emptyset,\chi_\emptyset)$. Therefore, there exists a basis (possibly one of many) $b = (U,\chi_{|U})$ covered by S having the maximum size of set U. Since b is violated by certain $\nu \in S$, either ν is a conflict vertex for $\langle U,\chi_{|U}\rangle$ or ν is a heavy vertex for $\langle U,\chi_{|U}\rangle$. Further, since χ was assumed to be a proper k-coloring of S, ν cannot be a conflict vertex for $\langle U,\chi_U\rangle$ and therefore it must be a heavy vertex for $\langle U,\chi_{|U}\rangle$. But this implies that $(U\cup\{\nu\},\chi_{|U\cup\{\nu\}})$ is a basis that is moreover covered by S. This yields a contradiction, because we assumed that there is no basis $(K,\chi_{|K})$ covered by S having the size of K greater than $|U|$. □

The three lemmas above combined with our framework from Theorem 10 imply the following result.

Theorem 12 *There is a property tester for the hypergraph k-colorability with the query complexity*

$$\mathcal{O}\left((k\,\ell\,\epsilon^{-2}\ln(k/\epsilon))^{\ell}\right) = \widetilde{\mathcal{O}}((k\,\ell/\epsilon^2)^{\ell}) \ .$$ □

4.8 Testable Hereditary Graph Properties

In this section we consider arbitrary *hereditary graph properties*. A *graph property* Π is a family of graphs that is preserved under graph isomorphism (that is, if G satisfies property Π and G' is a graph isomorphic to G then G' has property Π, too). A graph property Π is *hereditary* if it is closed under taking induced subgraphs, that is, if graph $G = (V, E)$ has Π then every subgraph G_S induced by a set $S \subseteq V$ has property Π (see, e.g., [20]). Similarly as in Section 4.6, we consider undirected graphs that are represented by a function $f : V \times V \to \{0, 1\}$ that encodes the adjacency matrix of the graph. W.l.o.g. we assume $V = [n]$. When we talk about graph properties we have to observe that in our framework a property is defined as a subset of the set of n-vertex graphs. When no confusion can arise we use Π to denote the graph property Π as well as the corresponding set of n-vertex graphs (which is the formal definition of a property in this thesis). Our main result is that a hereditary graph property is efficiently testable if and only if it can be reduced to an abstract combinatorial program of dimension that is independent of the size of the input graph.

For every graph $G = (V, E)$ and every subset $U \subseteq V$ we denote by G_U the subgraph of G induced by U.

We use the standard distance measure from Definition 2.1.3 for testing graph properties:

Definition 4.8.1 *Let Π be an arbitrary graph property. A graph G is* ϵ-far from (satisfying) Π *if in order to transform G into a graph satisfying Π one has to modify more than ϵn^2 entries in the adjacency matrix of G.*

Now, we can formally state the main result of this section.

Theorem 13 *Let Π be a hereditary graph property. Let $0 < \epsilon < 1$. Let $\mathcal{G}$ be the set of all graphs on the vertex set $V = [n]$. Then, Π is efficiently testable* **if and only if** *there are $\delta = \delta(\epsilon)$, $\rho = \rho(\epsilon)$ and $\lambda = \lambda(\epsilon)$, such that every $G \in \mathcal{G}$ can be mapped to an abstract combinatorial program $\mathcal{P}_G = ([n], \mathcal{B}, \varpi)$ with $\dim(\mathcal{P}_G) \leq \delta(\epsilon)$ and $width(\mathcal{P}_G) \leq \rho(\epsilon)$ satisfying the following two properties:*

(ϵ, λ)-Distance Preserving) *if G is ϵ-far from Π then every basis in $\mathcal{P}_G$ is λ-far from feasible, and*

(Feasibility Preserving) *for every $S \subseteq V$ with $|S| \geq \delta(\epsilon)$: If there is $G' \in \Pi$ with $G_S = G'_S$ then there is a self-feasible basis for S in $\mathcal{P}_G$.*

The remaining part of this section is devoted to the proof of Theorem 13. We begin with the following lemma that simplifies the analysis of property testers for graph properties. By this lemma, if a hereditary graph property Π has a property tester with a query complexity of $q(\epsilon)$ then it has property tester that samples a set S of $r(q(\epsilon))$ vertices and accepts, if and only if the subgraph induced by S has Π.

Lemma 4.8.2 (Alon, In Appendix E of [54]) *Let Π be a hereditary graph property. Suppose there is a property tester for property Π with query complexity $q(\epsilon)$. Then, there is a property tester for property Π that selects uniformly at random a set of $r(q(\epsilon))$ vertices and accepts the input graph if and only if the corresponding induced subgraph satisfies property Π.* □

In order to prove Theorem 13, we must prove that our condition is both necessary and sufficient for efficient testability of any property Π. We first observe that our proof of Theorem 10 with the interpretation I as defined in Section 4.6 implies directly the sufficiency of our conditions. Now, we prove the necessity of our condition.

Lemma 4.8.3 (Necessary Condition) *Let Π be a hereditary graph property. Let $0 < \epsilon < 1$. Suppose there is a property tester for property Π with query complexity $q(\epsilon) \leq \frac{1}{4}n$ and let us set $r = r(q(\epsilon))$. Let $\mathcal{G}$ be the set of all graphs on the vertex set $V = \{1, \ldots, n\}$. Let $\lambda = \lambda(\epsilon) = \frac{1}{3 \cdot 2^r}$. Then, for every $G \in \mathcal{G}$ there exists an abstract combinatorial program $\mathcal{P}_G = ([n], \mathcal{B}, \varpi)$ with $\dim(\mathcal{P}_G) \leq 2r =: \delta$ and width$(\mathcal{P}_G) = 1$ satisfying the (ϵ, λ)-Distance Preserving and the Feasibility Preserving properties.*

Proof : Let us fix an arbitrary graph $G = (V, E)$ for which we describe ACP $\mathcal{P}_G$ satisfying the required properties.

Bases for graph properties: We define the bases of $\mathcal{P}_G$ to be of the form $(K, 1)$ where $K \subseteq V$. We first give a set of basis candidates $\mathcal{BC}$ and then show how to obtain the set of bases from this set of candidates.

We define the set of *basis candidates* $\mathcal{BC}$ as follows:

- $(\emptyset, 1)$ *is a basis candidate,*
- if K is a set of vertices of size r and G_K (the subgraph induced by K) does not satisfies Π, then $(K, 1)$ *is a basis candidate,* and
- if $(K, 1)$ is a basis and $K' \subseteq K$, then $(K', 1)$ *is a basis candidate.*

We also define the notion of *violators* and *light* bases:

Definition 4.8.4 *Let $\mathcal{BC}$ be a set of basis candidates and let $b = (K, 1) \in \mathcal{BC}$. The set of* violators *Viol$^{\mathcal{BC}}(b)$ of b with respect to $\mathcal{BC}$ is defined as follows:*

- *if $|K| = r$ then Viol$^{\mathcal{BC}}(b) = V \setminus K$, and*
- *if $|K| < r$ then Viol$^{\mathcal{BC}}(b) = \{v \in V \setminus K :$ there exists a basis candidate $b' = (K \cup \{v\}, 1)\}$.*

Definition 4.8.5 *Let $\mathcal{BC}$ be a set of basis candidates and let $b \in \mathcal{BC}$. We call* b light *(with respect to $\mathcal{BC}$) if $|\text{Viol}^{\mathcal{BC}}(b)| \leq \frac{3}{2}\lambda n$. If b is not light then it is* heavy.

Now, we define the *set of bases* $\mathcal{B}$ as the output of the following algorithm COMPUTE-BASES:

COMPUTEBASES(Set of candidates $\mathcal{BC}$)
while there is a light basis b (with respect to the current $\mathcal{BC}$) **do**
$\mathcal{BC} = \mathcal{BC} \setminus \{b\}$
return $\mathcal{B} := \mathcal{BC} \cup \{(\emptyset, 1)\}$

Violation function for graph properties: *A basis $b \in \mathcal{B}$ is violated by its violators, that is, by all vertices contained in $Viol^{\mathcal{B}}(b)$.*

Notice that our construction of bases and the definition of the violation function implies that every basis $b \in \mathcal{B}$ is heavy, that is, it is violated by more than $\frac{3}{2}\lambda n$ vertices in V.

Now, the following claim follows trivially from our construction.

Claim 4.8.6 *$\mathcal{P}_G$ has dimension $(r, 1)$.* □

(ϵ, λ)-Distance Preserving Property: We prove now that the ACP $\mathcal{P}_G$ defined above satisfies the (ϵ, λ)-Distance Preserving Property, that is, that if G is ϵ-far from Π then every basis in $\mathcal{P}_G$ is λ-far from feasible, where $\lambda = \lambda(\epsilon) = \frac{1}{3 \cdot 2^r}$. Notice that by the definition of bases, every basis except for $(\emptyset, 1)$ is violated by more than λn ground set elements. Therefore, what remains to be proven is that also the basis $(\emptyset, 1)$ is violated by more than λn ground set elements. Our proof of this fact is via a sequence of claims that top-down establish lower bounds for the number of bases of given size.

For a basis $b = (K, 1)$, let $|K|$ be called its *size*. We begin with the following simple claim about bases of size r.

Claim 4.8.7 *Suppose $r < (1 - \frac{2}{3}\lambda)n$. If G is ϵ-far from Π, then the number of bases of size r in $\mathcal{B}$ is bigger than $\frac{2}{3} \cdot \binom{n}{r}$.*

Proof : The proof follows directly from the definition of the bases and from Lemma 4.8.2. Indeed, Lemma 4.8.2 implies that for a graph that is ϵ-far from Π, if one picks at random a set of r vertices in V, then with probability at least $\frac{2}{3}$ the subgraph induced by these vertices does not satisfy Π. This is equivalent to say that the number of subsets of V of size r for which the induced subgraph does not satisfy Π is bigger than $\frac{2}{3} \cdot \binom{n}{r}$. By the definition of the basis candidates, for every set $K \subseteq V$ of size r, $(K, 1) \in \mathcal{BC}$. Moreover, the set $V \setminus K$ is the set of violators of K. Hence $(K, 1)$ is a basis that is violated by $n - r$ violators, and thus $(K, 1)$ is heavy. Thus, $(K, 1)$ belongs to $\mathcal{B}$. Therefore, the number of bases of size r in $\mathcal{B}$ is bigger than $\frac{2}{3} \cdot \binom{n}{r}$. □

The next claim deals with the relation between the number of bases of size k and of size $k - 1$.

Claim 4.8.8 *Suppose that $n \geq 2r$. Let ζ, $0 \leq \zeta \leq 1$, and let k, $1 \leq k \leq r$, be an integer. If there are more than $\zeta \cdot \binom{n}{k}$ bases of size k in $\mathcal{B}$ then the number of bases in $\mathcal{B}$ of size $k - 1$ is bigger than $\frac{\zeta - 3\cdot\lambda}{2} \cdot \binom{n}{k-1}$.*

Proof : Recall that $\mathcal{B}$ contains only heavy bases. Furthermore, if (K, l) is a basis then our construction of bases ensures that for every $u \in K$, $(K \setminus \{u\}, l)$ is a basis (in the following we refer with basis also to basis candidates) and that this basis is violated by u. Thus, every basis of size k defines k violators for bases for size $k-1$. We conclude that overall, there are more than

$$k \cdot \zeta \cdot \binom{n}{k} = (n-k+1) \cdot \zeta \cdot \binom{n}{k-1} > \frac{1}{2} \cdot n \cdot \zeta \cdot \binom{n}{k-1}$$

violators for bases of size $k-1$. Observe that every light basis has at most $\frac{3}{2}\lambda n$ violators. Therefore, the number of violators of all light bases of size $k-1$ is at most $\frac{3}{2}\lambda n \binom{n}{k-1}$. It follows that the number of violators of heavy bases of that size is bigger than

$$\frac{1}{2} \cdot n \cdot \zeta \cdot \binom{n}{k-1} - \frac{3}{2} \cdot \lambda \cdot n \cdot \binom{n}{k-1} = \frac{1}{2} \cdot (\zeta - 3\lambda) \cdot n \cdot \binom{n}{k-1} \ .$$

Since each basis has at most n violators it follows that the number of heavy bases is larger than

$$\frac{1}{2} \cdot (\zeta - 3\lambda) \cdot \binom{n}{k-1} \ ,$$

which completes the proof of our claim. □

The next claim gives a lower bound for the number of bases of given size (bigger than 0).

Claim 4.8.9 *Let $1 \leq k \leq r$ be an integer. Then the number of bases of size $r-k$ is bigger than*

$$\left(\frac{1}{3 \cdot 2^{k-1}} - \frac{3}{2}\lambda\left(2 - \frac{1}{2^{k-1}}\right)\right) \cdot \binom{n}{r-k} \ .$$

Proof : The proof is by induction on k. For $k = 1$ the claim follows directly from Claims 4.8.7 and 4.8.8. Now, let us assume the claim is true for $k = k'$, for certain $1 \leq k' < r$, and we show that this implies that the claim is true also for $k = k' + 1$. By induction, this will yield the claim.

By the induction hypothesis, the number of bases in $\mathcal{B}$ of size $r - k'$ is bigger than

$$\begin{aligned} & \left(\frac{1}{3 \cdot 2^{k'-1}} - \frac{3}{2}\lambda\,(2 - \frac{1}{2^{k'-1}})\right) \cdot \binom{n}{r-k'} \\ = \ & \left(\frac{1}{3 \cdot 2^{k-2}} - \frac{3}{2}\lambda\left(2 - \frac{1}{2^{k-2}}\right)\right) \cdot \binom{n}{r-k+1} \ . \end{aligned}$$

Therefore, by Claim 4.8.8, the number of bases in $\mathcal{B}$ of size $r-k = (r-k') - 1$ is bigger than

$$\frac{\left(\frac{1}{3 \cdot 2^{k-2}} - \frac{3}{2}\lambda\left(2 - \frac{1}{2^{k-2}}\right)\right) - 3\lambda}{2} \cdot \binom{n}{r-k} = \left(\frac{1}{3 \cdot 2^{k-1}} - \frac{3}{2}\lambda\left(2 - \frac{1}{2^{k-1}}\right)\right) \cdot \binom{n}{r-k}$$

and the claim follows. □

Now, we are ready to complete the proof of the (ϵ, λ)-Distance Preserving Property for $\mathcal{P}_G$. By our discussion above, we only must show that the basis $(\emptyset, 1)$ is violated by more than λn ground set elements. By Claim 4.8.9, the number of bases in $\mathcal{B}$ of size 1 is bigger than

$$\left(\frac{1}{3 \cdot 2^{r-2}} - 3 \cdot \lambda\right) \cdot \binom{n}{1} = \left(\frac{4}{3 \cdot 2^{r}} - 3 \cdot \lambda\right) \cdot n \geq \lambda \cdot n ,$$

by our assumption that $\lambda = \frac{1}{3 \cdot 2^r}$. Each of the bases of size 1 is of the form $(\{u\}, 1)$ for some $u \in V$ and by the definition of violation, such a vertex u violates $(\emptyset, 1)$. Since the number of such vertices u is bigger than $\lambda \cdot n$, this completes the proof of the (ϵ, λ)-Distance Preserving Property for $\mathcal{P}_G$.

Feasibility Preserving Property: We prove now that the ACP $\mathcal{P}_G$ satisfies the Feasibility Preserving Property. We prove that for every $S \subseteq V$, $|S| \geq \delta$, if the subgraph G_S satisfies Π then there is a self-feasible basis for S in $\mathcal{P}_G$. This implies the Feasibility Preserving Property because Π is hereditary and so: If G_S does not have property Π then G cannot have Π, as well.

Our arguments are roughly the same as in the proof of Lemma 4.6.5. The proof is by contradiction. Let us suppose that the subgraph G_S satisfies Π, but every basis in S is violated. Observe that the set of bases covered by S is not empty, because it contains the "empty set" basis $(\emptyset, 1)$. Therefore, there exists a basis $b = (K, 1)$ covered by S maximizing the size of set K. Since b is violated by certain $v \in S$ and since K is of maximum size, we conclude that $|K| = r$. It follows that the subgraph G_K induced by K does not have property Π. Since Π is hereditary, it is closed under taking induced subgraphs, and hence we conclude that G_S does not have property Π.

If $n < 2r$ then we use the following simple ACP: The ACP has a single basis $(V, 1)$ which is violated by all ground set elements, if G is ϵ-far from Π and which is not violated otherwise. This completes the proof of Lemma 4.8.3. □

Now, we can conclude our discussion to complete the proof of Theorem 13.

Proof : The "only if" part follows from Lemma 4.8.3 and the "if" part follows from Theorem 10 with the interpretation I as defined in Section 4.6. □

5 Testing Algorithms with Geometric Queries

In Chapter 3 we considered Property Testing in the 'standard model' for geometric objects such as point sets. This means that the property testing algorithm is only allowed to ask basic queries about the given object. In the case of point sets, our algorithm was allowed to ask queries of the form 'What is the position of the i-th point of the point set ?'.

In this section we want to introduce a new type of query for point sets: Range queries. Range queries are of the form: 'What is the i-th point in a query range R ?' where R is a range in the $\mathbb{R}^d$ (typically, an axis parallel rectangle or a simplex). Such queries are efficiently supported by basic spatial data structures, e.g. range trees and partition trees [1].

Later in this section we show that the property of being in convex position (in the $\mathbb{R}^2$) can be tested with $\mathcal{O}(\log n/\epsilon)$ triangular range queries. This stands in contrast to the lower bound of $\Omega(\sqrt[d+1]{n^d/\epsilon})$ on the query complexity in the standard model from Chapter 3. We also consider two other examples, labeling and clustering, and show that we can achieve much better results in the new model than it would be possible in the standard model.

5.1 Property Testing with Range Queries

Many fundamental data structures for point sets are designed to efficiently answer *range queries* (cf. [1]), i.e. queries that ask to return all points (or just their number) located in a given query range. Usually, the shape of a query range is restricted to a certain class of ranges, e.g. axis parallel rectangles or simplices. This way it is possible to answer queries much more efficient than it would be possible for arbitrary query ranges.

In this section we introduce models of computation whose basic operations are range queries. All models are defined for point sets in the $\mathbb{R}^d$. Let P denote the point set under consideration.

The *Simplex Range Query Model*. In the *Simplex Range Query Model* a Property Testing algorithm is allowed to ask the following types of queries:

- 'What is the number of points of P contained in a (possibly degenerated) simplex S ?'.

- 'What is the position of the i-th point of P contained in a (possibly degenerated) simplex S ?' The ordering of the points of P contained in simplex S is fixed but unknown to the property testing algorithm. If i is bigger than the number of points contained in S then the symbol $\emptyset$ is returned to denote that such a point does not exist.
- 'What is the position of the i-th point of P ?' The ordering of the points of P is fixed but unknown. If i is bigger than $|P|$ then the symbol $\emptyset$ is returned. Such a query could be realized using the second type of query with a sufficiently large simplex. For convenience in notation we introduce this as a separate type of query.

Of course, the simplex S specified by the property testing algorithm may differ from query to query. This model is motivated by the large number of efficient data structures that support triangular range queries such as partition trees and cutting trees. The most efficient data structure for simplex range counting queries requires $O(n^{d+\epsilon})$ space (for any $\epsilon > 0$) and supports queries in time $O(\log^d n)$ [1].

The *Rectangle Range Query Model*. In the *Rectangle Range Query Model* a property testing algorithm is allowed to ask the following types of queries:

- What is the number of points of P contained in a (possibly unbounded) axis parallel rectangle R ?'
- 'What is the position of the i-th point of P contained in a (possibly unbounded) axis parallel rectangle R ?' The ordering of the points of P contained in rectangle R is fixed but unknown to the Property Testing algorithm. If i is bigger than the number of points contained in R then the symbol $\emptyset$ is returned to denote that such a point does not exist.
- 'What is the position of the i-th point of P ?' The ordering of the points of P is fixed but unknown. If i is bigger than $|P|$ then the symbol $\emptyset$ is returned. Similar to the Simplex Range Query Model such a query could be realized using the second type of query with a sufficiently large rectangle.

Orthogonal range queries are supported by many fundamental data structures such as range trees, R-tree, and quad trees. The most efficient data structure for orthogonal range counting queries in the RAM model supports queries in $O(\log^{d-1} n)$ time and uses $O(n \log^{d-2} n)$ space [1].

The *Mixed Range Query Model.* In the *Mixed Range Query Model* the property testing algorithm may ask every query that is allowed in the Simplex Range Query Model as well as every query that is allowed in the Rectangle Range Query Model.

Some further definitions. In all three models the *query complexity* of a property testing algorithm is the number of queries asked about the point set. The size $n = |P|$ of the point set is known to the testing algorithm.

5.2 Testing Convex Position with Range Queries

In Chapter 3 we designed a property tester for convex position in the standard property testing model. Its query complexity was $\mathcal{O}(\sqrt[d+1]{n^d/\epsilon})$. We also proved a matching lower bound on the query complexity of every testing algorithm in the standard testing model. Obviously, the simplex range query model is more powerful than the standard testing model. But how much does the possibility to use simplex range queries help us when we want to design property testing algorithms ?

We first design and analyze a property tester for convex position in the simplex range query model. We only consider the 2-dimensional case when the point set P is in the $\mathbb{R}^2$. Again we assume that the input point set is in general position. We prove that in the simplex range query model there is a property tester for convex position with query complexity $\mathcal{O}(\log n/\epsilon)$. Hence it is possible to achieve an exponential improvement in the number of queries using this new model.

Our algorithm is based on a randomized test for extremality of points. Hence, we first have to prove that many points of a point set are not extreme, if the set is ϵ-far from convex position. This is done in the following Proposition:

Proposition 5.2.1 *Let P be a point set that is ϵ-far from convex position. Then more than $\epsilon|P|$ points in P are not extreme.*

Proof : Let P be ϵ-far from convex position. Assume less than $\epsilon|P|$ points in P are not extreme. Then we can delete these points from P to obtain a point set in convex position. Hence the point set P cannot be ϵ-far from convex position. This is a contradiction. □

Our next step is to show that the simple algorithm IsFacet decides if two points are the vertices of the same facet (segment) of the convex hull.

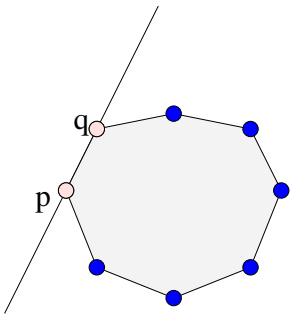

Figure 5.1: If two points p and q belong to the same facet of the convex hull then one of the halfspaces induced by a line through p and q contains exactly 2 points.

IsFacet(p, q)
Let $\mathcal{H}^+$ and $\mathcal{H}^-$ denote the two halfspaces induced by the line through p and q
if $\mathcal{H}^+$ contains exactly 2 points from P **then** accept
if $\mathcal{H}^-$ contains exactly 2 points from P **then** accept
reject

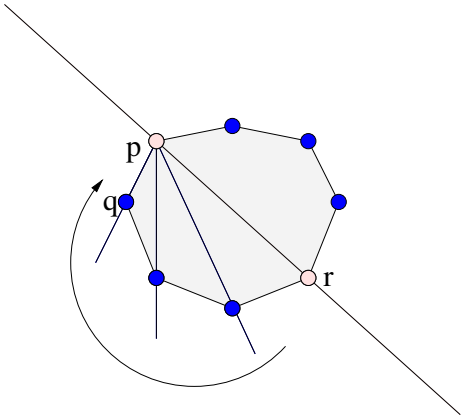

Figure 5.2: A circular ordering for all points in a halfspace induced by a line through p and r.

We show the following lemma:

Lemma 5.2.2 *Let $p, q \in P$ be two points from a point set $P \subseteq \mathbb{R}^2$. Then algorithm IsFacet(p, q) decides whether p and q are the vertices of the same facet of the convex hull of P. The algorithm uses 2 simplex range queries.*

Proof : First of all, let us observe that two points $p, q \subseteq P$ are the vertices of the same facet of the convex hull of P, if and only if one of the closed halfspaces induced by a line through p and q contains exactly the points p and q. A halfspace range query is a degenerated triangular range query and so we are allowed to use such a query in the simplex range query model. Hence the algorithm is correct and obviously uses at most 2 queries. □

So far, we can only find out whether two points belong to the same facet of the convex hull of P. This would imply that both of them are extreme points. We use this possibility to design an algorithm that finds out whether a single point p is extreme. If p is extreme (and if $|P| > 1$) then there is a point $q \in P$ such that p and q are vertices of the same facet of the convex hull of P. Thus we have to design a procedure that finds such a point q, if p is extreme.

The idea of our procedure is simple: Given an extreme point p we pick another point r uniformly at random from P. If pr is a facet of the convex hull of P then we have a witness that p is extreme. If not we select one of the halfspaces induced by a line through p and r. Let us call the selected halfspace $\mathcal{H}$ (see Figure 5.2). Then we observe that the point p induces a circular ordering among the points in halfspace $\mathcal{H}$. The 'last' point in this ordering is the point q we are looking for. We can find this point efficiently using a randomized binary search-like procedure on the points within the halfspace.

EXTREMECHECK (P, p)
  Choose a point $r \in P$ uniformly at random
  if ISFACET(p, r) **then return** p is extreme
  Let $\mathcal{H}$ denote a closed halfspace induced by the line through p and r
  Let τ_1 denote the (degenerated) triangle $\mathcal{H}$
  $i = 1$
  while τ_i contains more than 2 points and $i \leq 100 \log n$ **do**
    Choose a point $s_i \in \tau_i$ uniformly at random
    if $s_i \neq p$ and $s_i \neq r$ **then**
      Let $\mathcal{H}_s^{(i)}$ denote the halfspace induced by a line through p and s_i that does not contain r
      Let τ_{i+1} denote the intersection of $\mathcal{H}$ and $\mathcal{H}_s^{(i)}$
    $i = i + 1$
  if τ_i contains more than 2 points **then return** don't know
  else if ISFACET(p, s_i) **then return** p is extreme
    else return p is not extreme

We first show that the algorithm is correct, if it returns an answer other than "don't know".

Lemma 5.2.3 *Let p be an extreme point in P. If algorithm* EXTREMECHECK *is started with input p then the algorithm returns either "p is extreme" or "don't know".*

Proof : We first observe that p is incident to two facets of the convex hull of P since p is an extreme point. The segment pr is not a facet of the convex hull of P, since otherwise the algorithm had accepted the input in the first **if**-statement by Lemma 5.2.2. Now let q_1 and q_2 denote the points of p that share a facet with p in the convex hull of P. Then either one of the points must be in halfspace $\mathcal{H}$. If τ_{i+1} contains more than 2 points then we know that s_i cannot share a facet with p because both closed halfspaces induced by a line through p and s_i contain at least 3 points. One of the halfspaces contains p, s_i, and r and the other one contains all points in τ_{i+1}. Hence s_i can only share a facet with p, if τ_{i+1} contains exactly 2 points. If the algorithm does not answer "don't know" it computes a triangle that contains exactly 2 points. As we already observed there is a point in $\mathcal{H}$ that shares a facet with p it must be the computed point. □

Lemma 5.2.4 *Let p be a point in P that is not extreme. If algorithm* EXTREMECHECK *is started with input p then the algorithm returns either "p is not extreme" or "don't know".*

Proof : By Lemma 5.2.2 we know that the algorithm only return "p is extreme" if p is a vertex of a facet of the convex hull of P. The lemma follows since p cannot be a vertex of a facet of the convex hull, if it is not extreme. □

Then we show that the algorithm typically returns an answer other than "don't know".

Lemma 5.2.5 *Let P be a point set of n points in the $\mathbb{R}^2$. Let X_i^p denote the random variable for the number of points in triangle τ_i when running algorithm* EXTREMECHECK *with input $p \in P$. Let Z_i denote the indicator random variable for the event that $X_{i+1}^p \leq \frac{2}{3}X_i^p$. Then we have*

$$\mathbf{Pr}[Z_i = 1] \geq \frac{1}{4}$$

Proof : Let Q_i denote the set of points that are contained in triangle τ_i. We observe that $\mathbf{Pr}[s_i = q] = \frac{1}{|Q_i|}$ for each point $q \in Q_i$. For the sake of analysis we enumerate all points in Q_i but p according to their circular ordering around p. Then it is immediate that $X_{i+1}^p \leq \frac{2}{3}X_i^p$ holds, if s_i belongs to the last $\lfloor \frac{2}{3}|Q_i| \rfloor - 1$ points in this particular ordering. It follows for $|Q_i| > 2$ that

$$\mathbf{Pr}[Z_i = 1] \geq \frac{\lfloor 2/3|Q_i| \rfloor - 1}{|Q_i|} \geq \frac{1}{4}.$$

□

Lemma 5.2.6 *Algorithm* EXTREMECHECK *returns the answer "don't know" with probability less than $1/3$.*

Proof : We first observe that

$$\mathbf{Pr}[\text{EXTREMECHECK returns "don't know"}] \leq \mathbf{Pr}\Big[\sum_{1 \leq i \leq 100 \log n} Z_i \leq \log_{3/2} n\Big]$$

Now let Y_i be a 0-1 random variable with $\mathbf{Pr}\big[Y_i = 1\big] = 1/4$. Then by Lemma 5.2.5 we observe that

$$\begin{aligned}
\mathbf{Pr}\Big[\sum_{1 \leq i \leq 100 \log n} Z_i \leq \log_{3/2} n\Big] &\leq \mathbf{Pr}\Big[\sum_{1 \leq i \leq 100 \log n} Y_i \leq \log_{3/2} n\Big] \\
&= \mathbf{Pr}\Big[\sum_{1 \leq i \leq 100 \log n} Y_i \leq \frac{4 \log n}{\log(3/2)} \mathbf{E}[Y_i]\Big] \\
&\leq \mathbf{Pr}\Big[\sum_{1 \leq i \leq 100 \log n} Y_i \leq 8 \log n \mathbf{E}[Y_i]\Big] \\
&\leq \mathbf{Pr}\Big[\sum_{1 \leq i \leq 100 \log n} Y_i \leq (1 - \frac{9}{10}) 100 \log n \cdot \mathbf{E}[Y_i]\Big]
\end{aligned}$$

We apply a Chernoff bound [56] to the last inequality and obtain:

$$\mathbf{Pr}\Big[\sum_{1 \leq i \leq 100 \log n} Y_i \leq (1 - \frac{9}{10}) 100 \log n \cdot \mathbf{E}[Y_i]\Big] \leq e^{-(9/10)^2 25 \log n/2} \leq e^{-10 \log n} \leq 1/3$$

□

Now we can come up with the testing algorithm:

```
CONVEXTESTER-C(P, ε)
    for i = 1 to 6ε^{-1} do
        choose a point p from P uniformly at random
        if EXTREMECHECK(P, p) returns "p is not extreme" then reject
    accept
```

Theorem 14 *Algorithm* CONVEXTESTER-C *is a property tester for convex position in the plane in the simplex range query model. Its query complexity is* $\mathcal{O}(\log n/\epsilon)$.

Proof : We observe that the algorithm only rejects a point set, if algorithm EXTREMECHECK returns that the point is not an extreme point. By Lemma 5.2.3 algorithm EXTREMECHECK always returns "p is extreme" or "don't know" if p is extreme. Hence every point set in convex position is accepted because every point is an extreme point.

It remains to prove that a point set P is rejected, if P is ϵ-far from convex position. When a point p is chosen uniformly at random from P then by Proposition 5.2.1 the probability that p is not extreme is at least ϵ. By Lemma 5.2.6 we have that the probability that EXTREMECHECK answers "don't know" is at most $1/3$. We conclude that the probability that a point chosen uniformly at random by EXTREMECHECK is not extreme *and* EXTREMECHECK rejects it, is at least $2\epsilon/3$. Hence the probability that algorithm CONVEXTESTER-C does not reject a point set that is ϵ-far from convex position is at least

$$\mathbf{Pr}\big[\text{CONVEXTESTER-C accepts}\big] \leq (1-2\epsilon/3)^{6\epsilon^{-1}} \leq e^{-4} \leq 1/3$$

and hence,

$$\mathbf{Pr}\big[\text{CONVEXTESTER-C rejects}\big] \geq 2/3$$

□

5.3 Map Labeling

We continue the investigations in the power of our new model. We now consider a basic *map labeling* problem. Map labeling in general deals with problems where to place labels, e.g. names of towns, rivers, mountains, and oceans, on a geographic map. The constraints for such a placement of labels are fairly obvious: Labels should be close to the labeled feature and labels should not intersect. A basic map labeling problem can be formulated in the following way [47]:

Let P *be a set of* n *points in the* $\mathbb{R}^2$. *Decide whether it is possible to place* n *axis-parallel unit squares such that*

- *all squares are pairwise disjoint (labels do not overlap),*
- *each point is a corner of exactly one square (each point is labeled), and*
- *each square has exactly one point on its corners (each point has a unique label).*

If a set S of n squares satisfies the conditions above, then S is called a *valid labeling for* P. The map labeling problem is known to be $\mathcal{NP}$-complete and the corresponding optimization problem (with the goal to maximize the label size) is known to have no approximation algorithm with ratio better than 2, unless $\mathcal{P} = \mathcal{NP}$ [47].

Our goal is to design a property testing algorithm for the above labeling problem. We use the standard distance measure from Definition 2.1.3. In terms of map labeling this can be formulated as follows:

Definition 5.3.1 *A set P of n points in the plane is ϵ-far from having a valid labeling, if we have to delete more than $\epsilon\, n$ points to obtain a set of points that has a valid labeling.*

Before we consider the map labeling problem in the orthogonal range query model, we prove a strong lower bound on the query complexity in the standard testing model. This bound shows that we cannot hope for a property testing algorithm that has $o(n^{1-\delta})$ query complexity for every constant $\delta > 0$.

Theorem 15 *For every constant δ, $0 < \delta < 1$, there is a positive constant ϵ such that there is no property tester for the labeling problem with $o(n^{1-\delta})$ query complexity in the standard testing model.*

Proof : For a given δ let us define $c = \lceil \frac{1-\delta}{2\delta} \rceil$ and let $\epsilon = \frac{1}{2(2c+1)}$. We observe that by the chosen value of c it holds that $\delta \geq \frac{1}{2c+1}$. We construct a point set that is ϵ-far from having a valid labeling. The set consists of $n/(2c+1)$ copies of a set Q_c of $2c+1$ points. Q_c has the property that it does not have a valid labeling but if we delete one point from it, then it always has a valid labeling. A property tester with query complexity $o(n^{1-\delta})$ that samples a subset of points uniformly at random cannot reject such a point set with probability $2/3$.

Therefore, we first show that the existence of a property tester for a point set P implies the existence of a property tester with similar query complexity that selects a sample set S from P uniformly at random and accepts if and only if S has a valid labeling:

Claim 5.3.2 *Let $\mathbb{A}$ be a property tester for the map labeling problem with query complexity $q(n, \epsilon)$. Then there exists a property tester $\mathbb{A}''$ for the map labeling problem that sample a set S of $q(\epsilon, n)$ points uniformly at random and accepts if and only if the points in S have a valid labeling. Such a property tester is called* canonical.

Proof : By Lemma 2.3.1 we know that there is a property tester $\mathbb{A}'$ that samples a set S of $q(\epsilon, n)$ points uniformly at random and decides based on S and and its internal coin flips. Let $\mathbb{A}''$ be an algorithm that samples a set S of $q(\epsilon, n)$ points uniformly at random and that accepts if and only if S has a valid labeling. We want to prove that $\mathbb{A}''$ is a property tester. Clearly, algorithm $\mathbb{A}''$ accepts every point set having a valid labeling. Thus it remains to prove that it rejects every point set that is ϵ-far from having a valid labeling. We show this indirectly by proving the following statement: If algorithm $\mathbb{A}'$ rejects a sample set S then so does algorithm $\mathbb{A}''$. By definition a property tester has one-sided error and so $\mathbb{A}'$ must accept if the sample set has a valid labeling. But if the sample S does not have a valid

labeling we know that $\mathbb{A}''$ rejects the input. Hence, $\mathbb{A}''$ always rejects, if $\mathbb{A}'$ rejects. Since $\mathbb{A}'$ is a property tester it rejects every point set P that is ϵ-far from having a valid labeling with probability at least $2/3$. Thus algorithm $\mathbb{A}''$ is a property tester. This proves Claim 5.3.2. □

Now we show that there exists a point set that is ϵ-far from having a valid labeling and that is not rejected by a canonical property tester with query complexity $o(n^{1-\delta})$ with probability more than $1/3$. Our construction is based on a point set Q_c of size $2c+1$ with the following two properties:

- Q_c does not have a valid labeling,
- for every $p \in Q_c$ the set $Q_c \setminus \{p\}$ has a valid labeling.

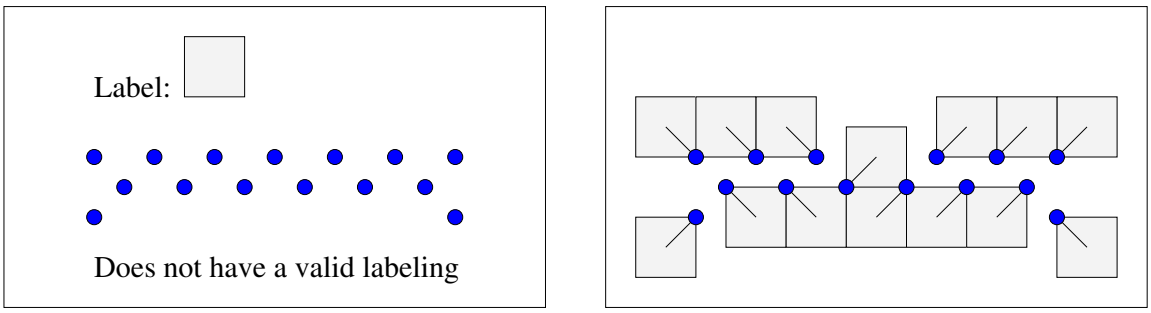

Figure 5.3: The lower bound construction. If one point is missing then the set has a valid labeling (right figure).

Claim 5.3.3 *For every $c \geq 2$ there exists a set Q_c of $2c+1$ points such that the set Q_c does not have a valid labeling and such that $Q_c \setminus \{p\}$ has a valid labeling for every $p \in Q_c$.*

Proof : For a given $c \geq 2$ we construct Q_c. W.l.o.g. we assume that two squares intersect if and only if their interiors intersect. We place points l_c and r_c at position $(1,0)$ and $(c,1)$, respectively, a set U_c of c points at the position $(1,1),\ldots,(c,1)$, and a set L_c of $c-1$ points at positions $(1.5,0.5),\ldots,(c-0.5,0.5)$. We claim that the set $Q_c = U_c \cup L_c \cup \{l_c, r_c\}$ does not have a valid labeling. This is because we have only $|L_c|-1$ label positions for the points in L_c that do not necessarily intersect with points from $U_c \cup \{l_c, r_c\}$. Hence there cannot be a valid labeling for Q_c. On the other hand, for every $q \in Q_c$ the set $Q_c \setminus \{q\}$ has a valid labeling as one easily verifies by a case distinction (see also the figure above):

- In the first case we either have $q = l_c$ or $q = r_c$ (these cases are symmetric). Wlog. let $q = l_c$. Then we choose the upper left square for all points in U_c, the lower left square for all points in L_c and the lower right square for r_c.
- Now let q be in L_c and let q have the position $(i, 0.5)$. We choose the upper left square for all points in U_c, the lower right square for all points $r = (j, 0.5) \in L_c$ with $j < i$ and the the lower left square for the remaining points in L_c. For the points l_c and r_c we choose the lower left and lower right square, respectively.

- It remains to consider the case $q \in U_c$. Let us assume that q has position $(i, 1)$. Then we pick the upper left square for all points $r = (j, 1) \in U_c$ with $j < i$ and the upper right square for the remaining points in U_c. For all points $r = (j, 0.5) \in L_c$ with $j < i - 1$ we choose the lower right square. For $(i - 0.5, 0.5) \in L_c$ we choose the upper right square, and for the remaining points from L_c we choose the lower left square. For the points l_c and r_c we choose the lower left and lower right square, respectively.

In all cases we easily verify that the labels do not intersect. Hence Claim 5.3.3 follows. □

Now we can construct a set P of n points that consists of $k = \lfloor \frac{n}{2c+1} \rfloor$ translated copies $W_1, \ldots, W_k$ of the set Q_c (such that their possible labelings do not intersect). In order to obtain a set with a valid labeling from P we must delete at least one point from each W_i, $1 \leq i \leq k$. Thus we have that P is ϵ-far from having a valid labeling. Further a canonical property tester can reject P only if the sample set S contains at least one of the sets W_i. If we now apply Lemma 2.4.2 with $l = 2c + 1$ and $p = 1/2$ then we obtain that

$$\mathbf{Pr}\big[\exists j \in [k] : (W_j \in S)\big] \leq 1/2$$

if the sample size is less than $(n - 2c) \cdot \left(\frac{1}{2\epsilon n}\right)^{\frac{1}{2c+1}} = \Omega(n^{1-\delta})$. Hence there is no canonical property tester with query complexity $o(n^{1-\delta})$ and so there cannot be any property tester with that query complexity. □

We show now that if we use the computational model that allows range queries we can design a tester with $\mathcal{O}(1/\epsilon^3)$ query complexity. It is based on the approach developed in [36] and [58].

LABELTEST(P):
  choose a sample set S of size $4/\epsilon$ uniformly at random from P
  for each $p \in S$ **do**
    $i = 0$, $\mathcal{T} = \emptyset$
    Let R be the axis parallel square with center p and side length $20\lceil 1/\epsilon \rceil$
    while $i \leq (20\lceil 1/\epsilon \rceil + 2)^2$ **do**
      Let q be the i-th point in the query range R
      if $q \neq \emptyset$ **then** $\mathcal{T} = \mathcal{T} \cup \{q\}$
      $i = i + 1$
    if $\mathcal{T}$ does not have a valid labeling **then** *reject*
  accept

Theorem 16 *Algorithm* LABELTEST *is a tester for the labeling problem that has query complexity* $\mathcal{O}(1/\epsilon^3)$ *and running time* $\exp(\mathcal{O}(1/\epsilon^2))$.

Proof : First of all, let us observe that algorithm LABELTEST accepts every input point set that has a valid labeling. Thus we have to show that every instance P that is ϵ-far

from having a valid labeling is rejected with probability at least $2/3$. Let us assume that P is ϵ-far from having a valid labeling. Now we partition the plane into grid cells of side length $10\lceil 1/\epsilon \rceil$ such that at most $\epsilon n/2$ points of P are within a distance of 1 or less to the boundary of the grid cell they are contained in. Such a partition can be constructed in the following way: We start by partitioning the plane into vertical stripes $A_i = \{(x,y) \in \mathbb{R}^2 : 2.5i \leq x < 2.5(i+1)\}$. For $j \in [k]$, with $k = 4\lceil \frac{1}{\epsilon} \rceil$, let $R_j = \bigcup_{i:i \bmod k = j} A_i$. Then we have that there is $m \in [k]$ with $|R_m \cap P| \leq \epsilon n/4$. If we put the vertical lines of the grid cells in the middle of the stripes in R_m then at most $\epsilon n/4$ points are within a distance of 1 to these vertical lines. A similar argument applied to horizontal lines gives us the desired grid partition.

For the analysis we delete all points that are within a distance of 1 or less to the boundary of their grid cell from the point set P. This way we obtain the point set P'. Let $X_{confl} \subseteq P'$ be a minimal-size set of points such that $P' \setminus X_{confl}$ has a valid labeling. Since P is ϵ-far from having a valid labeling we know that $X_{confl} > \epsilon n/2$. Now we consider a single grid cell C. By the construction of P' we know that the labeling of the points in C cannot intersect with labels from points outside of C.

Claim 5.3.4 *If $S \cap X_{confl} \neq \emptyset$ then algorithm* LABELTEST *rejects.*

Proof : Let x be a point from X_{confl} that is contained in S and let C denote the grid cell containing x. The **while**-loop either adds all points in R to $\mathcal{T}$ or it finds out that more than $(20\lceil 1/\epsilon \rceil + 2)^2$ points are contained in R. In the case that $\mathcal{T}$ contains all points in R it must also contain all points from C. Thus the points in $\mathcal{T}$ do not have a valid labeling. In the case when we have more than $(20\lceil 1/\epsilon \rceil + 2)^2$ points in $\mathcal{T}$ we would have to fit more than $(20\lceil 1/\epsilon \rceil + 2)^2$ unit square labels on an area of size $(20\lceil 1/\epsilon \rceil + 2)^2$. Thus there cannot be a labeling in this case, either. Hence the algorithm rejects, if $S \cap X_{confl} \neq \emptyset$. □

By Claim 5.3.4 the theorem follows with

$$\mathbf{Pr}\left[S \cap X_{confl} = \emptyset\right] \leq (1 - \epsilon/2)^{4/\epsilon} \leq 2/3$$

and so

$$\mathbf{Pr}\left[S \cap X_{confl} \neq \emptyset\right] \geq 2/3 \ .$$

□

5.4 Clustering Problems

In this section we revisit the radius and diameter clustering problem as defined in Chapter 4. Recall that the goal of a clustering problem is to decide whether a set of n points P in $\mathbb{R}^d$ can be partitioned into k subsets (called *clusters*) $S_1, \ldots, S_k$ such that the *cost* of each cluster is at most b. There are several different ways to define the cost of a cluster. Let S be a set of points in $\mathbb{R}^d$. In the radius clustering problem the cost $\mathrm{cost}_R(S)$ of a cluster S is twice the minimum radius of a ball containing all points of the cluster. In the diameter

clustering problem the cost $\mathrm{cost}_D(S)$ of a cluster S is the maximum distance between a pair of points of the cluster.

The goal of our property tester is to accept all instances that admit a clustering into k subsets of cost b and to reject with high probability those instances that cannot be clustered into k subsets of cost $(1+\epsilon)\,b$. Note that this distance measure is different from the one used in Chapter 4.

Definition 5.4.1 *A point set P is (b,k)-clusterable for a cost measure $\mathrm{cost}()$, if there is a partition of P into sets $S_1,\ldots S_k$ such that $\mathrm{cost}(S_i) \leq b$ for all $1 \leq i \leq k$. A point set P is ϵ-far from being (b,k)-clusterable, if for every partition of P into sets $S_1,\ldots S_k$ at least one set S_i has cost larger than $(1+\epsilon)\,b$.*

Let us assume, without loss of generality, that $b = 1$ and thus we want to design a tester for the problem whether a point set P is $(1,k)$-clusterable for the two cost measures above. We partition $\mathbb{R}^d$ into grid cells of side length $\epsilon/(3\sqrt{d})$. For each cell containing an input point, we choose an arbitrary input point from the cell as its representative. Then, we compute whether the set of representatives is $(1,k)$-clusterable. If it is so, then we accept it, if it is not so, then we reject it. Clearly, every set of points that is $(1,k)$-clusterable is accepted by the algorithm. On the other hand, every instance that is ϵ-far from $(1,k)$-clusterable will be rejected. (This approach has been introduced in [2] to obtain a $(1+\epsilon)$-approximation algorithm for the radius clustering problem.)

Our algorithms starts with an empty box with vertices at infinity. Then we query for a point in this box. This point is located in a grid cell C. We will use the point as representative for this cell. The bounding hyperplanes of C induce a partition of the space in 3^d boxes (one of them being the grid cell). We mark all empty boxes and the grid cell. Then we continue this process with an unmarked box. If there are only marked boxes the algorithm terminates.

So far, our partition into grid cells works fine, only if there are many points in a single cell. On the other hand, if no two points are in the same grid cell, the algorithm has $\Omega(n)$ query complexity. Thus we need an upper bound on the number of representatives that are $(1,k)$-clusterable. Similar to Chapter 4 we use a volume argument to show that we can stop the process when we find a set of representatives of size $k \cdot (6\sqrt{d}/\epsilon)^d$.

Lemma 5.4.2 *Let S be a set of points in $\mathbb{R}^d$ no two points of which belonging to the same cell of a grid of size $\epsilon/(3\sqrt{d}) \leq 1$. If $\mathrm{cost}(S) \leq 1$ for any of the two cost measures described above, then $|S| \leq (6\sqrt{d}/\epsilon)^d$, where $\mathrm{cost}(\cdot) \in \{\mathrm{cost}_R, \mathrm{cost}_D\}$.*

Proof : We first observe that if $\mathrm{cost}(S) \leq 1$ then S is contained in a hypercube with side length 1. We show that a no set S of size more than $(6\sqrt{d}/\epsilon)^d$ that satisfies the precondition of Lemma 5.4.2 is contained in a hypercube with side length 1. A hypercube with side length 1 can contain at most $(\frac{1}{\epsilon/(3\sqrt{d})}+2)^d \leq (6\sqrt{d}/\epsilon)^d$ grid cells. This proves Lemma 5.4.2. □

Let $V = k \cdot (6\sqrt{d}/\epsilon)^d$ the maximum number of cells that can contain points that belong to one of the k clusters. We observe that we can stop our procedure if the number

of representatives is more than V. Thus, we can guarantee that the algorithm requires at most $V \cdot 3^d$ range queries.

Theorem 17 *There is an property tester for the radius clustering and diameter clustering problem that uses at most* $k \cdot (18\sqrt{d}/\epsilon)^d$ *orthogonal range queries.*

Proof : We first describe the algorithm:

```
ClusterTest(P, k, b, ε):
    B = infinite box
    S = ∅
    Initialize empty queue Q
    insert B into Q
    while Q is not empty do
        B = Q.head()
        if B contains a point p then
            find the grid cell C that contains p
            S = S ∪ {p}
            split B into 3^d sub-boxes using the bounding hyperplanes of C
            insert all sub-boxes (except for C) into Q
        if there are more than V cells with representatives then
            reject
    if S is (k, b)-clusterable then accept
    else reject
```

We charge to each point $p \in S$ the $3^d - 1$ range queries that have to be done to check the $3^d - 1$ new boxes (other than the grid cell containing p) that are created when p is selected to be the representative for a grid cell. Thus the overall number of range queries is at most $V \cdot 3^d = k \cdot (18\sqrt{d}/\epsilon)^d$.

Now we have to prove that our algorithm is a property tester. Clearly, if P is (k, b)-clusterable then so is every subset of P and so the algorithm accepts. If the algorithm rejects because there are too many cells with representatives then by Lemma 5.4.2 the input points set is not (k, b)-clusterable. Thus it remains to show that the algorithm rejects every point set that is ϵ-far from (k, b)-clusterable.

Let us assume that P is ϵ-far from (k, b)-clusterable. Then for every partition into sets $C_1, \ldots, C_k$ there is a set C_i with $\mathrm{cost}(C_i) > (1 + \epsilon)b$. By definition of the diameter clustering problem this means that there are two points $p, q \in C_i$ such that $\mathrm{dist}(x, y) > (1 + \epsilon)b$. Since the diagonal of a grid cell is at most $\epsilon b/3$ the distance between the representatives of the cells of p and q is at least $(1 + \epsilon/3)b$. Thus the set S cannot be clustered and the algorithm rejects.

Similar arguments show that every ϵ-far instance of the radius clustering problem is rejected. □

6 Property Testing and Moving Data

In this chapter we apply the concept of property testing in the context of moving data. We develop a theoretical model for the analysis of approximate combinatorial structures under a very general type of motion. Then we illustrate our model on some examples. Among these examples are range trees and the Euclidean minimum spanning tree of objects moving in the $\mathbb{R}^2$. We consider the following general scenario: We are given a set $\mathcal{O}$ of n objects and these objects are moving in the $\mathbb{R}^d$. We assume that the objects are moved by a third party. We have *no information about the future movement* of the objects. The only thing we are allowed to do is to update the current position of an object at certain points of time and at unit cost per update. Such an update is also called an *update query*. Our goal is to approximately maintain a combinatorial structure defined uniquely by the positions of these objects. At certain points of time there are queries to the combinatorial structure we maintain. Before we answer such a query we may update some (possibly all) object positions and perform some additional computations. After these updates we require that our structure (which we call *hypothesis*) is *close to correct* with probability at least $2/3$. Then the query is answered. The combinatorial structure we maintain together with the update scheme we use is called a *soft kinetic data structure*. The quality of a soft kinetic data structure is measured by a form of *competitive analysis*. For every possible (non degenerate) *continuous motion* of the objects we compare the number of updates made by our algorithm against the *dynamics of the motion*. The dynamics is measured in terms of *combinatorial changes* that occur if the combinatorial structure is correctly maintained for the complete motion. The worst case ratio (over all continuous motions) between the number of updates and the dynamics is called the *dynamic efficiency* of a soft kinetic data structure.

Our strategy to design a soft kinetic data structure is the following: We first design a property tester for the combinatorial structure we want to maintain. If this property tester rejects a hypothesis it should provide a witness that this hypothesis is not correct. Then we try to find a strategy that fixes such a witnessed error by applying some local changes in the structure.

Each time before we have to answer a query we run the property tester. If the property tester rejects, it provides a witness. The witnessed error is then corrected and we invoke the property tester again. In order to show that this process terminates we have to prove that every error correction brings the combinatorial structure closer to the correct structure. In order to analyze the dynamic efficiency of such a soft kinetic data structure we have to relate the number of corrections we do to the number of combinatorial changes induced by the underlying motion.

Our model is partly inspired by the framework of kinetic data structures by Basch et al. [12]. In their framework Basch et al. make different assumptions on the motion of the objects than we do. In particular, they assume that there is a *flight plan* that describes the near future motion of every object. This flight plan is known to the algorithm and changes in the flight plan are passed to the algorithm. The information provided by the flight plan results in a completely different model of computation. The analysis of a kinetic data structure is in terms of combinatorial changes of the maintained structured and inspired our way to analyze the dynamics.

6.1 Soft Kinetic Data Structures

In this section we describe the framework of soft kinetic data structures.

Moving Objects. We first introduce some terminology used to describe objects and their motion. We are given a set $\mathcal{O} = \{O_1, \ldots, O_n\}$ of n objects (e.g., points, balls, cubes) that move in the $\mathbb{R}^d$. The motion of the objects is described by a *motion plan*. A motion plan contains the position $p_i(t)$ of each object $O_i \in \mathcal{O}$ at every point of time $t \geq 0$. The entity of all positions of objects $P(t) = \{p_1(t), \ldots, p_n(t)\}$ is called the *configuration* of $\mathcal{O}$ at time t. With this definition a *motion plan* is a function that assigns a configuration to each point of time $t \in \mathbb{R}$, $t \geq 0$. We assume that the motion plan is fixed but unknown to what we later define as a *soft kinetic data structure*. The way we can access the positions of our objects is similar to property testing: If we fix a point of time t then the current configuration of the objects can be seen as a point set. We may then update the position of every point by specifying its index. This is similar to our representation of point sets used in the context of property testing. We may update the position of an object certain points of time.

We say that a motion plan is *continuous* if for each $t, \delta \in \mathbb{R}, t \geq 0$, there exists an ε_0 such that $|p_i(t) - p_i(t+\varepsilon)| \leq \delta$ holds for each $i, 1 \leq i \leq n$, and every $\varepsilon \leq \varepsilon_0$.

Hypotheses. In our model each of the n objects is represented by a unique *object identifier* which is a number from $[n]$. At every point of time t the soft kinetic data structure may update the current position $p_i(t)$ of object O_i using the object identifier i. We introduce object identifier to syntactically distinguish between objects and their current position. That is, we may, for example, compute a combinatorial structure defined over the objects without knowing their current positions. In our framework we are interested in combinatorial structures that are uniquely defined by the objects' positions. When the set of objects is a point set then some examples for such structures could be sorted sequences in 1D, Voronoi diagrams, or the Euclidean minimum spanning tree. We use the fact that we can build such a structure over a given set of objects even if we do not know the current object configuration. In such a case we make a guess about the position of each object and then we compute the structure. These guesses are usually based on the past configurations of the point set and on additional information obtained by update queries. The computed structure may or may not be consistent with the current object configuration.

Therefore, we call such a structure a *hypothesis*. Given a distance measure σ a hypothesis H is called ϵ-close to correct if the distance $\sigma(H, H^*)$ to the correct hypothesis is at most ϵ. Otherwise, we say that H is ϵ-far from correct.

Now we want to illustrate our definitions on the example of a point set in 1D and the combinatorial structure 'sorted sequence'. When we consider point sets then at every fixed point of time t the current configuration $P(t)$ is a point set represented essentially in the same way as in the context of property testing in the previous chapters. In particular, we have that the object identifiers are the numbers from $[n]$ and the current position of each point can be accessed by an update query for the value of $p_i(t)$. Since we consider a point set in one dimension we have a total ordering defined by the positions of the points. This total ordering corresponds to a permutation of $[n]$ which in turn can be viewed as a sorted sequence. Now assume we do not know the positions of the points at a fixed point of time t. Then we can still make some guesses (e.g. based on the past configurations we know) about the positions of the points and compute a permutation π of $[n]$. Such a permutation is a hypothesis for the sorted sequence defined by the configuration $P(t)$.

The Objective of Soft Kinetic Data Structures. The objective of a soft kinetic data structure is to maintain a hypothesis for a predetermined combinatorial structure under an unknown motion plan. At certain points of time the soft kinetic data structure has to answer a query about the hypothesis it maintains. Such a query is usually the standard query supported by the combinatorial structure (typically, some kind of access or search query). In our example of the sorted sequence a query may ask for the i-th item in the sorted sequence. Each time before a query is processed, an algorithm called the *reorganizer* is invoked. This algorithm may perform update queries about the current position of some objects (we assume that the time does not proceed as long as the reorganizer is working) and perform changes in the hypothesis. Technically, a reorganizer consists of a property tester for the combinatorial structure and a procedure that corrects all errors in the hypothesis that are detected by the property tester. After the reorganizer finished its work the hypothesis should be ϵ-close to correct with probability at least $2/3$. Then the *query algorithm* is invoked and answers the query.

We assume that the soft kinetic data structure has to process a sequence of m chronological queries $\langle Q \rangle = (Q_1, \ldots, Q_m)$. Query Q_i takes place at time t_i and we associate with each query the point of time when it takes place.

Soft Kinetic Data Structures. Now we can say that a soft kinetic data structure consists of

- a *hypothesis* for a predetermined combinatorial structure of a set of n points,
- a *reorganizer* that is invoked each time before a query to the hypothesis has to be processed,
- a *query algorithm* that processes the queries to a hypothesis.

6.1.1 Analysis of Soft Kinetic Data Structures

In this section we explain how to evaluate the quality of a soft kinetic data structure. Our concept is to apply competitive analysis. That is, for every sequence of queries and every continuous motion plan $\mathcal{M}$ we compare the number of update queries made by our soft kinetic data structure against the number of combinatorial changes that occur in a correct hypothesis when the points move according to $\mathcal{M}$.

Let $\mathcal{M}$ denote an arbitrary motion plan describing a continuous motion and let $\mathcal{CS}$ denote a combinatorial structure uniquely defined by a configuration of $\mathcal{O}$. The dynamics $dyn_{\mathcal{CS}}(\mathcal{M})$ of $\mathcal{M}$ w.r.t. $\mathcal{CS}$ is defined as the number of combinatorial changes in $\mathcal{CS}$ when we move the objects according to $\mathcal{M}$ from time 0 to ∞. The term combinatorial change depends on the problem at hand. For example, when the objects are points moving in the $\mathbb{R}$ and the combinatorial structure $\mathcal{CS}$ is a sorted sequence then a combinatorial change takes places at each point of time t when the ordering in the sorted sequence changes. When the objects are points and the structure is a Euclidean minimum spanning tree then a combinatorial change takes place when the graph structure changes. We always assume that the motion plan is non degenerate, that is, at a fixed point of time there can be at most one combinatorial change in the structure.

Now let us considered a given soft kinetic data structure $\mathcal{DS}$. For a given sequence $\langle Q\rangle$ of m queries $Q_1,\ldots,Q_m$ taking place at time $t_1,\ldots,t_i$ and a motion plan $\mathcal{M}$ we denote the *update cost* $cost(\langle Q\rangle,\mathcal{M})$ of $\mathcal{DS}$ to be the number of update queries done by the reorganizer when the objects move according to $\mathcal{M}$. The *dynamic efficiency* of a soft kinetic data structure is roughly the worst case ratio between the update cost and the dynamics.

Definition 6.1.1 *The* dynamic efficiency *deff of a soft kinetic data structure is defined as follows:*

$$deff := \sup_{\langle Q\rangle,\mathcal{M}} \frac{cost(\langle Q\rangle,\mathcal{M})}{dyn_{\mathcal{CS}}(\mathcal{M}) + m}$$

where the supremum is taking over all non-degenerate motion plans $\mathcal{M}$ describing a continuous motion and all possible sequences of m queries $\langle Q\rangle$.

A second quality measure for soft kinetic data structures is the *update time*. The update time is the worst case ratio between overall running time of the reorganizer and the number of update queries. We denote by $time(\langle Q\rangle,\mathcal{M})$ the overall running time used by the reorganizer for sequence $\langle Q\rangle$ under a motion plan $\mathcal{M}$.

Definition 6.1.2 *The* update time *of a soft kinetic data structure is defined as*

$$t_u := \sup_{\langle Q\rangle,\mathcal{M}} \frac{time(\langle Q\rangle,\mathcal{M})}{cost(\langle Q\rangle,\mathcal{M}) + m}$$

where the supremum is taking over all non-degenerate motion plans $\mathcal{M}$ describing a continuous motion and all possible sequences of m queries $\langle Q\rangle$.

6.1.2 Discussion of The Model

There are a few points in our model that require a further discussion. First of all, let us make some notes on the assumption that the points do not move while the reorganizer works. We made this assumption because we have no other restrictions on the motion. As soon as we allow the points to move during this process in a theoretical model there is no real chance to make reasonable statements about the quality of the structure. In an implementation the reorganizer is a continuous process typically running in the background. When a query arrives we stop the reorganizer and answer the query. Typically, the changes that occur during the time when the query is processed are neglectable.

A second point that requires discussion is the distance measure needed to evaluate the quality of a hypothesis. Unlike in the case of property testing we want to use our soft kinetic data structure to answer queries. Therefore, a distance measure should reflect the 'correctness of a hypothesis w.r.t. the query algorithm' rather than the pure combinatorial distance to a correct combinatorial structure. For example, if we consider soft kinetic search trees then a query could be an access operation to an item stored in the tree. In such a case a single error at the root of the tree can make almost every access operation fail. Although the combinatorial distance to a correct structure would be rather small in such a case, a hypothesis for a balanced search tree having such an error should be ϵ-far from correct. In general, we want to have a distance measure that guarantees that, typically, an access operation works correctly.

6.2 Basic Soft Kinetic Data Structures

In this section we describe how to apply our framework to some combinatorial structures like sorted sequences, balanced search trees, range trees and Euclidean minimum spanning trees.

6.2.1 Sorted Sequences

The first example for a soft kinetic data structure we consider is that of a sorted sequence. W.l.o.g., we assume that our set of objects $\mathcal{O}$ is a set of n points moving in $\mathbb{R}$. We assume that the total order induced by the positions of the objects at a fixed point of time is always unique. That is, we never have two objects with the same position. We can achieve this by using an arbitrary tie-breaker (e.g., the object number). The object identifiers of $\mathcal{O}$ are stored in an array $A[1 \dots n]$. When we consider continuous motion the topological structure of a sorted sequence changes when the positions v_1, v_2 of two points change from $v_1 < v_2$ to $v_1 > v_2$. We say that a permutation $A(t) = (A[1], \dots, A[n])$ of $[n]$ stored in A at time t is a hypothesis that is ϵ-close to correct, if the sequence $p_{A[1]}(t), \dots, p_{A[n]}(t)$ has edit distance (see Definition 6.2.1 for an equivalent definition) at most ϵn to a sorted sequence. We use the following definition:

Definition 6.2.1 *Let $p_{A[1]}(t), \dots, p_{A[n]}(t)$ denote a sequence stored in an array $A[1 \dots n]$ at time t. We say that $p_{A[1]}(t), \dots, p_{A[n]}(t)$ is ϵ-close to sorted, if the longest increasing*

subsequence in $p_{A[1]}(t), \ldots, p_{A[n]}(t)$ *has length at least* $(1-\epsilon)n$*. Otherwise, we say that* $p_{A[1]}(t), \ldots, p_{A[n]}(t)$ *is* ϵ*-far from sorted.*

Now we are given a sequence of m access queries $\langle Q \rangle = (Q_1, \ldots, Q_m)$ to our array and we assume that query Q_i takes place at time t_i. Before the query is processed the reorganizer is invoked. This algorithm checks whether the current hypothesis is ϵ-close to correct. In order to do so, it invokes a property tester SORTEDTEST for sorted sequences from [40]. This property tester checks, if a sequence of number $p_{A[1]}(t), \ldots, p_{A[n]}(t)$ is a sorted sequence or ϵ-far from sorted (where the distance is measured according to the edit distance).

If this algorithm rejects it returns two indices k, l with $k < l$ and $p_{A[k]}(t) > p_{A[l]}(t)$. We then use a binary search like procedure to find an index k' such that $p_{A[k']}(t) > p_{A[k'+1]}(t)$ and swap the object identifier stored in $A[k']$ and $A[k'+1]$. Then we make two recursive calls to the reorganizer:

ARRAYREORGANIZER(A, t, ϵ)
  if SORTEDTEST$\big((p_{A[1]}(t), \ldots, p_{A[n]}(t)), \epsilon\big)$ rejects **then**
    Let (k, l) be the pair returned by SORTEDTEST $\big((p_{A[1]}(t), \ldots, p_{A[n]}(t)), \epsilon\big)$
    k'=FINDINVERSION(A, k, l)
    swap$(A[k'], A[k'+1])$
    ARRAYREORGANIZER(A, t, ϵ)
    ARRAYREORGANIZER(A, t, ϵ)

From the pair of indices (k, l) returned by the property tester SORTEDTEST from [40] algorithm FINDINVERSION can compute a pair $(k', k'+1)$ with $p_{A[k']}(t) > p_{A[k'+1]}(t)$ by a binary search like procedure in the following way: As long $k \neq l-1$ we compute $m = \lceil \frac{k+l}{2} \rceil$ and proceed either with (k, m) or with (m, l) depending on the value of $p_{A[m]}(t)$.

We now want to prove that algorithm ARRAYREORGANIZER computes a hypothesis that is ϵ-close to correct.

Lemma 6.2.2 *Let* $A(t) = \big(A[1], \ldots, A[n]\big)$ *be* ϵ*-far from a correct hypothesis for a sorted sequence. Then algorithm* ARRAYREORGANIZER *computes with probability at least* $2/3$ *a hypothesis that is* ϵ*-close to correct.*

Proof : We start with a simple claim about the edit distance:

Claim 6.2.3 *Let* k' *denote an index such that* $p_{A[k']}(t) > p_{A[k'+1]}(t)$*. If we swap* $A[k']$ *and* $A[k'+1]$ *then the edit distance of* $p_{A[1]}(t), \ldots, p_{A[n]}(t)$ *to a sorted sequence does not increase.*

Proof : $p_{A[1]}(t), \ldots, p_{A[n]}(t)$ has edit distance k, if and only if the longest increasing subsequence of $p_{A[1]}(t), \ldots, p_{A[n]}(t)$ has length $n-k$. The length of the longest increasing subsequence cannot decrease when we swap $A[k']$ and $A[k'+1]$ and hence the edit distance cannot increase. □

Now we want to analyze the probability that ARRAYREORGANIZER stops when the hypothesis is ϵ-far from correct. We assume that the probability that SORTEDTEST accepts a sequence that is ϵ-far from sorted is less than $1/10$. If the probability is larger then we can use standard amplification arguments to get the desired probability. Now we observe that for the case that ARRAYREORGANIZER terminates when the input is ϵ-far from correct there is a witness in form of a binary tree. Each node of this tree corresponds to a call of ARRAYREORGANIZER that rejects the current hypothesis and it has a left (right) child if the first (second) recursive call rejects the hypothesis. For example, if the first call to ARRAYREORGANIZER accepts the hypothesis, then the corresponding tree is the empty tree and it appears with probability $\frac{1}{10}$. It is well known that the number of binary trees with k nodes is less than 4^k.

Now let us fix a binary tree T with k nodes and analyze the probability that the behavior of the algorithm is according to the witness T. We observe that the probability that ARRAYREORGANIZER behaves according to T is at most $(\frac{1}{10})^{k+1}$ because each binary tree with k nodes has $k+1$ empty leafs and each empty leaf corresponds to a call to the reorganizer that accepted a sequence that is ϵ-far from sorted. The overall probability that the process stops before the hypothesis is ϵ-close to correct is bounded from the above by

$$\sum_{0\leq i\leq\infty} 4^i(\frac{1}{10})^{i+1} \leq \frac{1}{10}\sum_{0\leq i\leq\infty}(\frac{2}{5})^i \leq \frac{1}{3}$$

This proves Lemma 6.2.2. □

Let $\mathcal{M}$ denote an arbitrary non-degenerate continuous motion plan. In the following we want to make some observations w.r.t. the combinatorial changes induced by the motion plan. First let us observe that any combinatorial change in the sorted sequence can be written as a reversal $\sigma\langle i, i+1\rangle$ for some $i \in [n-1]$ (unless we have a degenerate case where more than 2 points are at the same position). Such a reversal is called a *transposition*. Now let $\langle Q\rangle = (Q_1, \ldots, Q_m)$ denote a sequence of m queries and let us denote by t_i the time when query Q_i takes place and $t_0 = 0$. Now we derive a lower bound on the number of combinatorial changes that can be expressed in terms of the configurations of the point set at times t_i, $1 \leq i \leq m$. For this purpose let us introduce some notation. For a number π_i in a permutation $\pi = (\pi_1, \ldots, \pi_n)$ of $[n]$ let us define the *rank*(π_i) as the number of elements π_j with $\pi_j < \pi_i$ and $j > i$. Then we define the weight $w(\pi)$ of a permutation π as $\sum_{i\in[n]}$ *rank*(π_i). It is well known that we can write every permutation π as a sequence of $w(\pi)$ transpositions and that every sequence of transpositions equal to π has length at least $w(\pi)$.

We further observe that at every point of time t there is a unique permutation $\pi = (\pi_1, \ldots, \pi_n)$ such that $p_{\pi_1}(t) < \cdots < p_{\pi_n}(t)$. Thus we can write $\pi^i = (\pi^i_1, \ldots, \pi^i_n)$ to denote the permutation that satisfies $p_{\pi^i_1}(t_i) < \cdots < p_{\pi^i_n}(t_i)$, that is, π^i denotes the sorted sequence at the time when the i-th query takes place. Then we can define permutations $\sigma^1, \ldots \sigma^m$ by the equation:

$$\pi^i \cdot \sigma^i = \pi^{i+1} \ .$$

Now we can give a lower bound on the dynamics $dyn(\mathcal{M})$ of the motion plan $\mathcal{M}$ w.r.t. the combinatorial structure sorted sequence:

$$dyn(\mathcal{M}) \geq \sum_{1 \leq i \leq m} w(\sigma^i) \ .$$

We use this lower bound to show that the number of errors corrected by the reorganizer is at most the dynamics of the motion plan $\mathcal{M}$.

Lemma 6.2.4 *Let $\langle Q \rangle = (Q_1, \ldots, Q_m)$ be an arbitrary sequence of m queries and let $\mathcal{M}$ be an arbitrary non-degenerate continuous motion plan. If query Q_i takes place at time t_i and the reorganizer corrects $err(i)$ errors at time t_i, then we have:*

$$\sum_{1 \leq i \leq m} err(i) \leq dyn(\mathcal{M})$$

where $dyn(\mathcal{M})$ denotes the dynamics of the motion plan $\mathcal{M}$ w.r.t. the combinatorial structure sorted sequence.

Proof : For a given point of time t_i let $A(t_i) = (A[1], \ldots, A[n])$ denote the permutation defined by the elements of array A at time t_i. Further let us define the permutation $\sigma_C(t_i)$ in the following way:

$$A(t_i) \cdot \sigma_C(t_i) = \pi^i \ .$$

Hence the permutation $\sigma_C(t_i)$ denotes the changes that must be performed to transform the array at time t_i into a sorted sequence. The weight $w(\sigma_C(t_i))$ denotes the minimum number of transpositions necessary to transform the array into a sorted sequence. We prove by induction on the number of queries that for $k \leq m$ the following inequality holds:

$$\sum_{1 \leq i \leq k} err(i) + w(\sigma_C(t_k)) \leq \sum_{1 \leq i \leq k} w(\sigma^i) \ .$$

Since every permutation can be expressed as a sequence of transpositions we know that we can apply the triangle inequality to the weight of permutations. This way, we observe that the number of errors corrected in the $(k+1)$-st call of the reorganizer is at most

$$err(k+1) \leq w(\sigma_C(t_k) \cdot \sigma^{k+1}) - w(\sigma_C(t_{k+1})) \leq w(\sigma_C(t_k)) + w(\sigma^{k+1}) - w(\sigma_C(t_{k+1})) \ .$$

And hence

$$err(k+1) - w(\sigma_C(t_k)) + w(\sigma_C(t_{k+1})) \leq w(\sigma^{k+1}) \ .$$

This proves the induction hypothesis for every $k \leq m$ and we get

$$\sum_{1 \leq i \leq m} err(i) \leq \sum_{1 \leq i \leq m} err(i) + w(\sigma_C(t_m)) \leq \sum_{1 \leq i \leq m} w(\sigma^i) \leq dyn(\mathcal{M})$$

which completes the proof of Lemma 6.2.4. □

Now we are ready to prove:

Theorem 18 *There is a soft kinetic sorted array with dynamic efficiency $\mathcal{O}(\frac{\log n}{\epsilon})$ and update time $\mathcal{O}(1)$. It answers queries about the i-th object in the soft kinetic sorted array in time $\mathcal{O}(1)$.*

Proof : By Lemma 6.2.2 we know that our reorganizer satisfies the requirements of a soft kinetic data structure, that is, with probability at least $2/3$ the sorted sequence maintained in the array is ϵ-close to correct (after the reorganizer has been invoked). Thus we only have to analyze the dynamic efficiency and update time of the data structure. The property tester SORTEDTEST makes $\Theta(\log n/\epsilon)$ (update) queries and has the same running time. The procedure FINDINVERSION makes $\mathcal{O}(\log n)$ update queries and its running time is proportional to the number of performed update queries. For a sequence of queries $\langle Q \rangle = (Q_1, \ldots, Q_m)$ the number of calls to the reorganizer is $m+$'twice the number of errors corrected by the reorganizer'. Thus, together with Lemma 6.2.4 we get that the dynamic efficiency of the data structure is $\mathcal{O}(\log n/\epsilon)$. The update time and the query time of $\mathcal{O}(1)$ are immediate because for each update we do only a constant amount of work and the query algorithm just has to return one entry in the array. □

6.2.2 Balanced Search Trees

The next structure we want to look at is a binary search tree. A (balanced) binary search tree is a rooted tree. At each node v of the tree a key $\text{KEY}(v)$ is stored. For every node v in the tree and every node u in the left (right, respectively) subtree of v it holds that $\text{KEY}(v) \geq \text{KEY}(u)$ ($\text{KEY}(v) < \text{KEY}(u)$). In the case of soft kinetic data structures each node stores an object identifier. The key of a node is the position of the corresponding object (w.l.o.g., we assume that the objects are points). Again we assume that every configuration of points induces a unique ordering among the objects (that is, the case '=' does not occur). We want to consider standard access (search) queries in binary trees. At first glance the problem seems to be similar to the case of soft kinetic arrays. The combinatorial structure we use for the analysis is (essentially) a sorted sequence. Thus we have a combinatorial change in the search tree, if there is a combinatorial change in the sorted sequence. The difference lies in the distance function used for soft kinetic balanced search trees. In the case of binary trees the edit distance to a sorted sequence is not a good distance measure w.r.t. the functionality of the trees. If only a single object is not at its correct position in the sorted sequence and this object is stored at the root of the tree then it might be the case that a huge fraction of access operations fails. Therefore, we consider a different distance measure:

Definition 6.2.5 *A search tree is ϵ-far from correct, if more than $\epsilon\, n$ access operations fail.*

With this new distance measure we also need a new property tester for the invariant of the search tree. But this is simple for balanced search trees with the distance measure from above. We store all nodes of the search tree in an array such that we can pick a node uniformly at random in constant time. Then for $s \geq 2/\epsilon$ the following algorithm is a property tester for the invariant of a search tree T w.r.t. the distance measure from Definition 6.2.5:

```
TREETEST(T, s, ε)
    for i = 1 to s
        pick a node v of the search tree uniformly at random
        if node v cannot be accessed reject
    accept
```

The test whether a node can be accessed can be implemented by walking from node v to the root and checking the invariant at each node of the search path. We show that for $s \geq 2/\epsilon$ algorithm TREETEST is a property tester.

Lemma 6.2.6 *For $s \geq 2/\epsilon$ algorithm* TREETEST *is a property tester for correctness of balanced binary search trees. Its query complexity and running time is $\mathcal{O}(\log n/\epsilon)$.*

Proof : Clearly, the algorithm accepts every correct search tree. Thus we assume that the tree is ϵ-far from correct. Then the probability that we find a node that cannot be accessed is more than ϵ in each pass of the **for**-loop. Hence the overall probability that the algorithm reject is

$$\mathbf{Pr}\big[\text{Algorithm TREETEST rejects}\big] \geq 1-(1-\epsilon)^{2/\epsilon} \geq 1-e^{-2} \geq 2/3 \ .$$

The query complexity and running time follow immediately from the fact that the search tree is balanced. □

If an object in the search tree cannot be accessed there must be an error on the search path (the path from the object's node to the root). We conclude that algorithm TREETEST rejects, only if it finds two objects whose current position is inconsistent with the tree order. Thus we are basically in the same situation as in the case of the sorted arrays. We therefore maintain an array $A[1 \ldots n]$ where the items are stored in the ordering induced by the search tree (this can be the same array that is used for the sampling). Then we can proceed similarly to the sorted arrays. We use a binary search to find two adjacent array entries whose ordering is inconsistent with the corresponding object positions. Then we swap these entries in the array and we swap the entries in the corresponding nodes in the tree (these nodes must *not* be adjacent in the tree). Unfortunately, the change of the distance function makes the proof for the sorted arrays invalid for the search trees. This is because of the fact that reducing errors in the tree order by swapping two objects (in the way just explained) does not necessarily decrease the distance to a correct data structure in terms of the distance measure from Definition 6.2.5. In fact, in situations it can even increase this distance. We must apply a different technique. We therefore change the sample size of algorithm TREETEST in such a way that it accepts an input that is ϵ-far from correct with probability at most $\frac{1}{3n^2}$:

Lemma 6.2.7 *Let $s \geq 2\ln(3n)/\epsilon$. Then the probability that algorithm* TREETEST *accepts an input that is ϵ-far from correct is at most $\frac{1}{3n^2}$.*

Proof : For the case $s \geq 2\ln(3n)/\epsilon$ we get:

$$\mathbf{Pr}\big[\text{Algorithm TREETEST accepts}\big] \leq (1-\epsilon)^{2\ln(3n)/\epsilon} \leq \frac{1}{3n^2} \ .$$

□

Then we use the following reorganizer:

```
TREEREORGANIZER(T, t, ε)
    if TREETEST(T, 2 ln(3n)/ε, ε) rejects then
        Let (k, l) be the pair returned by TREETEST(T, 2 ln(3n)/ε, ε)
        k′=FINDINVERSION(A, k, l)
        swap(A[k′], A[k′ + 1])
        swap the corresponding tree nodes
        TREEREORGANIZER(T, t, ε)
```

Now we prove that the probability that the reorganizer terminates when the tree is ϵ-far from correct is at most $1/3$.

Lemma 6.2.8 *The probability that* TREEREORGANIZER *stops when the hypothesis* T *is* ϵ*-far from correct is at most* $1/3$*.*

Proof : Let A denote the array in which the object identifiers are stored with respect to the tree order. Let $A(t) = (A[1], \ldots, A[n])$ denote the permutation of $[n]$ induced by array A at time t. Further let $\pi = (\pi_1, \ldots, \pi_n)$ denote the permutation such that $p_{\pi_1}(t) < \cdots < p_{\pi_n}(t)$. Let

$$A(t) \cdot \sigma = \pi \ .$$

Then we have that $w(\sigma) \leq n^2 - 3$ by the definition of the weight of a permutation. We further observe that each call of the reorganizer decreases the weight of σ by 1. In the case that T is ϵ-far from correct we know that the reorganizer has to reject at most n^2 times. Let X denote the indicator random variable for the event that algorithm REORGANIZER does not stop when T is ϵ-far from correct. Then we have:

$$\mathbf{Pr}\big[X = 1\big] \geq (1 - \frac{1}{3n^2})^{n^2-3} \geq \frac{1}{\sqrt[3]{e}} \geq 2/3$$

since

$$\mathbf{Pr}\big[\text{Algorithm TREETEST accepts}\big] \leq \frac{1}{3n^2} \ .$$

□

Lemma 6.2.4 holds also for trees because it does only assume that errors are corrected by swapping two adjacent objects in the maintained sorted sequence (which is equivalent to the tree order and is maintained in array A). Therefore we can conclude:

Theorem 19 *There is a soft kinetic balanced search tree with dynamic efficiency* $\mathcal{O}(\log^2 n/\epsilon)$ *and update time* $\mathcal{O}(1)$*. It answers access queries in time* $\mathcal{O}(\log n)$*.*

Proof : By Lemma 6.2.8 we know that our reorganizer satisfies the requirements of a soft kinetic data structure. The number of update queries made by the property tester is again proportional to its running time of $\mathcal{O}(\log^2 n/\epsilon)$. For a sequence of queries $\langle Q\rangle = (Q_1, \ldots, Q_m)$ the number of calls to the reorganizer is $m+$'the number of errors corrected by the reorganizer'. Together with Lemma 6.2.4 we get that the dynamic efficiency of the data structure is $\mathcal{O}(\log^2 n/\epsilon)$. The update time of $\mathcal{O}(1)$ is immediate. The query algorithm needs time $\mathcal{O}(\log n)$ for an access query because the tree is balanced. □

6.2.3 Range Trees

We now want to focus on structures from computational geometry. We first consider range trees in one and two dimensions. Then we design soft kinetic Euclidean minimum spanning trees.

1D Range Trees. 1D range trees are balanced binary search trees (see [1] for a formal definition of range trees). The only difference to a standard search tree lies in the type of query. A query to a range tree specifies a query range $[x, y]$ and the answer should return all points within the interval $[x, y]$. The standard way to answer such queries is to search for x and y in the range tree and then to report all points on the search paths that are within the query range as well as all points between the two search paths. Obviously, we can apply the analysis of soft kinetic search trees to range trees. So the only question is what time is needed to answer a query. A query to a soft kinetic range tree works in a similar way as a standard query to a range tree. But in soft kinetic range trees it may be the case that a point whose position is not within the query interval is stored between the two search paths (because there is an error in the structure). In such a case we do not report this point. For each point between the two search paths that is not contained in the query interval we can find an error in the data structure. This way we can amortize the time needed to process those points against the dynamics of the system.

We can use soft kinetic data structures for binary search trees to obtain the following result (see [1] for a formal definition of range trees):

Theorem 20 *There is a soft kinetic 1D-range tree with dynamic efficiency* $\mathcal{O}(\log^2 n/\epsilon)$ *and update time* $\mathcal{O}(1)$. *The soft kinetic range trees supports 1D-range queries in* $\mathcal{O}(\log n + k)$ *time, where* k *is the number of reported points (a point is reported, only if it is within the query range). If* k^* *denotes the number of points in the query range (at the time the query is processed) then it holds with probability at least* $2/3$ *that* $k \geq k^* - \epsilon n$.

Proof : We have to show that $k \geq k^* - \epsilon n$ holds with probability $2/3$ and we have to prove the running time for the range queries. The other results follow from the binary search trees. First of all, we can assume that the data structure is ϵ-close to correct with probability $2/3$ when the query is processed. This follows from Lemma 6.2.8 because the reorganizer is called before a query is answered.

A range query for a one dimensional range tree is answered in the following standard way: we search for the right and the left end of the query interval and output all nodes

that are between the search paths. Before we report a point we check if it is contained inside the query interval. If it is, we output it. If it is not we have found a conflict with some node on the search paths. After all nodes inside the query interval have been reported we partition the set of points P_{err} whose position is not in the query interval but that are located between the two search paths into two sets. The first set $P_{<x}$ contains all points of P_{err} whose position is smaller than x. The second set $P_{>y}$ contains all points of P_{err} whose position is larger than y. We pick the larger set of these two and correct an error for each point in this set. W.l.o.g. let us assume that $P_{<x}$ is the larger set. In the following we identify the point with the node where it is stored. We observe that there are two different orders among the points. The standard '$<$'-relation and the tree order $\prec_T$. We observe that each point p in $P_{<x}$ is either stored in a right subtree of a node of the left search path or in a left subtree of a node of the right search path. These are the two cases we consider. Let us start with the case when p is stored in a right subtree of a point q on the left search path. Then we observe that q must be larger than x (because the search path took the left edge). Therefore, we have $p < q$ but $q \prec_T p$. This is a conflict. In the second case, the point p is stored in a left subtree of a point on the right search path. Then we know that the point q stored at the split node (the node where the two search paths split) lies within the search interval $[x, y]$. Since p is stored in a left subtree of the right path it must be larger than q. But this is not the case. Hence there is a conflict because $q \prec_T p$. For each point we use the standard procedure to find two points that are adjacent in the tree order and we swap them. Since we process the points in increasing order none of the conflicts of the remaining points is resolved (though the 'conflict partner of a point may be moved in the tree order).

The number of inversions corrected in this process is at least half the number of points in the search interval that are not reported. Therefore we can amortize the update queries and the time needed to deal with these points against the dynamics of the motion.

Now assume that less than $k^* - \epsilon n$ points are reported. Then there are more than ϵn nodes that cannot be accessed, because every access operation to one of the missing nodes will end either on the search paths or between them. But the nodes that can be accessed this way are the nodes reported by the query algorithm. Thus we have that the tree is ϵ-far which occurs with probability at most $1/3$. □

2D-Range Trees. In this section we describe and analyze 2-dimensional soft kinetic range trees (see, e.g., [1] for a formal definition). We consider a standard implementation of 2D-range trees. A 2D-range tree for a set of n points in $\mathbb{R}^2$ is a two level data structure. The first level consists of a binary search tree (a 1D-range tree) sorted according to the x-coordinate. The second level consists of n trees (one for each node in the first level tree) sorted according to the y-coordinates of the points. (see, e.g., [1] for more details).

Similar to the binary search trees we want to have a data structure that supports range queries in a reasonable way. That is, we want to make sure that, if a structure is ϵ-close to correct, it cannot happen that almost every range query fails.

We say that a point is accessible in a 2D range tree, if it can be accessed in the first level tree using its x-coordinate as key and it can be accessed in *all* second level trees on

the search path using its y-coordinate.

Definition 6.2.9 *A 2D range tree is ϵ-far from correct, if there are more than ϵn points that are not accessible.*

Similar to the binary search trees we can design a simple property tester. We sample a set of $\mathcal{O}(1/\epsilon)$ nodes and check whether they are accessible by following the parents pointers in the trees. We also use two arrays as auxiliary data structures. Array A_X is used to define the order of objects in the first level of the range tree and array A_Y defines the order in the second level. We maintain the invariant that at any time the order of the objects in both levels is according to the order of the corresponding array. A combinatorial change in a range tree takes places when the there is a change in the sorted sequence w.r.t. the x-coordinate or in the sorted sequence w.r.t. the y-coordinate.

When the property tester rejects, it provides a proof that one of these arrays is not in the correct order and we can find an adjacent pair that is in the wrong order (using the same procedure as in the case of arrays and trees). Then we swap these two objects in the corresponding array and we adjust the range tree to the changes we made. The adjustment in the second level is done by deleting and reinserting the objects. This can be done in $\mathcal{O}(\log^2 n)$ time.

If we swap two objects in the first level tree, we delete all occurrences in the second level trees. For this purpose we maintain pointers to all occurrences of an object in the second level trees. Then we swap the two nodes in the first level tree and reinsert the objects into the second level trees (using the ordering maintained in array A_Y for comparisons). This procedure can be done in $\mathcal{O}(\log^2 n)$ time.

Now we can proceed in a similar way as for the search trees. We first show that the following algorithm is a property tester for correctness of range trees that accepts a range tree with probability at most $\frac{1}{6n^2}$ if it is ϵ-far from correct.

```
RANGETREETEST(T, s, ε)
    for i = 1 to s
        pick a node v of the search tree uniformly at random
        if node v is not accessible then reject
    accept
```

Lemma 6.2.10 *Let $s \geq 2\ln(6n)/\epsilon$. Then algorithm* RANGETREETEST *is a property tester. If an input tree T is ϵ-far from correct then it is accepted with probability at most $\frac{1}{6n^2}$.*

Proof : Since algorithm RANGETREETEST accepts every correct tree we only have to analyse the probability of rejectance. Let T be a range tree that is ϵ-far from correct. Then we have

$$\mathbf{Pr}\big[\text{algorithm RANGETREETEST accepts}\big] \leq (1-\epsilon)^{2\ln(6n)/\epsilon} \leq \frac{1}{6n^2}$$

□

We already explained the way the errors are corrected by the reorganizer. The remaining part of the reorganizer is similar to the one for search trees.

RANGETREEREORGANIZER(T, t, ϵ)
  if RANGETREETEST($T, 2\ln(3n)/\epsilon, \epsilon$) rejects **then**
    correct the detected error
    RANGETREEREORGANIZER(T, t, ϵ)

Lemma 6.2.11 *The probability that* RANGETREEREORGANIZER *stops when the hypothesis* T *is* ϵ*-far from correct is at most* $1/3$.

Proof : The distance between the hypothesis and a correct range tree is at most two times the distance in the one dimensional case (because we have two sorted sequences instead of one). In the case that T is ϵ-far from correct we know that the reorganizer has to reject at most $2(n^2-3)$ times. Let X denote the indicator random variable for the event that algorithm REORGANIZER does not stop when T is ϵ-far from correct. Then we have:

$$\mathbf{Pr}\big[X=1\big] \;\geq\; (1-\frac{1}{6n^2})^{2(n^2-3)} \geq \frac{1}{\sqrt[3]{e}} \geq 2/3$$

since

$$\mathbf{Pr}\big[\text{Algorithm RANGETREETEST accepts}\big] \leq \frac{1}{6n^2}$$

□

Checking whether a point is accessible takes $\mathcal{O}(\log^2 n)$ time. We conclude:

Theorem 21 *There is a soft kinetic 2D-range trees with dynamic efficiency* $\mathcal{O}(\log^3 n/\epsilon)$ *and update time* $\mathcal{O}(1)$. *The soft kinetic range tree supports 2D-range queries in* $\mathcal{O}(\log^2 n + k)$ *time where* k *is the number of reported points in the given query range. If* k^* *denotes the number of points in the given query range then it holds with probability* $2/3$ *than* $k \geq k^* - \epsilon n$.

Proof : The analysis of the running time of the queries is similar to the 1D case. And again, if there is a query that returns less than $k^* - \epsilon\, n$ points, then the structure must be ϵ-far which is a contradiction. □

6.2.4 Euclidean Minimum Spanning Trees

The last structure we consider is a Euclidean minimum spanning tree of a point set in the $\mathbb{R}^2$. In order to design a soft kinetic Euclidean minimum spanning tree we use the property tester and the distance measure from Section 3.3. Recall that this property tester only rejects, if it finds a cycle in the EMST-completion of the input graph. The longest edge of such a cycle cannot be in the EMST of the underlying point set. We use the following reorganizer:

```
EMSTREORGANIZER(T, t, ε)
    if EMSTTESTER (T, ε) rejects then
        Let e denote the edge that is not in the EMST
        mark e as deleted
        increase counter for deleted edges by 1
        if εn/200 edges have been deleted then
            recompute the EMST from scratch (query all point positions)
            set counter to 0
        EMSTREORGANIZER(T, t, ε)
```

Here EMSTTESTER(T, ϵ) denotes the property tester for EMSTs in degree bounded graphs from Section 3.3 with the exception that we do not test the connectivity of the tree. In the algorithm EMSTTESTER marked edges will be treated as missing edges. Since the number of marked edges is at most $\epsilon n/200$ our graph is always $\epsilon/200$-close to connected. By the analysis from Section 3.3 it follows that algorithm EMSTTESTER rejects the hypothesis with probability at least $2/3$, if it is ϵ-far from EMST.

Theorem 22 *There is a soft kinetic Euclidean minimum spanning tree with dynamic efficiency $\mathcal{O}(\sqrt{n/\epsilon} \cdot \log(n/\epsilon))$ and update time $\mathcal{O}(\log n/\epsilon)$. Queries about incident edges to a vertex v can be answered in time $\mathcal{O}(1)$.*

Proof : We first observe that the number of calls to the reorganizer is at most $m+$ 'number of detected errors'. If we found $\epsilon n/200$ errors then we correct all of them at once by recomputing the EMST from the scratch. This requires n update queries and $\mathcal{O}(n \log n)$ time. We observe that for each of the $\epsilon n/200$ errors there must be a change in the combinatorial structure of the Euclidean minimum spanning tree. Therefore, we can amortize the number of update queries and the running time over the $\epsilon n/200$ errors. We obtain that the dynamic efficiency is dominated by the property tester and therefore $\mathcal{O}(\sqrt{n/\epsilon} \cdot \log(n/\epsilon))$. The update time is dominated by the time needed to recompute the EMST and therefore $\mathcal{O}(\log n/\epsilon)$. □

7 Conclusions

In this thesis we studied geometric problems in the context of property testing and we applied concepts from geometry to property testing. In Chapter 3 we designed property testing algorithms for specific problems including disjointness of geometric objects, convex position, and the Euclidean minimum spanning tree. Our algorithms have sublinear query complexity and for some of them we give matching lower bounds. We also believe that every property tester for the Euclidean minimum spanning tree in 2D has a query complexity of $\Omega(\sqrt{n})$ though we do not have a proof of that.

We think that property testing is an interesting concept, i.e., if applied to geometric problems. We hope that further research in this area will provide new insights into the structure of geometric properties. Besides the development of new property testing algorithms it could be also an interesting problem to use existing property testers in the design of sublinear algorithms for geometric problems. For example, in the recent sublinear approximation algorithm for the weight of a minimum spanning tree in degree bounded graphs by Chazelle, Rubinfeld, and Trevisan [25] a property tester for connectivity in graphs has been used as a subroutine.

In Chapter 4 we designed a framework for property testing algorithms with one-sided error. The achievement of our framework is a nice decoupling of the probabilistic and the combinatorial aspects of property testing. As a consequence of this, a proof using our framework is usually elegant and gives a clear view of the important combinatorial features of the problem. We applied our framework to analyze a couple of different property testers and we showed that a large class of graph properties can be tested efficiently, if and only if there is a proof using our framework. There are several open problems regarding our framework and we want to mention two of them here:

The first problem is to extend the framework to algorithms with 2-sided error. The question is here, if it is possible to formulate a framework for a 2-sided error model and if such a model is useful for the design and analysis of property testing algorithms. Another interesting problem is to prove in our framework that triangle-free graphs can be tested efficiently. Although it is known that triangle-free graphs can be tested efficiently, the only known proof uses Szemerédi's Regularity Lemma. A proof using other techniques might provide new insights into property testing and the structure of dense graphs.

Chapter 5 and 6 were devoted to the design of new models that involve property testing. In Chapter 5 we introduced the possibility of range queries to property testing and showed that the query complexity for some problems can be dramatically improved, if range queries are allowed. Very recently, in a related model a sublinear time approximation algorithm for the weight of the Euclidean minimum spanning tree of a point set in

the $\mathbb{R}^d$ has been developed [27]. Is it possible to find other sublinear time approximation algorithms in such a model ?

In Chapter 6 we designed a model (called soft kinetic data structures) to analyze the maintainance of combinatorial structures under a very general model of motion. We analyzed some basic soft kinetic structures. Besides the obvious problem to design other soft kinetic data structures it would be very interesting to further develop models for moving data. Keeping the results from Chapter 5 in mind an interesting direction of research is to find out which queries should be supported by mobile objects in order to design efficient algorithms for mobile data.

Bibliography

[1] P. Agarwal. Range searching. *Handbook of Discrete and Computational Geometry*, 1997.

[2] P. Agarwal and C. Procopiuc. Exact and approximation algorithms for clustering. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 658–667, 1998.

[3] N. Alon. Testing subgraphs in large graphs. In *Proccedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 434 – 441, 2001.

[4] N. Alon, S. Dar, M. Parnas, and D. Ron. Testing of clustering. In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 240–250, 2000.

[5] N. Alon, E. Fischer, M. Krivelevich, and M. Szegedy. Efficient testing of large graphs. *Combinatorica*, 20(4):451 – 476, 2000.

[6] N. Alon and M. Krivelevich. Testing k-colorability. *SIAM Journal on Discrete Mathematics*, 15:211 – 227, 2002.

[7] N. Alon, M. Krivelevich, I. Newman, and M. Szegedy. Regular languages are testable with a constant number of queries. *SIAM Journal on Computing*, 30(6):1842 – 1862, 2001.

[8] N. Alon and A. Shapira. Testing satisfiability. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2002.

[9] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and approximation. combinatorial optimization problems and their approximability properties*. Springer, New York, NY, 1998.

[10] V. Bafna and P. A. Pevzner. Genome rearrangements and sorting by reversals. *SIAM Journal on Computing*, 25(2):272 – 289, 1996.

[11] Z. Bar-Yossef, R. Kumar, and D. Sivakumar. Sampling algorithms: Lower bounds and applications. In *Proceedings of the 33th Annual ACM Symposium on Theory of Computing (STOC)*, pages 266–275, 2001.

[12] J. Basch, L. Guibas, and J. Hershberger. Data structures for mobile data. *Journal of Algorithms*, 31(1):1 – 28, 1999.

[13] T. Batu, E. Fischer, L. Fortnow, R. Kumar, R. Rubinfeld, and P. White. Testing random variables for independence and identity. In *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 442 – 451, 2001.

[14] T. Batu, L. Fortnow, R. Rubinfeld, W. Smith, and P. White. Testing closeness of distributions. In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 259 – 269, 2001.

[15] M. A. Bender and D. Ron. Testing acyclicity of directed graphs in sublinear time. In *Proceedings of the Annual Interrnational Colloquium on Automata, Languages and Programming (ICALP)*, pages 809–820, 2000.

[16] J. L. Bentley and T. A. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Trans. Comput.*, C-28:643 – 647, 1979.

[17] P. Berman, S. Hannenhall, and M. Karpinski. 1.375-approximation algorithm for sorting by reversals. *Electronic Colloquium on Computational Complexity, Technical Report Series*, 8(47), 2001.

[18] P. Berman and M. Karpinski. On some tighter inapproximability results, further improvements. In *Proceedings of the 26th Annual Interrnational Colloquium on Automata, Languages and Programming (ICALP)*, pages 200 – 209, 1999.

[19] A. Bogdanov, K. Obata, and L. Trevisan. A lower bound for testing 3-colorability in bounded-degree graphs. In *Proceedings of the 43th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 93–102, 2002.

[20] B. Bollobás. Hereditary properties of graphs: Asymptotic enumeration, global structure, and colouring. *Documenta Mathematica*, III:333 – 342, 1998.

[21] H. Buhrman, L. Fortnow, I. Newman, and H. Röhrig. Quantum property testing. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2003.

[22] A. Caprara. Sorting permutations by reversals and Eulerian cycle decompositions. *SIAM Journal on Discrete Mathematics*, 12(1):91 – 110, 1999.

[23] C. Carathéodory. Über den Variabilitätsbereich der Fourierschen Konstanten von positiven harmonischen Funktionen. *Rendiconto del Circolo Matematico di Palermo*, 32:193–217, 1911.

[24] T. M. Chan. Output-sensitive results on convex hulls, extreme points, and related problems. *Discrete & Computational Geometry*, 16(3):369–387, 1996.

[25] B. Chazelle, R. Rubinfeld, and L. Trevisan. Approximating the minimum spanning tree weight in sublinear time. In *Proceedings of the 28th Annual Interrnational Colloquium on Automata, Languages and Programming (ICALP)*, pages 190 – 200, 2001.

[26] D. Cohen-Or, Y. Chrysanthou, C. T. Silva, and F. Durand. A survey of visibility for walkthrough applications. to appear in: IEEE Transactions on Visualization & Computer Graphics.

[27] A. Czumaj, F. Ergün, L. Fortnow, A. Magen, I. Newman, R. Rubinfeld, and C. Sohler. Sublinear approximation of Euclidean minimum spanning tree. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 813–822, 2003.

[28] A. Czumaj and C. Scheideler. An algorithmic approach to the general Lovász local lemma with applications to scheduling and satisfiability problems. In *Proceedings of the 32th Annual ACM Symposium on Theory of Computing (STOC)*, pages 38 – 47, 2000.

[29] A. Czumaj and C. Scheideler. Coloring non-uniform hypergraphs: A new algorithmic approach to the general Lovász Local Lemma. *Random Structures and Algorithms*, 17(3 - 4):213–237, 2000.

[30] A. Czumaj and C. Sohler. Property testing with geometric queries. In *Proceedings of the 9th Annual European Symposium on Algorithms (ESA)*, pages 266 – 277, 2001.

[31] A. Czumaj and C. Sohler. Soft kinetic data structures. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 865 – 872, 2001.

[32] A. Czumaj and C. Sohler. Testing hypergraph coloring. In *Proceedings of the 28th Annual Interrnational Colloquium on Automata, Languages and Programming (ICALP)*, pages 493 – 505, 2001.

[33] A. Czumaj and C. Sohler. Abstract combinatorial programs and efficient property testers. In *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 83–92, 2002.

[34] A. Czumaj, C. Sohler, and M. Ziegler. Property testing in computational geometry. In *Proceedings of the 8th Annual European Symposium on Algorithms (ESA)*, pages 155 – 166, 2000.

[35] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag Berlin Heidelberg, 1997.

[36] S. Doddi, M. Marathe, A. Mirzaian, B. Moret, and B. Zhu. Map labeling and its generalizations. In *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 148 – 157, 1997.

[37] Y. Dodis, O. Goldreich, E. Lehman, S. Raskhodnikova, D. Ron, and A. Samorodnitsky. Improved testing algorithms for monotonicity. In *Proceedings of the 3rd Internation Workshop on Randomization and Approximation Techniques in Computer Science*, pages 97 – 108, 1999.

[38] L. Engebretsen and J. Holmerin. Clique is hard to approximate within $n^{1-o(1)}$. In *Proceedings of the 27th Annual Interrnational Colloquium on Automata, Languages and Programming (ICALP)*, pages 2 – 12, 2000.

[39] P. Erdős and L. Lovász. Problems and results on 3-chromatic hypergraphs and some related questions. *Infinite and Finite Sets (to Paul Erdős on his 60th birthday)*, II:609 – 627, 1975.

[40] F. Ergün, S. Kannan, S. R. Kumar, R. Rubinfeld, and M. Viswanathan. Spot-checkers. *J. Comput. Syst. Sci.*, 60:717–751, 2000.

[41] T. Feder and D. H. Greene. Optimal algorithms for approximate clustering. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing (STOC)*, pages 434 – 444, 1998.

[42] E. Fischer. The art of uniformed decisions. a primer to property testing. *Bulletin of the European Association for Theoretical Computer Science*, 75:97 – 126, 2001.

[43] E. Fischer, G. Kindler, D. Ron, S. Safra, and A. Samorodnitsky. Testing juntas. In *Proceedings of the 43th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 103–112, 2002.

[44] E. Fischer, E. Lehman, I. Newman, S. Rashkodnikova, R. Rubinfeld, and A. Samorodnitsky. Monotonicity testing over general poset domains. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC)*, 2002.

[45] E. Fischer and I. Newman. Testing of matrix properties. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 286 – 295, 2001.

[46] E. Fischer and I. Newman. Functions that have read-twice constant width branching programs are not necessarily testable. In *Proceedings of the 17th Conference on Computational Complexity*, 2002.

[47] M. Formann and F. Wagner. A packing problem with applications to lettering of maps. In *Proceedings of the 7th Annual ACM Symposium on Computational Geometry (SoCG)*, pages 281 – 288, 1991.

[48] M. R. Garey and D. S. Johnson. *Computers and intractability: A guide to the theory of NP-completeness*. Freeman, New York, NY, 1979.

[49] O. Goldreich. Combinatorial property testing (a survey). In *Proceedings of DIMACS Workshop in Randomization Methods in Algorithm Design*, pages 45 – 59, 1997.

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

[51] 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.

[52] O. Goldreich and D. Ron. Property testing in bounded degree graphs. In *Proceedings of the Annual ACM Symposium on Theory of Computing (STOC)*, pages 406–415, 1997.

[53] O. Goldreich and D. Ron. A sublinear bipartiteness tester for bounded degree graphs. *Combinatorica*, 19(3):335–373, 1999.

[54] O. Goldreich and L. Trevisan. Three theorems regarding testing graph properties. In *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 460–469, 2001.

[55] V. Guruswami, J. Håstad, and M. Sudan. Hardness of approximate hypergraph coloring. In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 149 – 158, 2000.

[56] T. Hagerub and C. Rub. A guided tour to chernoff bounds. *Information Processing Letters*, 33:305 – 308, 1989.

[57] D. S. Hochbaum. *Approximation algorithms for $\mathcal{NP}$-hard problems*. PWS Publishing Company, Boston, MA, 1996.

[58] H. H. III, M. Marathe, V. Radhakrishnan, D. R. S. Ravi, and R. Stears. Nc-approximation for np- and pspace-hard problems for geometric graphs. *Journal of Algorithms*, 26(2):238 – 274, 1998.

[59] J. D. Kececioglu and D. Sankoff. Exact and approximation algorithms for sorting by reversals, with application to genome rearrangement. *Algorithmica*, 13(1/2):180 – 210, 1995.

[60] S. Khot. Hardness results for approximate hypergraph coloring. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC)*, 2002.

[61] D. Kirkpatrick and J. Radke. A framework for computational morphology. *Computational Geometry*, pages 217 – 248, 1985.

[62] Y. Kohayakawa, B. Nagle, and V. Rödl. Efficient testing of hypergraphs. In *Proceedings of the Annual Interrnational Colloquium on Automata, Languages and Programming (ICALP)*, pages 1017–1028, 2002.

[63] M. Krivelevich and B. Sudakov. Approximate coloring of uniform hypergraphs. In *Proceedings of the 6th Annual European Symposium on Algorithms (ESA)*, pages 477 – 489, 1998.

[64] L. Lovász. Coverings and colorings of hypergraphs. In *Proceedings of the 4th Southeastern Conference on Combinatorics, Graph Theory and Computing*, pages 3 – 12, 1973.

[65] R. Motwani and P. Raghavan. *Randomized algorithms*. Cambridge University Press, New York, NY, 1995.

[66] I. Newman. Testing of function that have small width branching programs. In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 251 – 258, 2000.

[67] M. Parnas and D. Ron. Testing the diameter of graphs. In *Proceedings of the RANDOM-APPROX'99*, pages 85–96, 1999.

[68] M. Parnas and D. Ron. Testing metric properties. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 276 – 285, 2001.

[69] M. Parnas, D. Ron, and R. Rubinfeld. Testing parenthesis languages. In *Proceedings of the 5th International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 261 – 272, 2001.

[70] M. Parnas, D. Ron, and A. Samorodnitsky. Proclaiming dictators and juntas or testing boolean formulae. In *Proceedings of the 5th International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 273 – 284, 2001.

[71] P. A. Pevzner. *Computational Molecular Biology*. MIT Press, 2000.

[72] P. A. Pevzner and M. S. Waterman. Open combinatorial problems in computational molecular biology. In *Proceedings of the 3rd Israel Symposium on Theory of Computing and Systems*, pages 158 – 173, 1995.

[73] J. Radhakrishnan and A. Srinivasan. Improved bounds and algorithms for hypergraph two-coloring. In *Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 684 – 693, 1998.

[74] D. Ron. Property testing. In *Handbook of Randomized Algorithms*. Kluwer Academic Publishers, 2001.

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

[76] M. Sharir and E. Welzl. A combinatorial bound for linear programming and related problems. In *Proceedings of the 9th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 569 – 579, 1992.

[77] R. E. Tarjan. Data structures and network algorithms. In *CBMS-NSF Regional Conference Series in Applied Mathematics*, volume 44. Society for Industrial and Applied Mathematics (SIAM), 1983.

[78] E. Welzl. Smallest enclosing disks(balls and ellipsoids. In H. Maurer, editor, *New Results and New Trends in Computer Science*, number 555 in LNCS. Springer, 1991.