

# **On Geometric Measures and Their Computation**

**Jérôme Urhausen**

Typeset with L<sup>A</sup>T<sub>E</sub>X. Figures and cover drawn using Ipe, the extensible drawing editor, with the following exceptions: Figure 1.1 was created using the spiropLOTS app (<https://spiropLOTS.sites.uu.nl/>). The map in Figure 1.2 was graciously provided by the Savoie Departmental Orienteering Committee (Comité Départemental de Course d’Orientation de la Savoie) which is part of the French Federation of Orienteering (Fédération Française de Course d’Orientation). The 3D-printed object in the left of Figure 1.3 was printed using a Craftbot XL 3D printer graciously provided by ALTEN Netherlands, my new employer. The image in the right of Figure 1.3 is a screenshot of the Print 3D app of a .stl file created using Python.

Printing: Ridderprint, [www.ridderprint.nl](http://www.ridderprint.nl)

This work is licensed under CC BY-NC-ND 4.0. To view a copy of this license, visit <https://creativecommons.org/licenses/by-nc-nd/4.0/>

ISBN: 978-90-393-7591-4

# On Geometric Measures and Their Computation

Over Geometrische Maten en Hun Berekening  
(met een samenvatting in het Nederlands)

## Proefschrift

ter verkrijging van de graad van doctor aan de  
Universiteit Utrecht  
op gezag van de  
rector magnificus, prof.dr. H.R.B.M. Kummeling,  
ingevolge het besluit van het college voor promoties  
in het openbaar te verdedigen op  
maandag 27 november 2023 des middags te 12.15 uur

door

**Jérôme Eric Urhausen**

geboren op 11 september 1991  
te Eupen, België

Promotor: Prof. dr. Marc J. van Kreveld

Copromotoren: Dr. Maarten Löffler  
Dr. Frank Staals

Beoordelingscommissie: Prof. dr. M. de Berg  
Prof. dr. H. L. Bodlaender  
Dr. S. A. Chaplick  
Prof. dr. ir. A. C. Telea  
Prof. dr. C. Wenk

Dit proefschrift werd mogelijk gemaakt met financiële steun van NWO onder TOP-project nummer 612.001.651.





# Contents

<b>1</b>	<b>Introduction</b>	<b>7</b>
1.1	Preliminaries . . . . .	12
1.1.1	Notation for Basic Concepts . . . . .	12
1.1.2	Measures . . . . .	13
1.1.3	Additional Concepts . . . . .	16
1.2	Overview of the Thesis . . . . .	19
<b>2</b>	<b>Mapping Regions to the Grid with Bounded Hausdorff Distance</b>	<b>25</b>
2.1	Introduction . . . . .	25
2.2	Input Regions Are Points . . . . .	29
2.3	Input Regions Are Convex $\beta$ -Fat Regions . . . . .	30
2.4	Input Regions Are Convex Regions . . . . .	33
2.5	Input Regions Are General Regions . . . . .	38
2.5.1	Two Regions . . . . .	38
2.5.2	Three or More Regions . . . . .	40
2.6	Conclusion . . . . .	45
<b>3</b>	<b>Mapping Points to the Grid with Bounded Hausdorff Distance</b>	<b>47</b>
3.1	Introduction . . . . .	47
3.2	NP-Completeness . . . . .	50
3.3	Approximation Algorithms . . . . .	55
3.3.1	A First Attempt . . . . .	55
3.3.2	A Constant-Factor Approximation Algorithm . . . . .	57
3.3.3	An Algorithm with Constant Additive Error . . . . .	60
3.4	Conclusion . . . . .	62
<b>4</b>	<b>Querying the Hausdorff Distance of a Line Segment</b>	<b>65</b>
4.1	Introduction . . . . .	65
4.2	A Data Structure for Red Points . . . . .	67
4.2.1	Querying with a Line . . . . .	68
4.2.2	Querying with a Line Segment . . . . .	70

4.3	A Data Structure for Red Line Segments . . . . .	73
4.4	A Space-Time Tradeoff . . . . .	78
4.5	The Directed Hausdorff Distance from Red to Blue . . . . .	79
4.6	Conclusion . . . . .	80
<b>5</b>	<b>The <math>k</math>-Fréchet Distance</b>	<b>81</b>
5.1	Introduction . . . . .	81
5.2	Gaining Intuition: The Box Problem . . . . .	87
5.3	NP-Completeness for the Cover Distance . . . . .	91
5.3.1	Intuition . . . . .	93
5.3.2	Gadgets . . . . .	94
5.3.3	Reduction . . . . .	101
5.3.4	Proof of Correctness . . . . .	102
5.4	Algorithms for the Cover Variant . . . . .	103
5.4.1	Approximation Algorithm . . . . .	104
5.5	Computing the Cut Variant . . . . .	105
5.5.1	Cut Placements . . . . .	105
5.5.2	Algorithm . . . . .	106
5.6	Conclusion . . . . .	110
<b>6</b>	<b>Diverse Partitions of Colored Points</b>	<b>113</b>
6.1	Introduction . . . . .	113
6.2	Convex versus Voronoi Partitions in 1D . . . . .	116
6.3	NP-Hardness Proofs for the Voronoi Partition . . . . .	117
6.3.1	Richness as the Diversity Measure in 1D . . . . .	117
6.3.2	Richness as the Diversity Measure in 2D . . . . .	121
6.3.3	Shannon Index as the Diversity Measure in 1D . . . . .	124
6.4	Polynomial-Time Solution for Discrete Candidate Sites . . . . .	126
6.5	Approximation for Diverse Voronoi Partition in 1D . . . . .	128
6.6	Diverse Convex Partition is NP-Hard in 2D . . . . .	130
6.6.1	Construction . . . . .	131
6.6.2	Equivalence . . . . .	134
6.6.3	NP-Hardness of Maximum Independent Set in Orthogonal Segments . . . . .	136
6.7	Conclusion . . . . .	138
<b>7</b>	<b>Conclusion</b>	<b>139</b>
7.1	Summary . . . . .	139
7.2	Open Problems . . . . .	140



# Chapter 1

## Introduction

**Geometry.** The world is inherently geometric. We are three-dimensional beings in a three-dimensional world. We move along a two-dimensional surface and jump up into the third dimension, all the while moving through time along the fourth dimension. Our eyes produce two flat images that are combined so that we are able to detect depth. Seeing is geometry, as we constantly guess distances and match shapes. Geometry cannot just be defined, it can be visualized, it can be drawn. Geometry can be beautiful.

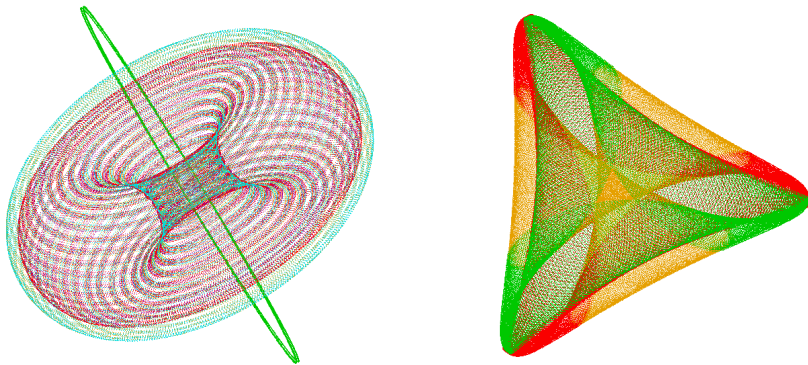


Figure 1.1: Two spiroplots [50].

This thesis defines and solves abstract computational problems in this geometric world. When working on a problem one does not look at the entire world, but restricts their view to a specific geometric space; this space then also defines the number of dimensions to consider. For example, a map is a drawing in two-dimensional space (see Figure 1.2) On the other hand, a hiker in the mountains moves around in three-

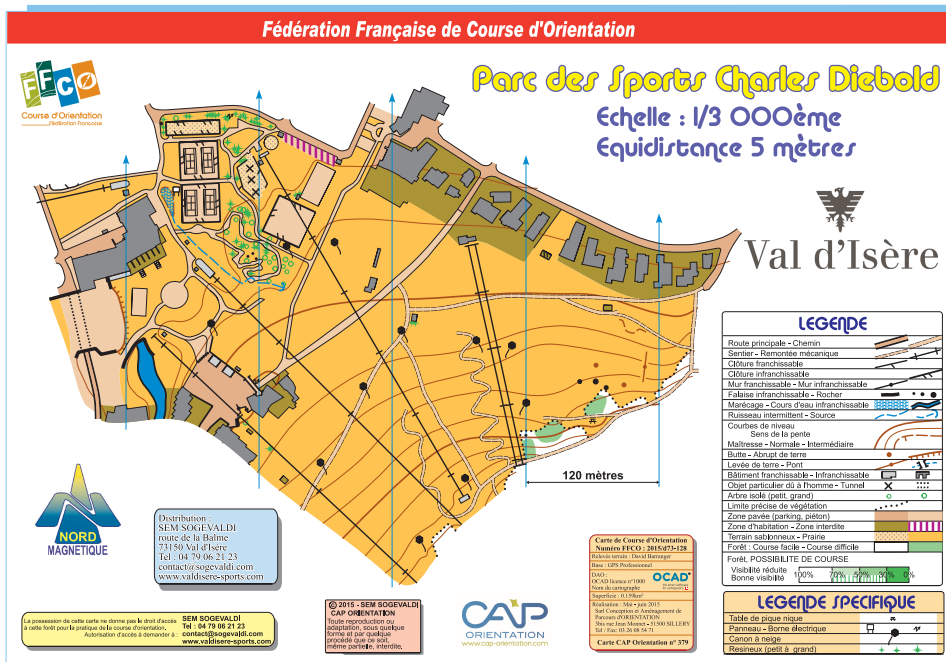


Figure 1.2: A map used in orienteering.<sup>1</sup> It depicts a hillside close to the village of Val d'Isère. The red curves are contour lines, the black line segments show ski lifts and the gray polygons represent buildings.

dimensional space. As long as there are no overhangs in this mountain range, when given the latitude and longitude of a hiker, we also know their elevation, in which case we talk of a 2.5-dimensional surface. As a third example we mention the inside of a 3D-printer, where we are clearly in a three-dimensional environment.

We concern ourselves with the objects in these spaces and in their properties. Continuing with the examples above, objects on a paper map are anything that you can draw with ink like points, lines, curves, circles, colored regions or labels. When hiking in the mountains, the interesting objects besides the surface of the mountain itself are the stones and boulders, the plants, the streams flowing down the mountain, the animals living there and the people walking around. Inside a 3D-printer we consider the robotic arm or arms that place the material, we consider the floor, the walls, the ceiling, and the platform. We are interested in the shape of the half-finished product and in the shape of the end product.

<sup>1</sup>The map was graciously provided by the Savoie Departmental Orienteering Committee (Comité Départemental de Course d'Orientation de la Savoie) with is part of the French Federation of Orienteering (Fédération Française de Course d'Orientation).

**Measures.** All of the above objects have multiple properties like their size, shape, color, momentum, or their position in space. If a characteristic can be quantified with a single number, it is a measure. We can measure the length of a path on a map, the average steepness of a hiking path, or the volume printed so far during a 3D-printing process.

However, instead of focusing our attention on just the features of one object at a time, we can also investigate the relations between two objects. We can measure how far apart two points on a map are, using for example the Euclidean distance or the Manhattan distance. The relation between the two objects can also be how similar they are. One can evaluate how closely a path drawn on a map reflects the real path it represents, or whether the finished 3D-printed product matches the design (see Figure 1.3). These relations can be quantified using distance or similarity measures. Note that distance measures can also measure similarity. The Hausdorff distance for instance measures similarity between regions or point sets by determining the distance to outliers.

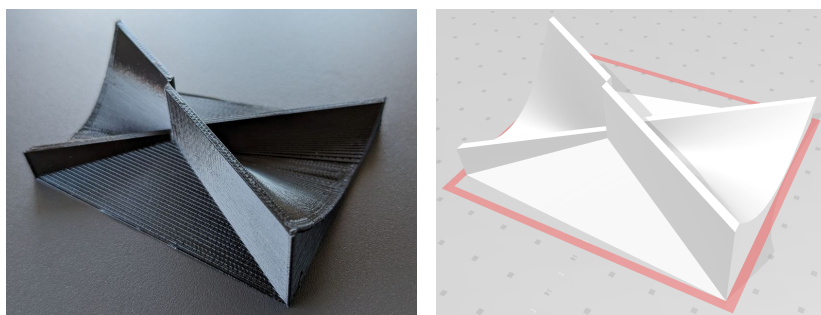


Figure 1.3: A 3D-printed object and the corresponding model. The curved surface in this sculpture plays a role in the data structure in Chapter 4.

Measures may quantify an aspect of one, two or multiple objects and the measured trait does not have to be of a geometric nature. One can measure how grid-like the roads of a city are. One can determine the diversity of the fauna or flora of a mountain range by counting the number of species. One can quantify how consistent a printing process is using a batch of products.

**Algorithms.** An algorithm is a sequence of basic operations. Algorithms are used to instruct machines which steps to take to produce a certain result (see Figure 1.4). Since nowadays most processes are automated, the art of developing efficient algorithms is more important than ever. Throughout this thesis, we approach different problems and aim to develop efficient algorithms that solve these problems.

But when is an algorithm considered to be *efficient*? There are multiple parameters that can be considered when defining the efficiency of an algorithm, like the execution

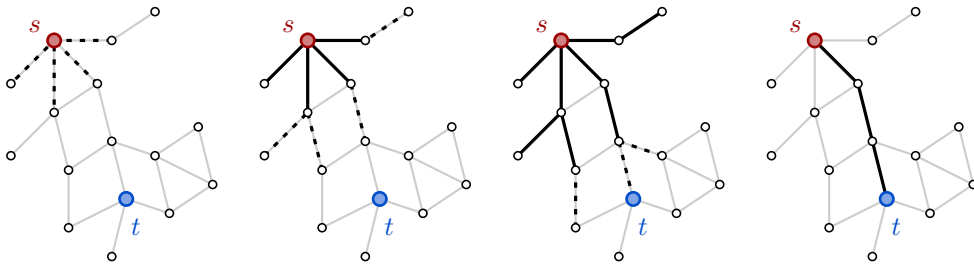


Figure 1.4: To find the shortest path from the red vertex  $s$  to the blue vertex  $t$  on this graph, we use the breadth-first search algorithm. This algorithm finds all vertices with distance 1 to  $s$ , then all vertices with distance 2 to  $s$ , and so on, until it finds  $t$ .

time, or the storage space needed. For algorithms that do not aim to find the optimal solution, but just try to approximate the result, the quality of the output is another parameter of interest. Similarly, for algorithms with random elements, an important aspect is the probability that the output is as desired. Some parameters like the storage space needed, or the time it takes to execute the algorithm generally change depending on the size of the input. Furthermore, the execution time depends on the machine that runs the algorithm. That is why, when displaying these properties, we focus on the scalability of these properties depending on the input size.

Another important aspect of the execution time of algorithms is that, in order to prove that a problem can be solved within certain time constraints, it suffices to showcase an algorithm that solves that problem provably within the given time. Note that finding such an algorithm may be far from trivial. On the flip side, proving that a given problem is *hard*, that is, proving that a given problem cannot be solved within a certain time span turns out to be more difficult in most cases. For that reason the concept of *complexity classes* was introduced. Depending on the complexity class, membership of a problem in a class can be shown by reduction, meaning one shows that an efficient algorithm for the investigated problem would imply an efficient algorithm for the other problems in that class. Thus, a proof that one of the problems in a class cannot be solved quicker than a certain bound implies a lower bound on the execution time of all problems in that class. Therefore, if one is faced with a hard problem, it is common that one attempts to show that it belongs to a well-known complexity class. A common example is the class of NP-complete problems; informally, it contains problems for which it is believed to be hard to find an efficient solution, but for which one can quickly check if a given solution is correct. An example of an NP-complete problem is the SUBSET SUM problem (see Figure 1.5): given a set  $A$  of integers and an integer  $b$ , we ask if one can select some of the integers in  $A$  such that they sum up to  $b$ . The problem is hard to solve, that is, it is hard to determine which integers to select. If on the other hand, one is given a supposed solution, that is, a selection of integers, it is easy to check whether they sum up to  $b$ . The problem

$$A = \{6, 6, 9, 13, 17, 22, 24, 27, 30\}, \quad b = 59$$

$$6 + 22 + 30 = 58 \quad \times$$

$$13 + 17 + 30 = 60 \quad \times$$

$$6 + 9 + 17 + 27 = 59 \quad \checkmark$$

Figure 1.5: An instance of the NP-complete problem SUBSET SUM, that is, a set of integers  $A$  and an integer  $b$ . The question is whether there is a subset of numbers in  $A$  that sum up to  $b$ .

SUBSET SUM is used in a reduction in Chapter 6.

In the field of algorithms one can classify problems depending on the setting. A problem is static if input is given in full before any computation is run. In the dynamic setting, the input is given incrementally, that is, an algorithm is required to not only calculate a result, but also to update that result when more input appears, respectively when the input changes. The query setting sits in the middle between the two: some data is given in advance and the input of the query is given later. For the query setting, algorithms are expected to precompute additional information on the data available at the start so that when queried, an answer can be found quickly. Data structures are helpful in all three settings but are a major part of algorithm design in the latter two. A data structure is simply a method of storing information that allows specific queries concerning the stored information to be evaluated efficiently. An example of precomputing data and putting it into a data structure is sorting books lexicographically onto a bookshelf. It allows finding a specific book faster compared to searching through books placed arbitrarily on the bookshelf. A data structure is dynamic if it allows the data it stores to be updated. The bookshelf can be considered a dynamic data structure if the books are not packed too tight. Then, new books can be added without much work while still keeping the lexicographic order intact.

**Computational Geometry.** Finally, the field of computational geometry combines geometry and algorithms, that is, a computational geometer investigates how to obtain an efficient solution to a geometric question using a set of basic operations. Going back to the examples used above, a question in computational geometry is how to find the shortest path between two points between obstacles (see Figure 1.6) or how to best place labels on maps to maximize readability. Given the description of a mountain range, one problem to approach would be to determine which slopes risk forming avalanches in winter. During the 3D-printing process, if the model to print has overhanging parts and if the printing method creates the structure in layers, one is concerned with placing supporting structures while minimizing printing overhead.

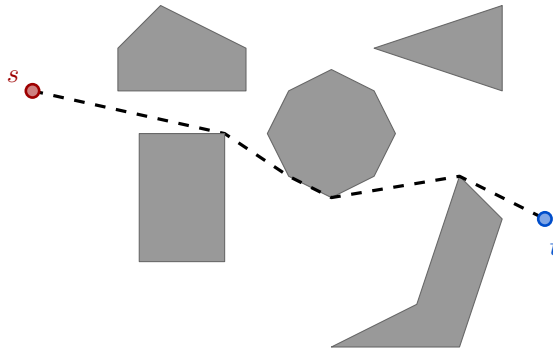


Figure 1.6: The shortest path from  $s$  to  $t$  in the plane among polygonal obstacles.

One way computers can store data is using a random-access memory, for short RAM. This type of memory allows to read and write data independent of the actual location of the data within the storage medium. Data is stored using a certain number of bits, which implies that numbers can only be stored up to a certain precision. In contrast, in the field of computational geometry, we generally use the real RAM model, that is, we assume the algorithm we develop runs on a hypothetical computer with the following features: we can store and access real numbers with infinite precision in constant time and perform arithmetic operations on them in constant time. In general, we are not allowed modulo or the floor operator in the real RAM model, as they can be exploited to create algorithms that solve hard problems faster than they should be possible to be solved. This model is used because it allows to focus on the geometric decision points, instead of the numerics of the situation. In practice, algorithms developed on the real RAM model do run on actual computers, although with restrictions. For a good overview of the real RAM model and the implication of its use, see [57].

This thesis combines the aspects listed above, that is, it treats geometric measures and how to define, calculate and use them.

## 1.1 Preliminaries

Here we define the notation that is used throughout the thesis. Additionally, we introduce and define core concepts.

### 1.1.1 Notation for Basic Concepts

We first define the notation for some basic concepts. For a set  $S$ , its *cardinality*  $|S|$  is the number of elements in  $S$ . For real numbers  $a \leq b \in \mathbb{R}$ , we write  $(a, b) = \{c \in \mathbb{R} \mid a < c < b\}$  for the *open interval* between  $a$  and  $b$ . Further, we write

$(a, b) = \{c \in \mathbb{R} \mid a < c \leq b\}$  and  $[a, b) = \{c \in \mathbb{R} \mid a \leq c < b\}$  for the *half-open intervals* and  $[a, b] = \{c \in \mathbb{R} \mid a \leq c \leq b\}$  for the *closed interval*.

Throughout this thesis, we work in the  $h$ -dimensional Euclidean space  $\mathbb{R}^h$  for  $h \in \{1, 2, 3\}$ . A *point*  $p \in \mathbb{R}^h$  is a  $h$ -dimensional vector  $p = (p_1, p_2, \dots, p_h)$ . We furthermore define  $p_x = p_1$ ,  $p_y = p_2$  and  $p_z = p_3$ . For a region  $S \subseteq \mathbb{R}^h$  defined by an equation  $E$ , we write  $S \equiv E$ ; for example for the two-dimensional line  $\ell$  with slope  $s$  and  $y$ -intercept  $t$ , we write  $\ell \equiv y = sx + t$ , that is,  $\ell = \{(x, y) \mid y = sx + t\}$ . The *Euclidean distance* between the points  $p \in \mathbb{R}^h$  and  $q \in \mathbb{R}^h$  is  $d(p, q) = \sqrt{\sum_{i=1}^h (p_i - q_i)^2}$ . For two regions  $A, B \subseteq \mathbb{R}^h$ , we denote by  $d(A, B) = \min_{a \in A, b \in B} d(a, b)$  the distance between their closest points. We define  $\overline{pq}$  as the (*closed*) *line segment* between the two points  $p, q \in \mathbb{R}^h$ . The length of  $\overline{pq}$  equals the distance  $d(p, q)$ . Lastly, a *curve* is a continuous function  $T : [0, 1] \rightarrow \mathbb{R}^h$ . We sometimes overload the term and also refer to the image of that function as the curve.

### 1.1.2 Measures

In this thesis, a measure is simply a function that returns a number. Depending on the nature of the measure, we require additional properties. For distance and similarity measures it can be helpful if the measure is a metric. Formally, a set  $S$  and a measure  $f : S \times S \rightarrow \mathbb{R}$  on that set form a *metric space* if the following holds:

$$\begin{aligned} \forall a, b \in S : f(a, b) = 0 &\iff a = b && \text{(separation)} \\ \forall a, b \in S : f(a, b) &\geq 0 && \text{(non-negativity)} \\ \forall a, b \in S : f(a, b) &= f(b, a) && \text{(symmetry)} \\ \forall a, b, c \in S : f(a, c) &\leq f(a, b) + f(b, c) && \text{(triangle inequality)} \end{aligned}$$

We call the measure  $f(\cdot, \cdot)$  in a metric space a *metric*.

Some similarity measures below are so-called bottleneck measures. Those measures, when comparing two objects, look at the most dissimilar parts between them (the bottleneck) and use those parts to quantify the similarity between the objects. We now define the measures relevant for this thesis.

The **Hausdorff distance** [76] between two regions  $A \subseteq \mathbb{R}^h$  and  $B \subseteq \mathbb{R}^h$  is

$$d_H(A, B) = \max \left\{ \overrightarrow{d}_H(A, B), \overleftarrow{d}_H(B, A) \right\}, \text{ with}$$

$$\overrightarrow{d}_H(A, B) = \sup_{a \in A} \inf_{b \in B} d(a, b).$$

The function  $\overrightarrow{d}_H(A, B)$  is the *directed* Hausdorff distance. The Hausdorff distance is determined by the point in one of the sets that is the farthest away from its closest point in the other set, see Figure 1.7. Thus, it is a bottleneck distance. The Hausdorff distance

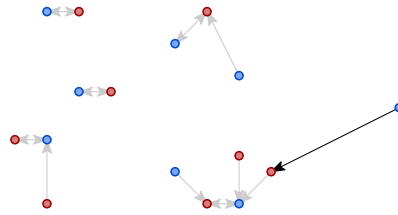


Figure 1.7: Two sets of points. For each red point its closest blue point is indicated and vice versa. Here, the Hausdorff distance between the two sets is determined by the distance of the blue point on the right to its closest red point.

satisfies the non-negativity, symmetry and triangle inequality conditions to be a metric. For compact sets the separation condition holds as well. The Hausdorff distance between two polygons of size  $n$  and  $m$  can be computed in  $O((n+m)\log(n+m))$  time [7]. The same holds for sets of  $n$  and  $m$  points or segments. On the other hand, computing the Hausdorff distance between two semi-algebraic sets is hard for a quite general complexity class ( $\forall\exists\mathbb{Z}\mathbb{R}$ -hard) which implies that the problem is both NP-hard and co-NP-hard (as well as  $\exists\mathbb{R}$ -hard and  $\forall\mathbb{R}$ -hard) [82]. The concept of NP-hardness is introduced in Subsection 1.1.3. The Hausdorff distance is widely used, for example in computer vision [105], or for detecting anomalies in trajectories [92].

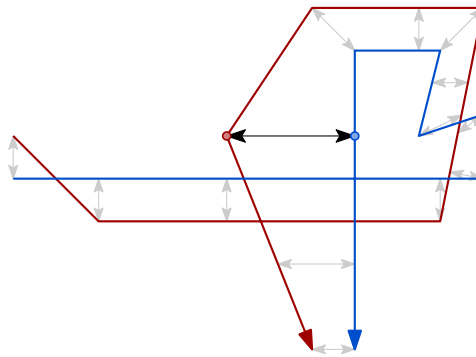


Figure 1.8: The Fréchet distance of two curves is calculated by traversing both curves simultaneously while staying within a certain distance indicated by the black arrow.

The *Fréchet distance* [61] is also a bottleneck similarity measure. The Fréchet distance between two curves  $P$  and  $Q$  is

$$d_F(P, Q) = \inf_{\sigma, \tau} \max_{t \in [0, 1]} d(P(\sigma(t)), Q(\tau(t))).$$

The reparametrizations  $\sigma, \tau : [0, 1] \rightarrow [0, 1]$  range over all orientation-preserving homeomorphisms. The Fréchet distance is a metric. The main difference compared



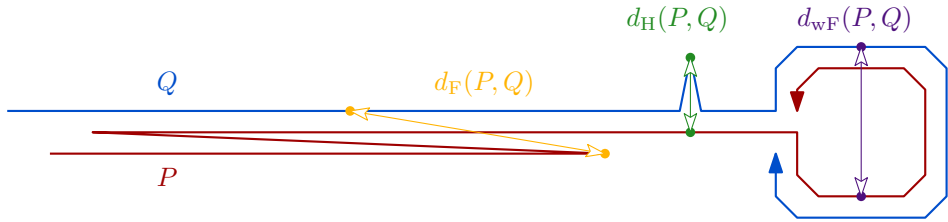


Figure 1.9: The difference between the Hausdorff distance and the (weak) Fréchet distance. The two bottleneck points that induce the Fréchet distance between the two curves are shown in yellow. For the Hausdorff distance they are green, and for the weak Fréchet distance the points are purple.

to the Hausdorff distance is that the Hausdorff distance works on sets whereas the Fréchet distance is only defined for curves, although it can be extended to surfaces. When comparing both distances on curves, we see that the Fréchet distance takes into account the order in which the curves are traversed (see Figure 1.8), which the Hausdorff distance does not. For polygonal chains  $P$  and  $Q$  of length  $n$  and  $m$ , the Fréchet distance can be calculated in  $O(nm \log(n+m))$  time [8]. The Fréchet distance has been used in a variety of applications, for example when analyzing geographic data, such as curves describing animal movement, or comparing chemical structures like protein chains or DNA [109].

The *weak Fréchet distance* [8] is a variation of the Fréchet distance that allows backtracking along the curves. Formally, the weak Fréchet distance between two curves  $P$  and  $Q$  is

$$d_{wF}(P, Q) = \inf_{\sigma, \tau} \max_{t \in [0, 1]} d(P(\sigma(t)), Q(\tau(t))).$$

Here, the reparametrizations  $\sigma, \tau : [0, 1] \rightarrow [0, 1]$  range over all continuous surjective functions. Note that contrary to [8], we do not require for endpoints to be matched in this thesis. The weak Fréchet distance satisfies the non-negativity, symmetry and triangle inequality conditions to be a metric but not the separation condition, that is, two distinct curves can have weak Fréchet distance 0. Note that for any two curves  $P$  and  $Q$ ,  $d_H(P, Q) \leq d_{wF}(P, Q) \leq d_F(P, Q)$  holds. Figure 1.9 shows curves where there is a significant difference between the Fréchet, the weak Fréchet and the Hausdorff distance.

The *species richness* [110] is a diversity measure that aims to determine the diversity of a set of labeled objects, like a set of animals of various species or a set of colored points. The species richness of a labeled set  $S$  is simply the number of different labels present in that set  $S$ , see Figure 1.10. This means the species richness of a labeled set  $S$  is  $|\{\ell \mid s \in S, \ell \text{ is the label of } s\}|$ .

The *Shannon index* [111] is also a diversity measure that, contrary to the species richness, takes the ratios of the labeled objects into account. Formally, the Shannon






Richness:	2	2	3	3	4
					
Shannon index:	$\log_2 3 - \frac{2}{3}$ $\simeq 0.9$	$1$ $= 1$	$\log_2 3$ $\simeq 1.6$	$\frac{3}{2}$ $= 1.5$	$\frac{5}{2} - \frac{3}{8} \log_2 3$ $\simeq 1.9$

Figure 1.10: For each depicted set its richness and its Shannon index are indicated.

index of a labeled set  $S$  is  $-\sum_{i=1}^r \rho_i \log_2 \rho_i$ , where  $r$  is the species richness of  $S$ ,  $\rho_i = |S_i|/|S|$  and  $S_i \subseteq S$  is the set of objects with label  $i$ . Note that since the ratio  $\rho_i$  is between 0 and 1, the term  $\log_2 \rho_i$  is negative. Therefore, we use a minus so that the Shannon index yields a positive number, as shown in Figure 1.10. Note that multiple bases for the Shannon index have been discussed. We chose 2 as this simplifies our calculations. Note that this diversity measure is strongly related to entropy, in a way that the Shannon index of a labeled set is the entropy of the probability distribution induced by the set.

### 1.1.3 Additional Concepts

**Big-O Notation.** An important aspect of an algorithm is its running time. It is common to indicate the running time of an algorithm in terms of the number of basic steps the algorithm needs to perform in the worst case, as a function of the input size. Furthermore, as the exact running time depends on the programming language used to implement the algorithm and the machine running the code, it is common to disregard constant factors in the running time. To that end, we define the *big O* notation [14]: for functions  $f, g : \mathbb{R} \rightarrow \mathbb{R}$ ,

$$f(n) \in O(g(n)) \iff \exists c \in \mathbb{R}^+, \exists n' \in \mathbb{R}, \forall n > n' : f(n) \leq c \cdot g(n);$$

$$f(n) \in \Omega(g(n)) \iff g(n) \in O(f(n));$$

$$\Theta(g(n)) = O(g(n)) \cap \Omega(g(n)).$$

Therefore, when we state that an algorithm has a running time of  $O(g(n))$ , we mean that our algorithm will produce a result using at most  $c \cdot g(n)$  basic operations in the worst case, where  $c$  is a constant. For more information, see Chapter 3 in [49].

**Computational Complexity.** We define the complexity classes  $P$ ,  $NP$  and  $NP$ -complete [48]. A problem is in  $P$  if there exists an algorithm that solves all instances of that problem with polynomial running time, that is, with running time in  $O(n^c)$ , where  $n$  is the size of the input and  $c$  is a constant independent of  $n$ . A problem is in  $NP$  if first, a

solution can be described using polynomial space and second, a potential solution can be tested in polynomial time. Note that  $P$  is contained in  $NP$ . The question whether there are problems in  $NP$  that are not in  $P$  is open. A problem is *NP-hard* if it is at least as hard as all other problems in  $NP$ . That is, a problem  $A$  is NP-hard if any problem  $B$  in  $NP$  can be reduced to  $A$  in polynomial time. The class NP-complete contains the NP-hard problems that are in  $NP$ . The existence of a polynomial time algorithm for any NP-hard problem would imply  $P=NP$ . Thus, proving that a problem is NP-complete, means that, unless one of the most impactful discoveries in the field of theoretical computer science is made, we cannot solve all instances of that problem in polynomial running time. More information can be found in Chapter 34 in [49].

**Approximation Algorithm.** Another important concept, especially in the context of NP-hard problems, is approximation. An optimization problem being NP-hard only means that finding an algorithm that produces an optimal solution is challenging. An approximation algorithm is an algorithm that guarantees to produce a result that is within a certain distance of an optimal solution. For example, a constant factor approximation algorithm guarantees a solution that is at most a constant factor worse than an optimal solution. For example, a 2-approximation algorithm for a shortest path problem produces a path that is at most twice as long as the shortest possible path.

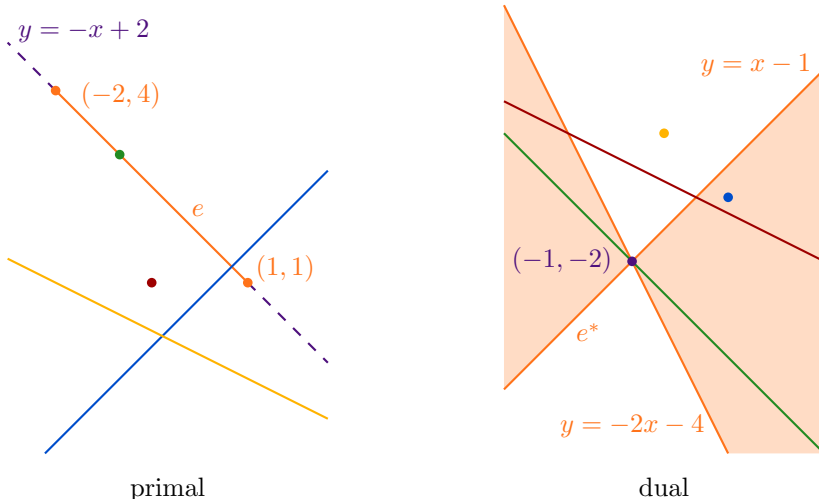


Figure 1.11: An orange segment, its purple supporting line and the corresponding dual double wedge. The blue line intersects the edge and therefore its dual point is inside the double wedge.

**Dual Transformation.** We define the dual transformation, a useful tool to look at problems from a different perspective [29]. For certain geometric objects  $o$ , we define their dual  $o^*$ . Let  $p = (p_x, p_y)$  be a point,  $\ell \equiv y = sx + t$  be a line and  $s = \overline{qw}$  be the line segment between the points  $q$  and  $w$ . The dual of the point  $p$  is the line  $p^* \equiv y = p_x x - p_y$ . The dual of the line  $\ell$  is the point  $\ell^* = (s, -t)$ . Note that this implies  $(v^*)^* = v$  for any point or line  $v$ . The duality transformation preserves incidences, that is, we have  $p \in \ell \iff \ell^* \in p^*$ . Finally,  $s^*$  is the double wedge between the two lines  $q^*$  and  $w^*$ , that is,  $s^*$  is the set of points below  $q^*$  or below  $w^*$ , but not below both. As incidence is preserved,  $s^*$  is the set of all points dual to lines intersecting  $s$ . Note that the dual transformation is not defined for vertical lines. Also, the dual of a vertical segment is the strip between two parallel lines. For a more in-depth explanation, see Chapter 8 in [18].

**Voronoi Diagrams.** Finally, we briefly explain Voronoi diagrams [116]. Intuitively, a Voronoi diagram is a subdivision of space using nearest neighbors, as shown in Figure 1.12. The Voronoi diagram of a set  $S$  of points in  $\mathbb{R}^2$  is a subdivision of the plane into cells with the following properties: each cell is associated with a point  $p \in S$  and the cell  $C_p$  associated with  $p \in S$  contains all the points whose nearest neighbor among  $S$  is  $p$ , that is,  $C_p = \{w \in \mathbb{R}^2 \mid d(w, p) < \min_{q \in S \setminus \{p\}} d(w, q)\}$ . In general, a Voronoi diagram of a set  $S$  of regions is a subdivision of the two-dimensional space  $\mathbb{R}^2$  into cells  $C_r$  associated with the regions  $r \in S$ , with  $C_r = \{w \in \mathbb{R}^2 \mid d(w, r) = \min_{q \in S \setminus \{r\}} d(w, q)\}$ . The boundary between two cells is called a *Voronoi edge*. When  $S$  is a set of points, the cells are convex and the Voronoi edge between two adjacent cells associated with the points  $p$  and  $q$  is a subsegment of the bisector between  $p$  and  $q$ . For more information, see Chapter 7 in [18].

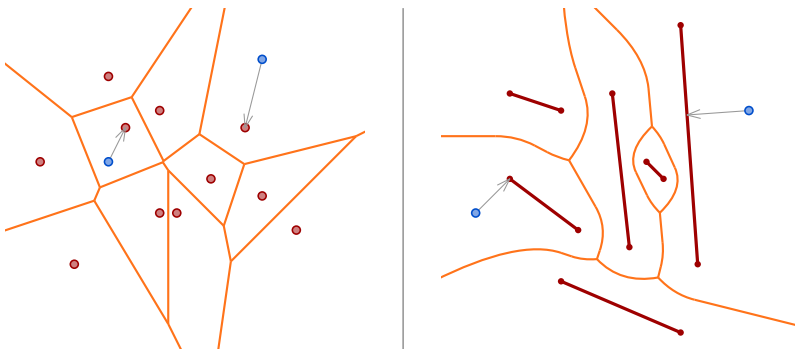


Figure 1.12: Left: A set of red points and the corresponding orange Voronoi diagram. Right: a set of red segments and the corresponding Voronoi diagram. For the blue points, the closest red element is shown.

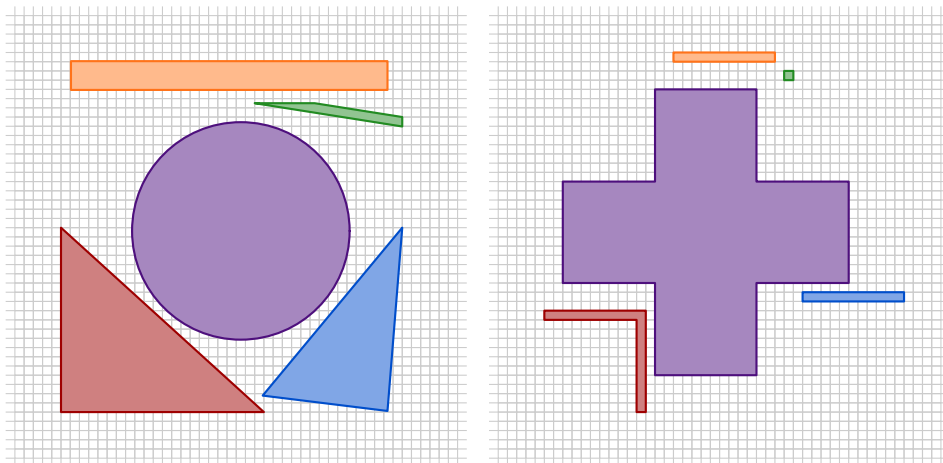
## 1.2 Overview of the Thesis

Below, I briefly introduce the chapters of the thesis, all focusing on different aspects concerning algorithms on geometric measures.

### Mapping Regions to the Grid with Bounded Hausdorff Distance

As described above, measures can be used to quantify the similarity of two geometric objects. One use of this is elaborated in Chapters 2 and 3. Here, we consider an abstraction of displaying geometric regions on a screen. More formally, we represent a set of objects given as a vector graphic using a raster image. The input is a vector graphic which is an analytical representation of geometric objects using points, segments, lines, circles and curves in general. A raster image is a discrete representation of geometric objects using a set of colored pixels on a grid. We use the Hausdorff distance to determine how well the pixelated version represents the input. In Chapter 2 the input is a set of disjoint and simply-connected regions and we investigate the influence of the shapes of the regions on how similar a pixelated version can be to the original. We prove asymptotically tight bounds for several classes of input regions.

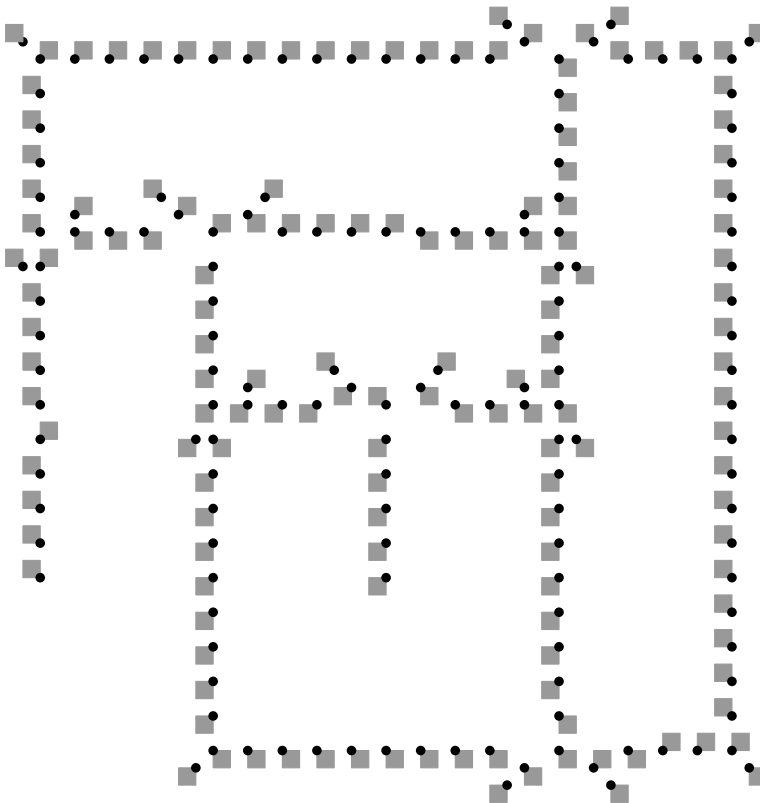
I. van der Hoog, M. van de Kerkhof, M. van Kreveld, M. Löffler, F. Staals, J. Urhausen and J. L. Vermeulen. Mapping Multiple Regions to the Grid with Bounded Hausdorff Distance. *17th Algorithms and Data Structures Symposium (WADS)*, pages 627–640, 2021.



### Mapping Points to the Grid with Bounded Hausdorff Distance

Again, we study a problem in digital geometry in Chapter 3: given a set of objects as a vector graphic, represent those objects using a raster image. However, instead of representing general regions using a set of grid cells, we focus on representing points using disjoint pixels on a grid, again with bounded Hausdorff distance. We explore algorithms that compute—or at least approximate—the pixels with the least distance to the respective points. We prove that optimizing the problem is NP-hard. Additionally, we present a constant factor approximation algorithm, as well as a slower algorithm with only a constant additive error.

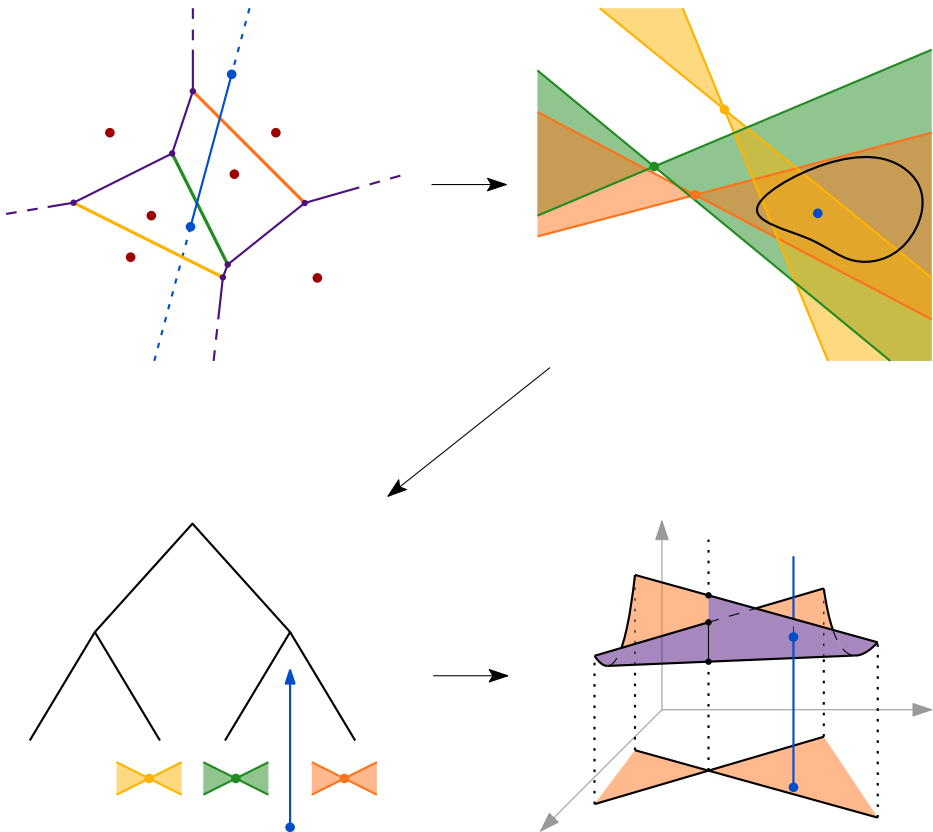
M. Löffler and J. Urhausen. Mapping Points to the Grid with Bounded Hausdorff Distance. *33rd Canadian Conference on Computational Geometry (CCCG)*, pages 47–55, 2021.



### Querying the Hausdorff Distance of a Line Segment

We then move on to determining the Hausdorff distance efficiently in a query setting: given one possibly large set and multiple smaller sets, we want to determine the Hausdorff distance between the large set and each individual smaller set one by one. In Chapter 4, we focus on line segments and thus attack the problem of preprocessing a set of line segments  $R$  with the aim to quickly determine the Hausdorff distance between  $R$  and a query line segment  $b$ . We present a data structure for this problem that allows a trade-off between the query time and the space required for storing the structure.

F. Staals, J. L. Vermeulen and J. Urhausen. Querying the Hausdorff Distance of a Line Segment. *Abstracts of the 38th European Workshop on Computational Geometry (EuroCG)*, pages 60:1–60:8, 2022.

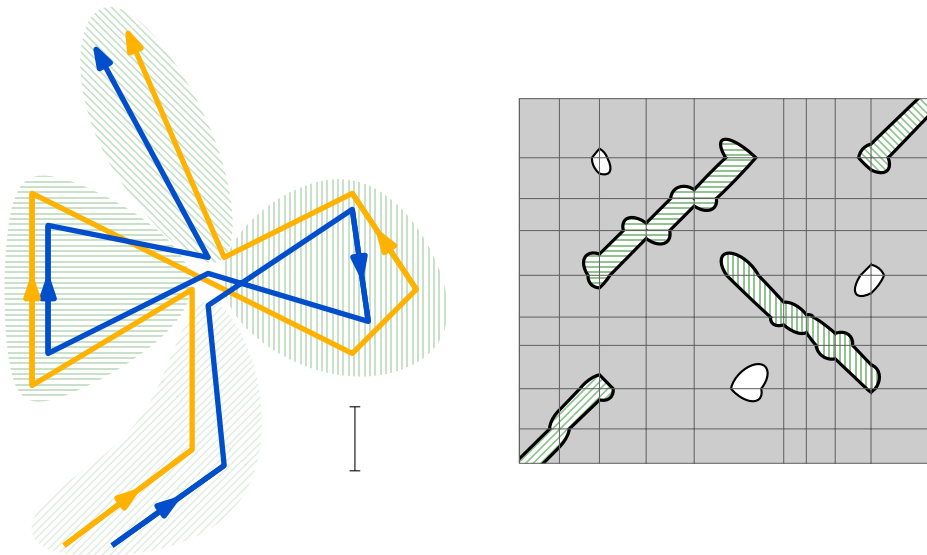


### The $k$ -Fréchet Distance: How to Walk your Dog while Teleporting

Next, in Chapter 5, we describe the  $k$ -Fréchet distance, a similarity measure that bridges between the Fréchet distance and the Hausdorff distance, by testing similarity between curves that resemble each other only piecewise. The parameter  $k$  denotes the number of subcurves into which we divide the input curves. The  $k$ -Fréchet distance allows two variants, the *cover* and the *cut* distance. The cut variant is not just NP-hard, but APX-hard [38]. We show that computing the cover variant of  $k$ -Fréchet is NP-hard, which is interesting since both the (weak) Fréchet and Hausdorff distance are computable in polynomial time. We then present two algorithms for the cover variant: a polynomial time 2-factor approximation and an exact algorithm with exponential running time. Finally, we present a polynomial time algorithm for the case  $k = 2$  in the cut variant.

H. A. Akitaya, M. Buchin, L. Ryvkin and J. Urhausen. The  $k$ -Fréchet Distance: How to Walk Your Dog While Teleporting. *30th International Symposium on Algorithms and Computation (ISAAC)*, pages 50:1–50:15, 2019.

M. Buchin, L. Ryvkin and J. Urhausen. Computing the Cut Distance of Two Curves. *Abstracts of the 36th European Workshop on Computational Geometry (EuroCG)*, 2020.

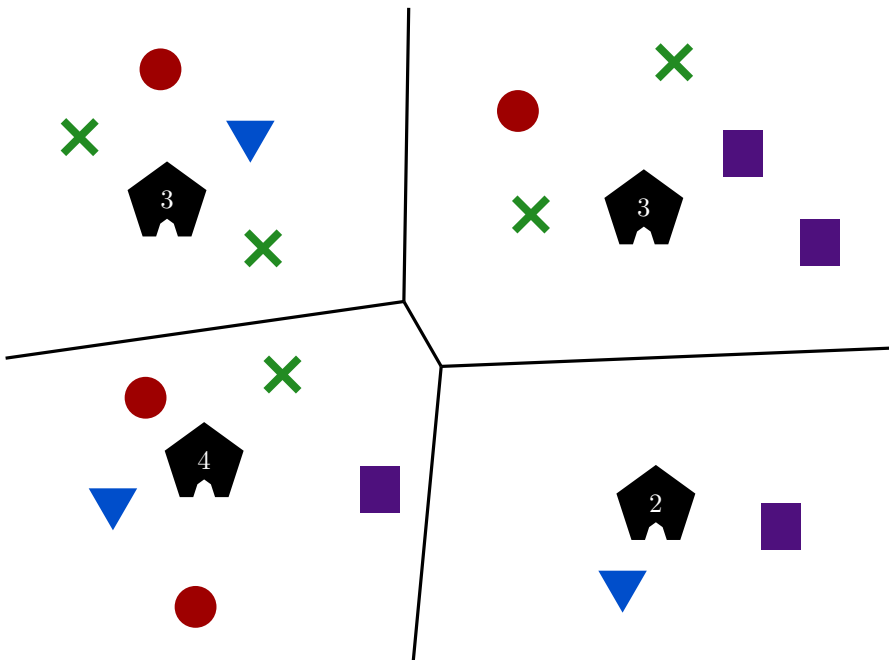




### Diverse Partitions of Colored Points

Instead of *similarity* we measure *diversity* in Chapter 6. We determine how well a set of colored points is mixed, that is, how well the colors are distributed. We approach this by partitioning space into well-shaped cells with the aim that overall the points within the cells form diverse sets, as measured using the species richness or the Shannon index. The diversity of a partition is the sum of the diversity scores of its cells. Note that this is the first spatial definition of diversity that does not use a predetermined subdivision. The algorithmic question asked in this chapter is, given a set of colored points, how to best partition this set into diverse and well-shaped cells. We study multiple versions of this problem in one and two dimensions. We measure the diversity of the cells using species richness, or the Shannon index. Furthermore, we consider partitioning the set using convex cells, or cells derived from a Voronoi diagram. Our results include hardness proofs and efficient algorithms.

M. van Kreveld, B. Speckmann and J. Urhausen. Diverse Partitions of Colored Points. *17th Algorithms and Data Structures Symposium (WADS)*, pages 641–654, 2021.





## Chapter 2

# Mapping Regions to the Grid with Bounded Hausdorff Distance

We study a problem motivated by digital geometry: given a set of disjoint geometric regions, assign each region  $R_i$  a set of grid cells  $P_i$ , so that  $P_i$  is connected, similar to  $R_i$ , and does not touch any grid cell assigned to another region. Similarity is measured using the Hausdorff distance. We analyze the achievable Hausdorff distance in terms of the number of input regions, and prove asymptotically tight bounds for several classes of input regions. This chapter is based on the following publication:

I. van der Hoog, M. van de Kerkhof, M. van Kreveld, M. Löffler, F. Staals, J. Urhausen and J. L. Vermeulen. Mapping Multiple Regions to the Grid with Bounded Hausdorff Distance. *17th Algorithms and Data Structures Symposium (WADS)*, pages 627–640, 2021.

### 2.1 Introduction

*Digital geometry* is concerned with the proper representation of geometric objects and their relationships using a grid of pixels. This greatly simplifies both representation and many operations, but the downside is that common properties of geometric objects no longer hold. For example, two lines in  $\mathbb{R}$  intersect in at most a single point, but it may be that two digitized lines intersect in multiple connected components. One objective of digital geometry is how to *consistently* digitize a set of geometric objects. Another objective is the presentation of vector objects with bounded error, using subsets of pixels.

Early results in digital geometry were mostly concerned with consistency and arose in computer vision. For a survey, see Klette and Rosenfeld [86, 87]. More recently, also error bounds under the Hausdorff distance have been studied. Chun et al. [46] investigate the problem of digitizing rays originating in the origin to digital rays such that certain properties are satisfied. They show that rays can be represented on the  $n \times n$  grid in a consistent manner with Hausdorff distance  $O(\log n)$ . This bound is tight in the worst case. By ignoring one of the consistency conditions, the distance bound improves to  $O(1)$ . Their research is extended by Christ et al. [44] to line segments (not necessarily starting in the origin), who obtain the logarithmic distance bound in this case as well. A possible extension to curved rays was developed by Chun et al. [45]. Other results with a digital geometry flavor within the algorithms community are those on snap rounding [19, 66, 77], integer hulls [10, 75], and discrete schematization [95].

In a recent paper, Bouts et al. [26] showed that any simple polygon, no matter how detailed, can be represented by a simply connected set of unit pixels such that the Hausdorff distance to and from the input is bounded by  $3\sqrt{2}/2$ .

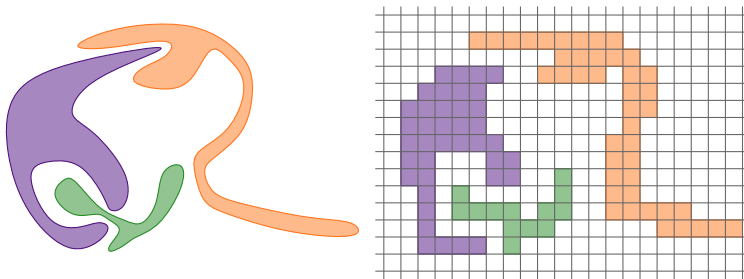


Figure 2.1: Three disjoint simply connected regions and a representation by simply connected sets of disjoint pixels.

### 2.1.1 Contribution

We extend the result from Bouts et al. [26] to multiple regions, see Figure 2.1. We investigate several restrictions on the class of regions and we show that stricter restrictions allow for pixel representations with a smaller Hausdorff distance. All our bounds are tight. We express our bounds in the number of input regions. Our results are shown in Table 2.1; they are fundamental results on the error that may be incurred when converting vector to grid representations, a common operation in computer graphics and GIS.

We do not make any assumptions on the resolution of the input. If the minimum distance between any pair of polygons is at least some constant (e.g.,  $4\sqrt{2}$  is enough), then we can realize a constant Hausdorff bound in all cases by applying the results

Table 2.1: Worst-case bounds on Hausdorff distances for  $m$  regions;  $\beta$  is constant.

region class	Points	Convex $\beta$ -fat	Convex	General Polygons and $m = 2$	General Polygons and $m \geq 3$
<b>Hausdorff distance</b>	$\Theta(\sqrt{m})$	$\Theta(\sqrt{m})$	$\Theta(m)$	$\Theta(1)$	unbounded

from Bouts et al. [26] separately on each polygon. We consider the case where no such assumptions are made.

### 2.1.2 Notation and Definitions

We denote by  $\Gamma$  the (infinite) unit grid, whose unit squares are referred to as *pixels*. We say two different pixels are *adjacent* if they share a vertex or edge. Recall that the *Hausdorff distance* between two sets  $A, B \subset \mathbb{R}^2$  is defined as  $d_H(A, B) = \max\{\sup_{a \in A} \inf_{b \in B} d(a, b), \sup_{b \in B} \inf_{a \in A} d(a, b)\}$ , where  $d(a, b)$  is the distance between the points  $a$  and  $b$ . Further we denote by  $d'_H(A, B) = \max\{d_H(A, B), d_H(\partial A, \partial B)\}$  the Hausdorff distance between the sets themselves and between their boundaries. See Figure 2.2 for an example where the distinction between  $d_H(\cdot, \cdot)$  and  $d'_H(\cdot, \cdot)$  is important.

Let  $\mathcal{R} = \{R_1, R_2, \dots, R_m\}$  be a set of  $m$  disjoint simply connected regions in the plane. In this chapter, we show how to assign a subset of the pixels  $P_i \subset \Gamma$  to each region  $R_i \in \mathcal{R}$ , such that the result is a set of  $m$  disjoint simply connected regions. Two such *grid polygons* are disjoint if they do not meet in any edge or vertex of the grid. A grid polygon is connected if its pixels are connected by edge adjacency, and simply connected if it is connected and its complement is also connected by edge adjacency. Hence, we do not allow vertex adjacency at all as it is ambiguous. We call the set  $\{P_1, P_2, \dots, P_m\}$  of such grid polygons a *valid assignment* for  $\mathcal{R}$ .

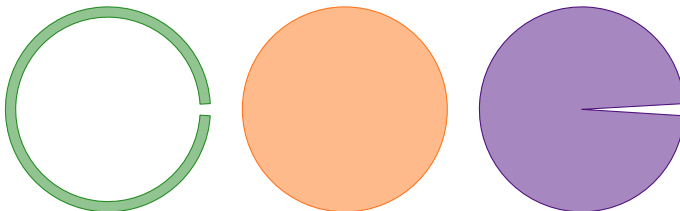


Figure 2.2: When we assume that their centers are aligned, the Hausdorff distance between the green and red regions is large while the Hausdorff distance between their boundaries is small. The inverse is true for the red and purple regions.

### 2.1.3 Overview

We are interested in finding for any set of regions  $\mathcal{R}$  a valid assignment such that for all  $i \in \{1, \dots, m\}$  the Hausdorff distance between  $R_i$  and  $P_i$  is at most  $h$ , and the Hausdorff distance between their boundaries is also at most  $h$ . In general, a worst-case bound on  $h$  will be a function of  $m$ . We study this problem under several restrictions on  $\mathcal{R}$ ; refer to Table 2.1. For each class of restrictions, first we show that there is a set of regions in that class for which any valid assignment contains at least one region  $R_i$  with a grid polygon  $P_i$  where  $d'_H(R_i, P_i) = \Omega(h)$ . Second we show that for any set of regions in that class, we can find a valid assignment such that for all regions  $R_i \in \mathcal{R}$  with corresponding grid polygon  $P_i$ , we have  $d'_H(R_i, P_i) = O(h)$ . Hence, our bounds are asymptotically tight.

We may interpret a solution to our problem as a *coloring* of  $\Gamma$ : each pixel  $q \in \Gamma$  is assigned one color in  $C = \{c_1, \dots, c_m\} \cup \{b\}$ , where  $c_i$  is the color of the input region  $R_i$  and  $b$  is the background color.

Our upper bound constructions all use similar concepts. Let  $\Gamma_k$  be a coarsening of the grid  $\Gamma$  whose cells have  $k \times k$  pixels, see Figure 2.3. We call these cells *superpixels*. For any superpixel  $S \in \Gamma_k$ , we denote by  $S[x, y]$  the pixel that is the  $(2x)^{\text{th}}$  from the left and  $(2y)^{\text{th}}$  from the bottom within  $S$ . We will determine for each region from  $\mathcal{R}$  which superpixels it contains and which ones it properly intersects, that is, for which it intersects the interior of the superpixel. If a region  $R_i$  contains a superpixel, then all pixels of  $\Gamma$  in that superpixel will be part of  $P_i$ . If  $R_i$  properly intersects a superpixel, we ensure that at least one, but not all pixels in that superpixel will be part of  $P_i$ . A superpixel not intersecting  $R_i$  will have no pixels in  $P_i$ . The main challenge is then finding a scheme by which each grid polygon becomes simply connected yet all remain disjoint. It is then relatively straightforward to see that  $d'_H(R_i, P_i) \leq k\sqrt{2}$ .

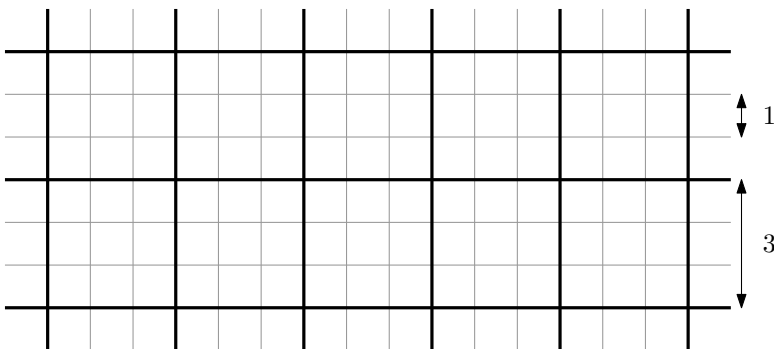


Figure 2.3: A coarsening  $\Gamma_3$  of the grid  $\Gamma$ . Each superpixel has  $3 \times 3 = 9$  pixels.

## 2.2 Input Regions Are Points

In this section we first consider the simplest possible case, namely,  $\mathcal{R}$  is a set of points. We will construct a map that assigns points to pixels such that the Hausdorff distance between each point and its corresponding pixel is bounded. For a lower bound, consider a set of  $m$  points  $\mathcal{R}$  that all lie within a single pixel. If we want to assign each point to a unique pixel, we clearly need to use  $m$  different pixels. Any set of  $m$  pixels has diameter  $\Omega(\sqrt{m})$ , so at least one of the point regions will be mapped to a pixel at distance  $\Omega(\sqrt{m})$ .

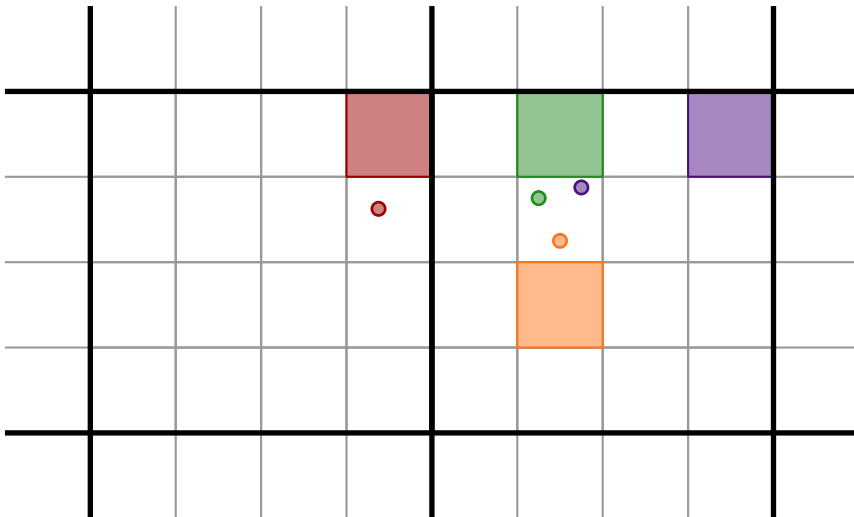


Figure 2.4: The algorithm assigns to each point a pixel in the superpixel that contains the point.

Accompanied by Figure 2.4, we now present a scheme that maps any set of  $m$  points  $\mathcal{R}$  to a set of pixels, such that the Hausdorff distance between any point and its pixel is at most  $O(\sqrt{m})$ . Let  $\Gamma_k$  be a coarsening of  $\Gamma$  with  $k = 2\lceil\sqrt{m}\rceil$ . Each superpixel has the space to accommodate  $m$  disjoint pixels without using the bottom row and left column. For each region  $R_i \in \mathcal{R}$ , we determine the superpixel  $S$  containing  $R_i$ . We choose a pixel  $S[x, y]$  that is not yet colored and color it with  $c_i$ ; that pixel is  $P_i$ . As, per definition, no two pixels  $S[x, y]$  and  $S'[x', y']$  are adjacent, even if  $S = S'$ , this results in a valid assignment. As for each  $R_i \in \mathcal{R}$ , the point  $R_i$  and the pixel  $P_i$  are in the same superpixel, we have  $d'_H(R_i, P_i) = \Omega(\sqrt{m})$ .

**Theorem 2.1.** *If  $\mathcal{R}$  is a set of  $m$  points, a valid assignment exists such that for each region  $R_i \in \mathcal{R}$  with a corresponding region  $P_i$ , we have  $d'_H(R_i, P_i) = O(\sqrt{m})$ . Furthermore, there exists a set  $\mathcal{R}$  of  $m$  points such that for every valid assignment we have  $d'_H(R_i, P_i) = \Omega(\sqrt{m})$ .*

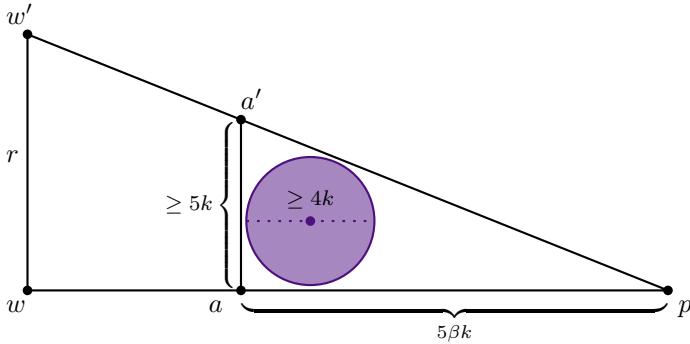


Figure 2.5: If the distance between  $p$  and  $a$  is  $5\beta k$  then the triangle induced by  $p$ ,  $a$  and  $a'$  contains a disk with diameter at least  $2\sqrt{2}k$ . This disk contains a superpixel.

### 2.3 Input Regions Are Convex $\beta$ -Fat Regions

A connected region  $R$  is  $\beta$ -fat if for some point  $w$  in  $R$ , the ratio of the radius of the smallest  $w$ -centered circle containing  $R$  and the radius of the largest  $w$ -centered circle contained in  $R$ , is at most  $\beta$  [96]. Observe that the only regions that are 1-fat are points and disks, as points are  $\beta$ -fat regions for any  $\beta \geq 1$  by convention. In this section we consider the class of convex  $\beta$ -fat regions for a constant  $\beta$ . From Section 2.2 it follows that for any  $m$ , there exists a set of  $m$  regions for which the Hausdorff distance between  $\mathcal{R}$  and any valid assignment is  $\Omega(\sqrt{m})$ .

Let  $\mathcal{R}$  be a set of convex  $\beta$ -fat regions and let  $\Gamma_k$  be a coarsening of  $\Gamma$  with  $k = 4\lceil\sqrt{m}\rceil + 2$ . For a superpixel  $S_i$ , we denote by  $v_i$  the center of  $S_i$ . Also, for two superpixels  $S_1$  and  $S_2$  both contained within the same region  $R$ , the line segment  $\overline{v_1 v_2}$  is called the *spine*. We show some lemmata that lead to an algorithm that maps  $\mathcal{R}$  to a set of grid polygons  $\mathcal{P}$ , such that the Hausdorff distance between any region  $R_i$  and its assigned region  $P_i$  is at most  $O(\beta\sqrt{m})$ .

**Lemma 2.2.** *Let  $R$  be a convex  $\beta$ -fat region, and let  $p$  be a point in  $R$ . Either  $R$  has diameter less than  $2\sqrt{2}\beta k$ , or  $R$  contains a superpixel within distance  $5\beta k$  from  $p$ .*

*Proof.* Let  $w \in R$  be a point such that, for  $r$  ( $r'$ ) being the radius of the incircle (excircle) centered at  $w$  inside (outside)  $R$ ,  $r'/r \leq \beta$ . Also, let  $d$  be the diameter of  $R$ . If  $r \leq \sqrt{2}k$  holds, we have  $r' \leq \sqrt{2}\beta k$  and therefore  $d \leq 2\sqrt{2}\beta k$ . So in the following we assume  $r > \sqrt{2}k$ . We distinguish two cases.

Case  $d(w, p) \leq 5\beta k$ : The superpixel containing  $w$  is inside the incircle centered at  $w$ , as  $r > \sqrt{2}k$ . So now  $p$  has distance less than  $5\beta k$  to a superpixel contained within  $R$ .

Case  $d(w, p) > 5\beta k$ : the construction is depicted in Figure 2.5. Let  $a$  be the point on the segment  $\overline{wp}$  at distance  $5\beta k$  from  $p$ . Further let  $w'$  and  $a'$  be points on the



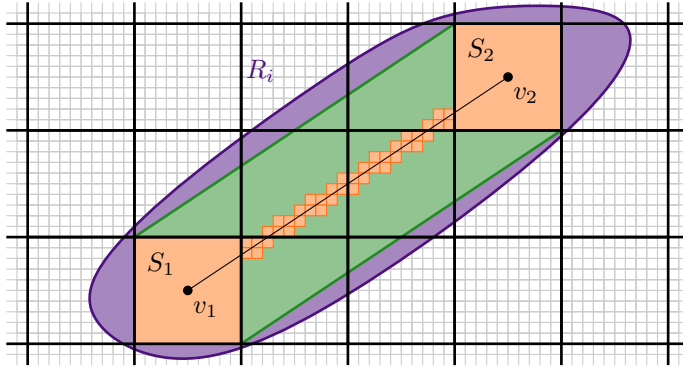


Figure 2.6: A convex  $\beta$ -fat region  $R_i$  (purple), and the region formed by sweeping a superpixel from  $S_1$  to  $S_2$  (green). The grid polygon  $P_i$  (orange) consists of  $S_1$ ,  $S_2$ , and all pixels on the segment between the centers  $v_1$  and  $v_2$ .

same side of  $\overline{wp}$  such that  $\overline{ww'}$  and  $\overline{aa'}$  are both perpendicular to  $\overline{wp}$  and such that  $d(w, w') = r$  and  $a' \in \overline{pw'}$ . Due to  $R$  being convex and  $p, w$  and  $w'$  being inside  $R$  we also know that the triangle induced by those points is contained in  $R$ . We have  $d(a, a') = r \cdot 5\beta k / d(w, p)$ . Due to  $d(w, p) \leq r'$  and  $r' \leq \beta r$  we get  $d(a, a') \geq 5k$ . Note that the isocles right triangle with side length  $2 + 2\sqrt{2}$  contains a disk of radius  $\sqrt{2}$ . As  $5 > 2 + 2\sqrt{2}$  and  $\beta \geq 1$ , the triangle induced by the points  $p, a$  and  $a'$  contains a disk of radius  $\sqrt{2}k$ . This disk contains a superpixel. Also the distance between that superpixel and  $p$  is less than  $5\beta k$ .  $\square$

**Lemma 2.3.** *Let  $S_1, S_2 \subset R$  be two superpixels. Let  $p$  be a point on the spine  $\overline{v_1 v_2}$  and let  $q$  be a point such that  $|p_x - q_x| \leq k/2$  and  $|p_y - q_y| \leq k/2$ . Then,  $q \in R$ .*

*Proof.* We define  $d = q - p = (q_x - p_x, q_y - p_y)$  and have  $|d_x| \leq k/2$  and  $|d_y| \leq k/2$ . Thus  $v_1 + d = (v_{1,x} + d_x, v_{1,y} + d_y) \in S_1 \subset R$  and  $v_2 + d \in S_2 \subset R$ . As  $R$  is convex and as  $q = p + d \in \overline{v_1 + d v_2 + d}$ , we have  $q \in R$ .  $\square$

**Algorithm.** This leads to the following algorithm with two cases for each region  $R_i$ , depending on the diameter of  $R_i$ .

Case 1: the diameter of  $R_i$  is at least  $2\sqrt{2}\beta k$ . Then, the set of superpixels  $S_i$  contained in  $R_i$  is not empty. We need two steps. First we assign all pixels in each superpixel of  $S_i$  to  $R_i$ . Note that  $S_i$  is not necessarily connected, as can be seen in Figure 2.6. Nonetheless we can connect the superpixels in the second step.

Let  $S_1$  and  $S_2$  be two superpixels in different connected components of the superpixels contained in  $R_i$ . We aim to connect  $S_1$  and  $S_2$  with a path of pixels. We do so by coloring all pixels that intersect the spine  $\overline{v_1 v_2}$  with  $c_i$ . We repeat this process for each pair of superpixels contained in  $R_i$ , resulting in a connected grid polygon  $P_i$ .

We finish by also coloring all pixels enclosed within  $P_i$  with  $c_i$ . As a result,  $P_i$  is simply connected.

Finally, we show that no two such colored pixels of different regions are adjacent. For a region  $R$ , using Lemma 2.3, we know that each point within distance  $k/2$  to a point on a spine of  $R$  is contained in  $R$ . As  $k/2 \geq 3 > 2\sqrt{2}$ , each pixel adjacent to a pixel intersected by the spine is in  $R$ . Thus, the grid polygons  $P_i$  are disjoint.

Case 2: the diameter of  $R_i$  is smaller than  $2\sqrt{2}\beta k$ . We select any superpixel  $S$  intersected by  $R_i$ . First, if  $S$  is not intersected by any spine of two superpixels that are within another region, let  $Q$  be any quadrant of  $S$ . Else assume that  $S$  is intersected by a spine of two superpixels that are within a region  $R_j$ , see Figure 2.7. Lemma 2.3 implies that the center of  $S$  is contained in  $R_j$ , thus no spine of another region can intersect  $S$ . Also, as  $R_j$  is convex and  $S \not\subseteq R_j$ , at least one corner of  $S$  is not contained in  $R_j$ . This, together with Lemma 2.3, implies that at least one quadrant  $Q$  of  $S$  is not intersected by any spine spanned by two superpixels contained in another region.

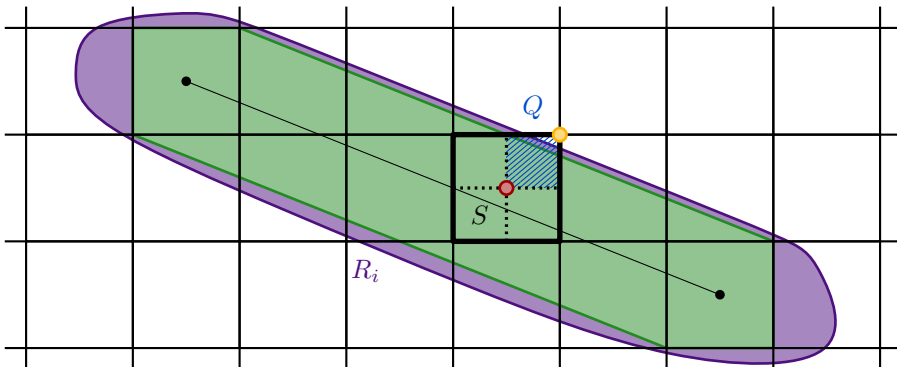


Figure 2.7: A convex  $\beta$ -fat region  $R_i$  (purple), and the region (green) formed by sweeping a superpixel between the superpixels in  $R_i$ . For a superpixel  $S$  not contained in  $R_i$  but intersected by the spine of two of its superpixels, the center (red) of  $S$  is in  $R_i$ , but there is at least one corner (yellow) of  $S$  outside  $R_i$ . There is one quadrant  $Q$  (blue) that will not contain pixels of  $P_i$ .

For the quadrant  $Q$ , let  $Q[x, y]$  be the pixel that is the  $(2x)^{\text{th}}$  from the left and  $(2y)^{\text{th}}$  from the bottom within  $Q$ . As the side length of  $Q$  is  $k/2 = 2\lceil\sqrt{m}\rceil + 1$ ,  $Q$  contains at least  $m$  distinct pixels  $Q[x, y]$  and no pixel  $Q[x, y]$  is adjacent to the boundary of  $Q$ . Similarly to Section 2.2, we choose a pixel  $Q[x, y]$  that is not yet colored and color it with  $c_i$ ; that pixel is  $P_i$ .

Note that we slightly deviate from the scheme proposed in Subsection 2.1.3. For example, we do not require that any superpixel intersected by a region contains a pixel of the corresponding color; we only require that a superpixel within bounded distance contains a pixel of the corresponding color.

**Bounds on the Hausdorff distance.** What remains to be proven is that for each region  $R_i$ ,  $d'_H(R_i, P_i) \leq 5\beta k$  holds. If the region  $R_i$  is in case 2, it has diameter smaller than  $2\sqrt{2}\beta k$ . Then the maximum distance between a point in  $R_i$  and any point in the pixel  $P_i$  is  $2\sqrt{2}\beta k + \sqrt{2}k \leq 5\beta k$ . Thus,  $d'_H(R_i, P_i) \leq 5\beta k$ .

Else,  $R_i$  falls into case 1. First, we prove that for each (boundary) point  $p$  of  $P_i$ , there is a (boundary) point  $q$  of  $R_i$  within distance  $5\beta k$ . By construction, we know  $P_i \subseteq R_i$ , so the claim holds for interior points. Now, let  $p \in \partial P_i$ . We assume for the sake of contradiction that there is no point of  $\partial R_i$  within distance  $\sqrt{2}k$ . As  $p$  is contained within  $R_i$ , we have that  $R_i$  contains the superpixels containing  $p$ , a contradiction. Second, we prove the inverse. For a point  $q$  of  $R_i$ , Lemma 2.2 guarantees that  $R_i$  contains a superpixel  $S$  within distance  $5\beta k$  of  $q$ . Then  $S \subseteq P_i$  holds, proving the claim. As  $P_i \subseteq R_i$ , this also proves that for each boundary point  $q$  of  $R_i$ , there is a boundary point  $p$  of  $P_i$  within distance  $5\beta k$ .

**Theorem 2.4.** For each  $\beta \geq 1$ , if  $\mathcal{R}$  is a set of  $m$   $\beta$ -fat convex regions, a valid assignment exists such that for each region  $R_i \in \mathcal{R}$  with a corresponding region  $P_i$ , we have  $d'_H(R_i, P_i) = O(\beta\sqrt{m})$ . Furthermore, there exists a set  $\mathcal{R}$  of  $m$   $\beta$ -fat regions such that for every valid assignment  $d'_H(R_i, P_i) = \Omega(\sqrt{m})$ .

## 2.4 Input Regions Are Convex Regions

When  $\mathcal{R}$  is a set of convex regions, we can show a higher lower bound on the Hausdorff distance than in the previous cases.

**Lemma 2.5.** There is a set of regions  $\mathcal{R}$ , such that for any valid assignment, we have  $d'_H(R_i, P_i) = \Omega(m)$ .

*Proof.* Let  $\mathcal{R}$  be a set containing  $m$  long, horizontal line segments stacked close each other, as shown in Figure 2.8. Formally, let  $R_i = \overline{p_i q_i}$  with  $p_i = (-m, i/(m+1))$  and  $q_i = (m, i/(m+1))$ , for  $i \in \{1, \dots, m\}$ . Assume for the sake of contradiction that

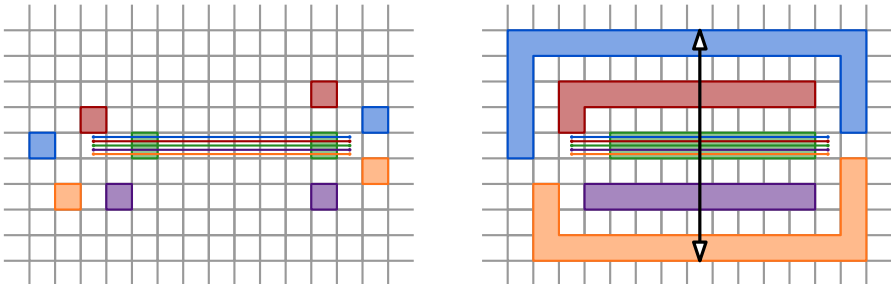


Figure 2.8: The lower bound construction for convex regions.

there exists a valid assignment such that  $d'_H(R_i, P_i) < m - 1$  for each  $R_i \in \mathcal{R}$  and corresponding  $P_i$ . Then, for each  $i \in \{1, \dots, m\}$ , there is a pixel in  $P_i$  at distance at most  $m - 1$  to  $p_i$  and a pixel in  $P_i$  at distance at most  $m - 1$  to  $q_i$ . This means  $P_i$  has pixels left and right the  $y$ -axis. Furthermore,  $P_i$  is connected and thus it has at least one pixel intersecting the  $y$ -axis. As no two pixels from different grid polygons are adjacent, we know the distance between the top of the uppermost and the bottom of the lowermost pixels belonging to grid polygons is at least  $2m - 1$ . Thus for at least one grid polygon, the Hausdorff distance between the grid polygon and its corresponding segment is at least  $m - 1$ .  $\square$

We will describe an algorithm that, given a set of convex regions  $\mathcal{R}$ , gives a valid assignment of disjoint orthoconvex grid polygons such that, for all  $i$ ,  $d'_H(R_i, P_i) = O(m)$ .

**Observation 2.6.** *Let  $R_1, R_2 \in \mathcal{R}$  be two disjoint convex regions, and let  $\ell$  be a horizontal line that intersects  $R_1$  left of  $R_2$ . Then any horizontal line intersecting both  $R_1$  and  $R_2$  intersects  $R_1$  left of  $R_2$ . Similarly, all vertical lines that intersect both  $R_1$  and  $R_2$  do so in the same order.*

Observation 2.6 allows us to define two partial orders  $\preceq_x$  and  $\preceq_y$  on  $\mathcal{R}$ :  $R_i \preceq_x R_j$  if and only if there is a horizontal line intersecting both regions and  $R_i$  intersects the line left of  $R_j$ ; since the regions are convex we get a partial order [68]. We extend this partial order as follows: first we add transitive arrows, where we recursively add the inequality  $R_i \preceq_x R_j$  if there exists a region  $R_k$  with  $R_i \preceq_x R_k$  and  $R_k \preceq_x R_j$  and we denote this partial order by  $\Pi_x(\mathcal{R})$ . We then transform  $\Pi_x(\mathcal{R})$  into a linear order  $X_{\mathcal{R}} : \mathcal{R} \rightarrow [1, m]$  in any manner. A linear order  $Y_{\mathcal{R}} : \mathcal{R} \rightarrow [1, m]$  is defined symmetrically.

Given  $X_{\mathcal{R}}$  and  $Y_{\mathcal{R}}$ , we assign a coloring. Let  $\Gamma_k$  be a coarsening of  $\Gamma$  with  $k = 2m$ . Recall that, for any superpixel  $S \in \Gamma_k$ , we denote by  $S[x, y]$  the pixel that is the  $(2x)^{\text{th}}$  from the left and  $(2y)^{\text{th}}$  from the bottom within  $S$ . Additionally the horizontal and vertical lines induced by  $\Gamma_k$  are called *major lines*. Each region  $R_i$  that intersects at most one major horizontal line and at most one major vertical line is a *small region*. Each region  $R_i$  that intersects at least two major parallel lines is a *large region*. Our assignment of regions to pixels, illustrated in Figure 2.9, is:

1. For each small region  $R_i$  we choose one superpixel  $S$  intersected by  $R_i$  and color the pixel  $p(S, R_i) = S[X_{\mathcal{R}}(R_i), Y_{\mathcal{R}}(R_i)]$  with  $c_i$ . This single pixel will be  $P_i$ .
2. For each superpixel  $S$  and each large region  $R_i$  intersecting  $S$  that also intersects the two major horizontal lines incident to  $S$ , or the two major vertical lines incident to  $S$ , we color  $p(S, R_i) = S[X_{\mathcal{R}}(R_i), Y_{\mathcal{R}}(R_i)]$  with  $c_i$ . Note that region  $R_i$  need not intersect two opposite edges of  $S$ .
3. For any two pixels that are colored with  $c_i$  in edge-adjacent superpixels ( $R_i$  must be large), we color all pixels in the row or column between them with  $c_i$ .

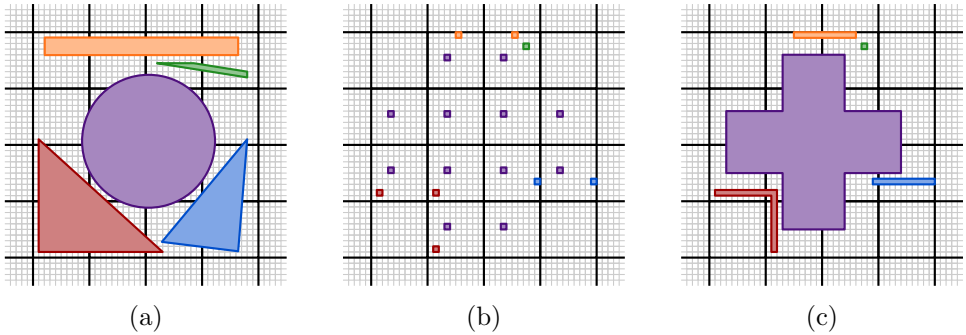


Figure 2.9: The coloring algorithm for convex regions. (a) The input of five convex regions, overlaid onto a superpixel grid with  $k = 10$ . (b) The pixels colored in Steps 1 and 2 of the algorithm. (c) The final coloring obtained after Steps 3 and 4.

4. For any four superpixels that share a common vertex, if they each contain a pixel colored with  $c_i$  in Step 2, we color all pixels in the square between these pixels with  $c_i$ .

Let  $\mathcal{P}$  be the set of polygons induced by this grid coloring.

**Lemma 2.7.** *Each polygon  $P_i \in \mathcal{P}$  is simply connected and orthoconvex.*

*Proof.* If  $R_i$  is small,  $P_i$  is a single pixel and thus simply connected and orthoconvex. If  $R_i$  is large, we argue as follows: let  $B$  be the axis-aligned bounding-box of  $R_i$ . All superpixels intersected by  $B$  are incident to two major parallel lines intersected by  $R_i$  with the exception of the four superpixels containing the corners of  $B$ . Also, the superpixels intersected by  $R_i$  form a simply connected, orthoconvex grid polygon  $\mathbb{S}$ . Recall that in Step 2 we place a pixel in each superpixel intersected by  $R_i$  that has two major parallel lines incident to it. Thus, the only superpixels intersected by  $R_i$  that do not have a pixel are superpixels that contain a corner of  $B$ . Let  $\mathbb{S}'$  be the grid polygon formed by the superpixels that have a pixel of  $P_i$ . Since our algorithm picks the pixel with the same relative position inside each superpixel, connects the chosen pixels in adjacent superpixels with a horizontal or vertical row of pixels, and fills the holes, we have that  $P_i$  is an orthoconvex grid polygon. It also follows that  $P_i$  is simply connected if and only if  $\mathbb{S}'$  is simply connected.

Now, assume for the sake of contradiction that  $P_i$  and  $\mathbb{S}'$  are not simply connected. Recall that the only superpixels in  $\mathbb{S}$  that are not in  $\mathbb{S}'$  contain corners of  $B$  and that  $\mathbb{S}$  is simply connected. Let  $S \subset \mathbb{S}$  be a superpixel containing a corner of  $B$  such that  $\mathbb{S}'$  and  $S$  together form a connected grid polygon. Without loss of generality, we assume that  $S$  contains the bottom-left corner of  $B$ , see Figure 2.10. Let  $S_t$  ( $S_r$ ,  $S_{tr}$ ) be the superpixel directly above (right of, to the top-right of)  $S$ . Both  $\mathbb{S}$  and  $\mathbb{S}'$  contain  $S_t$  and  $S_r$ . Also,  $\mathbb{S}'$  does not contain  $S_{tr}$ . We know that  $R_i$  does not intersect the

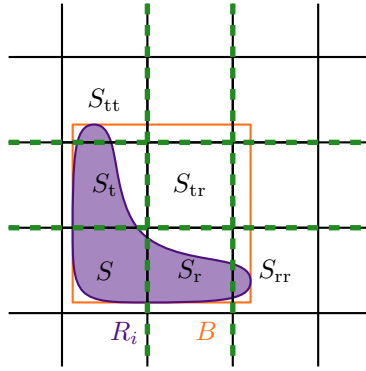


Figure 2.10: A region  $R_i$  and its bounding box  $B$ . If  $P_i$  is not simply connected,  $R_i$  intersects the superpixels  $S$ ,  $S_t$ , and  $S_r$ , as well as the major lines marked in green, but not the superpixel  $S_{tr}$ . Note that this is only possible for non-convex  $R_i$ .

major vertical line incident to the left of  $S_t$ , so  $R_i$  intersects the two major horizontal lines incident to  $S_t$  which also are the two major horizontal lines incident to  $S_{tr}$ . Since  $S_{tr} \not\subseteq S$ ,  $R_i$  does not intersect  $S_{tr}$ . It follows that  $R_i$  intersects the superpixel  $S_{tt}$  directly above  $S_t$ . Symmetrically,  $R_i$  intersects the superpixel  $S_{rr}$  directly right of  $S_r$ . Any line segment between a point in  $S_{tt}$  and a point in  $S_{rr}$  intersects  $S_{tr}$ . Since  $R_i$  is convex,  $R_i$  intersects  $S_{tr}$ ; a contradiction.  $\square$

**Lemma 2.8.** *The polygons in  $\mathcal{P}$  are pairwise disjoint.*

*Proof.* Assume by contradiction that the colorings of two regions  $R$  and  $Q$  intersect. Then the intersection was created during one of the four coloring steps. In Steps 1 and 2, we assign each color to single pixels per superpixel in unique rows and columns, hence they cannot create two colorings that intersect.

Let the colorings of  $R$  and  $Q$  intersect after Step 3. This implies that  $R$  and  $Q$  are both large regions. The intersection occurs between a vertical and horizontal pixel sequence in a super pixel  $S$ . Assume without loss of generality that the vertical sequence belongs to  $R$  and the horizontal sequence belongs to  $Q$ . Consider the case that the pixel  $p(S, R)$  assigned to  $R$  in  $S$  in Step 2 is to the top-left of  $p(S, Q)$  (see Figure 2.11); the other three cases are symmetric. Then the intersection occurs between the column sequence connecting  $p(S, R)$  to  $p(S_d, R)$  and the row sequence connecting  $p(S, Q)$  to  $p(S_\ell, Q)$ , where  $S_d$  is the superpixel directly below  $S$  and  $S_\ell$  is the superpixel directly to the left of  $S$ .

Since  $Q$  is large and assigned a pixel in  $S$  it intersects both horizontal major lines incident to  $S$  or both vertical major lines incident to  $S$ . The same applies for  $R$ . We first consider the case where  $Q$  does not intersect the major horizontal line incident to the bottom of  $S$ , and hence it intersects both vertical lines. That is,  $Q$  spans the

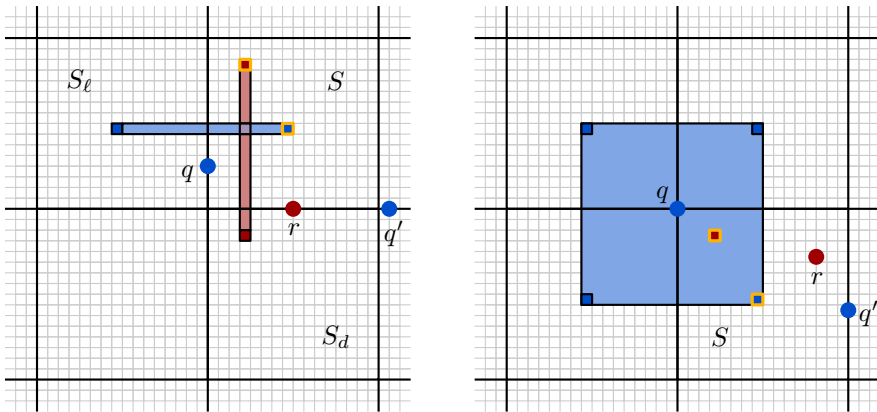


Figure 2.11: The cases for the proof of Lemma 2.8. The pixels  $p(S, R)$  and  $P(S, Q)$  are highlighted in yellow.

vertical slab defined by  $S$  and does so in or above  $S$ . Since  $R$  intersects the cell  $S_d$  below  $S$  it then follows that  $R \preceq_y Q$ . However, since  $p(S, R)$  lies above  $p(S, Q)$  we also have  $Q \preceq_y R$ . Since  $Q \neq R$  we thus obtain a contradiction.

Thus,  $Q$  intersects the major horizontal line  $\ell$  incident to the bottom of  $S$ . Since  $R$  is convex, and intersects both  $S$  and  $S_d$  it intersects the bottom edge of  $S$  (and thus  $\ell$ ) in a point  $r$ . Symmetrically,  $Q$  intersects the left edge of  $S$  in a point  $q$ . If  $Q$  also intersects the horizontal line  $\ell$  in some point  $q'$  this point cannot be left of  $r$ , as this would immediately imply that  $Q \preceq_x R$ , contradicting the assignment of  $p(S, R)$  and  $p(S, Q)$ . So  $q'$  lies right of  $r$ . However, then the vertical ray starting at  $r$  pointing upwards intersects the segment connecting  $q$  and  $q'$ . Since  $Q$  is convex, this segment is contained in  $Q$ . This implies  $R \preceq_y Q$ , which again contradicts the assignment of  $p(S, R)$  and  $p(S, Q)$ . It follows that Step 3 does not create intersecting colorings.

Finally, let the colorings of  $R$  and  $Q$  intersect only after Step 4. Without loss of generality, the coloring of a region  $R$  is entirely contained in the coloring of a large region  $Q$ . Let  $S$  be the superpixel containing the lone pixel of  $R$ . Without loss of generality we assume that the pixel  $p(S, R)$  assigned to  $R$  in  $S$  is to the top-left of  $p(S, Q)$  (See Figure 2.11). Thus,  $Q$  intersects  $S$ , the superpixel above  $S$ , the superpixel left of  $S$ , and the superpixel left and above  $S$ . The point  $q$  where these four superpixels meet lies inside  $Q$  by convexity. Let  $r$  be any point in  $R \cap S$ .

As  $Q$  is a large region it needs to intersect two opposite major lines incident to  $S$ . Assume that  $Q$  intersects the vertical major lines, in particular the one incident to the right edge of  $S$  in a point  $q'$ . The vertical line through  $r$  intersects the segment between  $q$  and  $q'$ . The point  $r$  is above that segment, because the opposite would imply  $R \preceq_y Q$ . As a consequence  $r$  is also right of the segment between  $q$  and  $q'$ , which implies that the horizontal line through  $r$  intersects this segment left of  $R$ , a

contradiction. The case where  $Q$  intersects the major horizontal line through the bottom edge of  $S$  is symmetric.  $\square$

Lastly, it remains to be shown that the Hausdorff distance  $d'_H(R_i, P_i)$  are in  $O(m)$  for each region  $R_i \in \mathcal{R}$ . If a region  $R_i$  intersects a superpixel  $S$ ,  $P_i$  has a pixel in  $S$  or in at least one of the eight adjacent superpixels. Conversely, if  $P_i$  contains a pixel in  $S$ , we know that  $R_i$  intersects  $S$ . This gives a bound on the Hausdorff distance between the regions and the grid polygons. For the boundaries, note that if  $R_i$  contains a superpixel  $S$  and all four edge-adjacent superpixels,  $P_i$  contains  $S$ . Furthermore, if  $P_i$  contains a superpixel  $S$ ,  $R_i$  also contains  $S$ . Together this gives a bound on the Hausdorff distance between the boundaries. Since superpixels have size  $\Theta(m)$ , the Hausdorff distance between  $R_i$  and  $P_i$  and between their boundaries is at most  $O(m)$ . We thus obtain the following result.

**Theorem 2.9.** *If  $\mathcal{R}$  is a set of  $m$  convex regions, a valid assignment exists such that for each region  $R_i \in \mathcal{R}$  with a corresponding region  $P_i$ , we have  $d'_H(R_i, P_i) = O(m)$ . Furthermore, there exists a set  $\mathcal{R}$  of  $m$  convex regions such that for every valid assignment, there exists some  $1 \leq i \leq m$  with  $d'_H(R_i, P_i) = \Omega(m)$ .*

## 2.5 Input Regions Are General Regions

When the input regions are arbitrary, we see a sharp contrast between the case  $m \leq 2$ , where constant Hausdorff distance can be realized, and the case  $m \geq 3$ , where the Hausdorff distance may be unbounded. The fact that a single region can be represented as a grid polygon with constant Hausdorff distance was shown before by Bouts et al. [26]. In Subsection 2.5.1 we show that the same result holds for two regions. In Subsection 2.5.2 we show that for three regions, no bounded Hausdorff distance bound exists that applies to all inputs.

### 2.5.1 Two Regions

Our result for two arbitrary regions is based on a combination of two previous results: mapping a polygon to the grid with constant Hausdorff distance by Bouts et al. [26], and a result on the Painter's Problem by van Goethem et al. [64]. We briefly explain the former result in our framework using superpixels first (see Figure 2.12), and then extend it to our case with two regions using the latter result.

Assume we have a region  $R$  that we want to represent by a grid polygon  $P$ . Consider the grid coarsening  $\Gamma_3$ , which has superpixels of  $3 \times 3$  pixels. For every superpixel fully covered by  $R$ , choose all nine pixels in  $P$ . For every superpixel visited but not covered by  $R$ , take the middle pixel. Take nothing from superpixels not visited by  $R$ . Let the chosen pixels be  $P'$ .



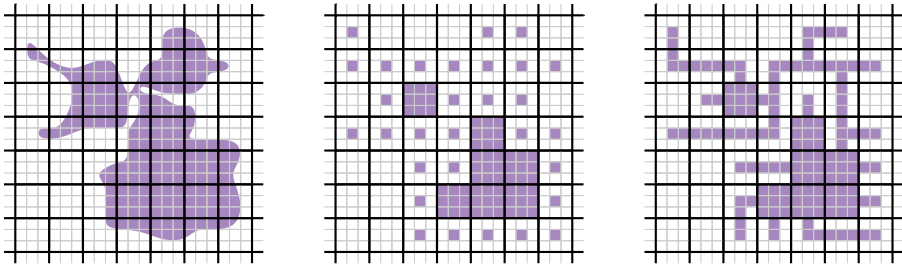


Figure 2.12: Left, a region with  $\Gamma$  and  $\Gamma_3$ . Middle, the set  $P'$  of pixels chosen in the first selection. Right, the set  $P$  of pixels chosen after the spanning tree pixels are added.

Observe that  $P'$  forms a set of grid polygons that has no interior boundary cycles. Also observe that all superpixels for which at least one pixel is in  $P'$  is a connected (but not necessarily simply connected) part of  $\Gamma_3$ .

We make  $P'$  into one simply connected grid polygon  $P$  by using a (minimum) spanning tree on the components of  $P'$ . We will add pixels from visited superpixels only, and only ones adjacent to the already chosen center pixel. Two separate components will always be connected using one or two pixels.

Since the boundary of  $P$  does not intersect the interior of fully covered superpixels and visited superpixels always have a piece of boundary of  $P$ , it is easy to see that  $d_H(R_i, P_i) = \Theta(1)$  and  $d_H(\partial R_i, \partial P_i) = \Theta(1)$ . This is an alternative way to show the same results as Bouts et al., albeit with worse constants.

A Painter's Problem instance takes a grid, and for each cell, the color white, blue, red, or purple. White indicates the absence of red and blue while purple indicates the presence of both red and blue. The question is whether two disjoint simply connected regions for red and blue exist that are consistent with all specifications of the cells, or, in the terminology of [64], "admits a painting". Since red cells can simply be colored red and blue cells blue, the problem boils down to recoloring the purple cells with red and blue pieces. The red and blue pieces in a cell provide a panel, and all panels together make up a painting. They prove:

**Lemma 2.10. (Theorem 2 in [64])** *If a partially 2-colored grid admits a painting, then it admits a 5-painting.*

In a 5-painting each cell contains at most 5 components. The components make sure that the overall red and blue parts are connected across the whole painting. Additionally [64] show that each cell has at most 3 intervals of alternating red and blue along each side. This implies that there are only a constant number of configurations within a cell, so all configurations can be represented using a grid of constant size  $c$  for each cell.

In our problem, we have two regions  $R_1$  and  $R_2$  that we call red and blue, for consistency. We create a grid coarsening  $\Gamma_{c+2}$ . We record for every superpixel whether it is fully covered by red or blue, or visited by red and/or blue. If one color covers a superpixel completely, we assign all of its pixels to that color. If a color, say, red, visits a superpixel but blue does not, we start by making the middle  $c \times c$  pixels of that superpixel red. Finally, for all superpixels visited by both red and blue, we apply the results from [64]. Since the recording of colors with panels comes from disjoint simply connected regions, namely, our input, we know that the 2-colored grid of superpixels admits a painting with connected regions/colors, so it admits one as specified in Lemma 2.10.

Once we choose a coloring of pixels in each 2-colored superpixel according to the panels, it remains to make the red set and blue set of pixels simply connected. The method from [64] did not produce any cycles in the 2-colored superpixels, the visited 1-colored superpixels are separate connected components of  $c \times c$  pixels in the middle, and the covered 1-colored superpixels cannot create cycles either. We create a single red component by making a spanning tree of the red components. To achieve this, we only need to use pixels in the outer ring of the visited 1-colored superpixels. Then we do the same with blue. Since we add pixels of the same color to 1-colored superpixels, we will never try to color a pixel in both colors or create crossings. We then obtain the following result:

**Theorem 2.11.** *If  $\mathcal{R}$  consists of two disjoint, simply connected regions, a valid assignment exists such that for each region  $R_i \in \mathcal{R}$  with corresponding  $P_i$ , we have  $d'_H(R_i, P_i) = \Theta(1)$ .*

## 2.5.2 Three or More Regions

In the following we argue that the Hausdorff distance between an input of at least three general regions and any corresponding grid polygons is unbounded. Formally, for a given integer  $h > 0$ , we show a construction of regions  $\mathcal{R} = \{R, B, G\}$  for which there are no corresponding grid polygons with Hausdorff distance smaller than  $h$ .

We construct regions  $\mathcal{R} = \{R, B, G\}$  that form nested spirals with a long bottleneck of height 1. The bottleneck is traversed from left to right  $h$  times by each of  $R$ ,  $B$ , and  $G$ . If we remove the parts of  $R$ ,  $B$ , and  $G$  inside the bottleneck, we get  $3h + 3$  connected components in total. This is illustrated in Figure 2.13 for  $h = 2$ . Outside the horizontal strip of height 1 containing the bottleneck, the three regions are more than  $2h$  apart. We define the part of the plane within distance  $h$  of at least one of the bottom horizontal segments of the regions  $\mathcal{R}$  as  $\mathcal{I}$ . All region components are connected inside  $\mathcal{I}$ . Inside  $\mathcal{I}$ , it is possible that the grid polygons make different connections than those in  $\mathcal{R}$ . However, we argue that no matter how these connections are made, the grid polygons  $P_R$ ,  $P_B$ , and  $P_G$ , together have to pass through  $\mathcal{I}$  from left to right at least  $h + 2$  times, thus requiring  $\mathcal{I}$  to have height at least  $2h + 3$ . However, the available vertical space is only  $2h + 1$  if the Hausdorff distance is below  $h$ , allowing  $h + 1$  connections of pixel polygons. Hence, we obtain a contradiction.

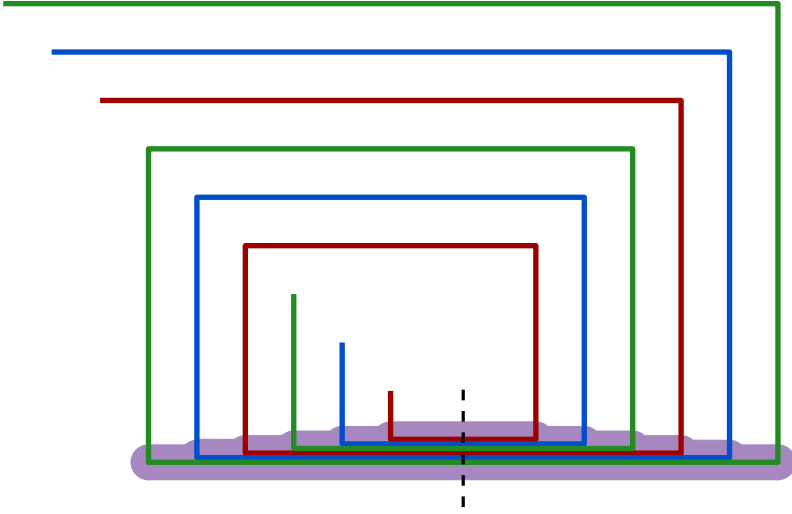


Figure 2.13: The regions for  $h = 2$ ;  $\mathcal{I}$  is highlighted. The dashed line subdivides the boundary of  $\mathcal{I}$  into its left and right part. The height of  $\mathcal{I}$  and the distance between two vertical segments is  $2h + 1$ . The figure is not to scale.

**The construction.** Refer to Figure 2.13. Formally, the red region  $R$  is a polyline starting at  $(-1.5, 0)$ . From then on it consists of a downward segment  $\alpha_1^R$ , a rightward segment  $\beta_1^R$ , an upward segment  $\gamma_1^R$ , a leftward segment  $\delta_1^R$ , a downward segment  $\alpha_2^R$  and so on, until the final segment is the leftward segment  $\delta_h^R$ .

Similarly the blue (green) region  $B$  ( $G$ ) is a polyline starting at  $(-2.5, 1)$  ( $(-3.5, 2)$ ) and with segments of the same orientations  $\alpha_i^B, \beta_i^B, \gamma_i^B$  and  $\delta_i^B$  ( $\alpha_i^G, \beta_i^G, \gamma_i^G$  and  $\delta_i^G$ ) for  $i \in \{1, \dots, h\}$ . Let  $A \in \{R, B, G\}$ ,  $j^R = 1$ ,  $j^B = 2$ , and  $j^G = 3$ , then the lengths of the line segments are:

- $|\alpha_i^A| = (6i + 2j^A - 1)(2h + 1) - (3i + j^A - 4)(2h + 1 - \frac{1}{3h})$
- $|\beta_i^A| = (6i + 2j^A + 1)(2h + 1)$
- $|\gamma_i^A| = (6i + 2j^A + 2)(2h + 1) - (3i + j^A - 4)(2h + 1 - \frac{1}{3h})$
- $|\delta_i^A| = (6i + 2j^A + 4)(2h + 1)$ .

Observe that the distance between any two vertical segments of the spirals (the segments  $\alpha$  and  $\gamma$ ) is at least  $2h + 1$ . This also holds for the horizontal segments on the top of the spiral (the segments  $\delta$ ). The horizontal segments (the segments  $\beta$ ) at the bottom, however, are only  $\frac{1}{3h}$  apart. Hence, any point that is within distance  $h$  of at least two regions, say  $A$  and  $A'$ , must be within distance  $h$  of two of the bottom segments (that is, some  $\beta_i^A$  and  $\beta_j^{A'}$ ).

**The region  $\mathcal{I}$ .** Let  $\mathcal{I}$  be the region of all pixels that contain a point that is within distance  $h$  of at least two regions in  $\mathcal{R}$ , and observe that this region has height at most  $2h + 1$ . See Figure 2.13. We say a point on the boundary  $\partial\mathcal{I}$  of  $\mathcal{I}$  is part of the left (right) boundary of  $\mathcal{I}$  if its  $x$ -coordinate is negative (positive).

Now note that each region  $A$  in  $\mathcal{R}$  passes through  $\mathcal{I}$  exactly  $h$  times, and thus  $A \setminus \mathcal{I}$  consists of  $h + 1$  connected components. Moreover, note that  $h$  of the connected components of  $A \setminus \mathcal{I}$  touch the left part of the boundary of  $\mathcal{I}$ , and  $h$  connected components touch the right part of the boundary of  $\mathcal{I}$  (so  $h - 1$  components touch both). In particular, let  $r_1, \dots, r_h$  be the  $h$  points, ordered from left to right, at which region  $R$  intersects the right part of the boundary of  $\mathcal{I}$ . Each such a point uniquely corresponds to a connected component of  $R \setminus \mathcal{I}$ . We analogously define the intersection points  $b_1, \dots, b_h$  of  $B$  and the intersection points  $g_1, \dots, g_h$  of  $G$ . Observe that when traversing the boundary of  $\mathcal{I}$  from left to right, these points form an alternating sequence  $V_R = (r_1, b_1, g_1, \dots, r_h, b_h, g_h)$  of length  $3h$ :

**Observation 2.12.** *There is a sequence  $V = (r_1, b_1, g_1, \dots, r_h, b_h, g_h)$  of  $3h$  points along the right part of the boundary of  $\mathcal{I}$ , such that each point in  $V$  corresponds to a unique connected component of a region in  $\{R \setminus \mathcal{I}, B \setminus \mathcal{I}, G \setminus \mathcal{I}\}$ .*

Similarly, there is such an alternating sequence  $V'$  of length  $3h$  on the left part of the boundary of  $\mathcal{I}$  so that each point uniquely corresponds to a connected component of a region in  $\{R \setminus \mathcal{I}, B \setminus \mathcal{I}, G \setminus \mathcal{I}\}$ .

**The grid polygons.** Assume by contradiction that there is a valid assignment of grid polygons  $\mathcal{P} = \{P_R, P_B, P_G\}$  that have Hausdorff distance at most  $h$  to  $\mathcal{R}$ .

For each region  $A \in \mathcal{R}$ , let  $Q_A = P_A \cap \mathcal{I}$  be the part of the grid polygon inside  $\mathcal{I}$  and let  $U_A = P_A \setminus \mathcal{I}$  be the part of the grid polygons outside  $\mathcal{I}$ . As each connected component of  $Q_A$  and  $U_A$  is a grid polygon, we consider  $Q_A$  and  $U_A$  as sets of grid polygons. Let  $\mathcal{Q} = Q_R \cup Q_B \cup Q_G$  denote the set of all connected components inside  $\mathcal{I}$ . We now first argue that outside  $\mathcal{I}$  the grid polygons more or less follow the same structure as the regions in  $\mathcal{R}$ .

**Lemma 2.13.** *For each region  $A \in \mathcal{R}$ , the part of the grid polygon  $U_A$  outside of  $\mathcal{I}$  consists of at least  $h + 1$  connected components. Moreover, each point  $a \in V \cap A$  corresponds to a unique connected component of  $U_A$  that touches the right part of the boundary of  $\mathcal{I}$  only within distance  $h$  of  $a$ .*

*Proof.* Outside of  $\mathcal{I}$ , the regions in  $\mathcal{R}$ , so in particular the connected components of  $A \setminus \mathcal{I}$ , are far apart (that is, at distance more than  $2h$ ), so no connected component in  $U_A$  can be close to two such connected components. This proves the first part of the statement. For the second part: for each point  $a \in V \cap A$  consider the point  $a' = (a_x, a_y + h + \varepsilon)$ , for some arbitrarily small  $\varepsilon > 0$ . There must be a connected component  $U$  of  $U_A$  that is within distance  $h$  of this point, as it is too far away from any point in  $\mathcal{I}$ . As argued above, no connected component of  $U_A$  can cover two such

points as they are too far apart. Hence, each such point  $a'$  corresponds to a unique connected component  $U \in U_A$ . Since  $a'$  is uniquely associated point  $a$ , the same holds for  $a$ . Finally, the connected component of  $A \setminus \mathcal{I}$  corresponding to  $U$  intersects the right part of  $\partial\mathcal{I}$  only in point  $a$ , and thus any points in  $U$  on the right part of  $\partial\mathcal{I}$  lie within distance  $h$  of  $a$ .  $\square$

By Lemma 2.13 the total number of connected components in  $U_R, U_B$ , and  $U_G$  is at least  $3h + 3$ . Moreover, we can represent ( $3h + 3$  of) these components by the two sequences  $V$  and  $V'$  on the boundary of  $\mathcal{I}$ : each such component is represented by at most one point in  $V$  and at most one point in  $V'$ .

For each region  $A \in \mathcal{R}$ ,  $Q_A$  consists of an unknown number of connected components (grid polygons), which connect the connected components of  $U_A$  into a single grid polygon  $P_A$ . A grid polygon  $Q$  of  $Q_A$  can “connect” to some connected component  $U$  of  $U_A$  by *covering* either the point representing  $U$  in  $V$  or the point representing  $U$  in  $V'$ . Polygon  $Q$  covers  $a \in V$  if and only if  $Q$  and  $U$  have a point in common (which must be on the boundary of  $\mathcal{I}$ ) within distance  $h$  of  $a$ .

We now argue that in total, over  $Q_R, Q_B$ , and  $Q_G$ , there are at least  $h + 2$  such grid polygons  $Q$  that connect to grid polygons in  $U_A$  both on the left and on the right boundary of  $\mathcal{I}$ . Such a grid polygon must occupy a pixel at  $x = 0$ . Moreover, since these grid polygons form different connected components, there is at least one empty pixel in between them at  $x = 0$ . It follows that at  $x = 0$ ,  $\mathcal{I}$  must have height at least  $2(h + 2) - 1 = 2h + 3$ . This contradicts with the fact that  $\mathcal{I}$  has height less than  $2(h + 1) = 2h + 2$ .

The polygons of  $\mathcal{Q}$  induce a partition  $V_{\mathcal{Q}}$  of the points in  $V$  through this covering relation: two points  $u, v \in V$  are in the same set of  $V_{\mathcal{Q}}$  if and only if they are covered by a connected component  $Q \in \mathcal{Q}$ . Let  $|V_{\mathcal{Q}}|$  denote the number of sets in the partition, and let  $C_{\mathcal{Q}} = |V| - |V_{\mathcal{Q}}|$  denote the number of *connections* realized by  $\mathcal{Q}$ . Intuitively,  $C_{\mathcal{Q}}$  corresponds to the decrease in the number of connected components in  $U_R, U_B$ , and  $U_G$  on the right part of  $\partial\mathcal{I}$ . Similarly,  $\mathcal{Q}$  realizes a number of connections on the left part of  $\partial\mathcal{I}$ .

Since the set  $\mathcal{Q}$  connects all regions in  $U_R, U_B$ , and  $U_G$  into just three regions ( $P_R, P_B$  and  $B_G$ ), the partitions  $V_{\mathcal{Q}}$  and  $V'_{\mathcal{Q}}$  realize a total of  $3h$  connections. However, we now argue that any *valid* partition on  $V$ , that is, a partition that can be realized by pairwise disjoint non-intersecting grid polygons inside  $\mathcal{I}$ , can realize only  $h - 1$  connections. This means that by connecting points in  $V$  to each other, we can reduce the number of connected components by only  $h - 1$ . The same holds for the number of connections on the left part of the boundary of  $\mathcal{I}$ . It therefore follows that the remaining  $3h - 2(h - 1) = h + 2$  connections need to be realized by polygons in  $\mathcal{Q}$  that cover points both on the left and on the right boundary of  $\mathcal{I}$ .

**Lemma 2.14.** *Given an alternating sequence  $V = (r_1, b_1, g_1, \dots, r_k, b_k, g_k)$  of  $3k$  3-colored points on a line, any planar drawing below the line connecting points of the same color induces a partition of the points into at least  $2k + 1$  components.*

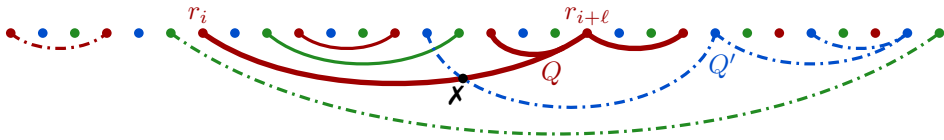


Figure 2.14: A set  $Q$  that includes two red points  $r_i$  and  $r_{i+\ell}$  splits  $V$  into two disjoint subsequences  $V_1$  and  $V_2$ , that have at most one set, namely  $Q$ , in common. If there was a second such a set  $Q'$ , the grid polygons corresponding to  $Q$  and  $Q'$  would intersect.

*Proof.* We prove this by induction on  $k$ . For the base case  $k = 1$ , the sequence  $V$  consists of only one group of three differently colored points. Since the points all have different colors, no drawing can connect two points. Hence, there are at least (exactly) 3 connected components as claimed.

For the induction step, consider an alternating sequence  $V$  of length  $3k$ , with  $k > 1$ . In any drawing that connects no two points of  $V$  all points form singletons, and thus we trivially get a partition with  $3k > 2k + 1$  sets as in the base case. Otherwise, consider a planar drawing in which two points  $u$  and  $v$  of  $V$  are connected. Refer to Figure 2.14. Moreover, let  $Q$  be the set of all points connected to  $u$  and  $v$  in the drawing. Since  $u$  and  $v$  are connected, they must be of the same color. Assume without loss of generality that  $u = r_i$  and  $v = r_{i+\ell}$ , for some  $i$  and  $\ell > 0$ . Observe that  $Q$  partitions the sequence  $V$  into two alternating sequences  $V_1 = (r_i, b_i, g_i, \dots, r_{i+\ell-1}, b_{i+\ell-1}, g_{i+\ell-1})$  and  $V_2 = (r_1, \dots, g_{i-1}, r_{i+\ell}, \dots, g_k)$  of lengths  $3\ell$  and  $3(k - \ell)$ . We now use the induction hypothesis on both  $V_1$  and  $V_2$ , which gives us that any drawing of  $V_1$ , so in particular the current drawing restricted to  $V_1$ , induces a partition of  $V_1$  into at least  $2\ell + 1$  sets. Similarly, any drawing of  $V_2$  partitions  $V_2$  into at least  $2(k - \ell) + 1$  sets. Since the drawing is planar, it follows that if there are points  $u' \in V_1$  and  $v' \in V_2$  that are connected, these points must both be in  $Q$ ; if they were in any other connected set  $Q'$  the drawings of  $Q$  and  $Q'$  would intersect (see Figure 2.14). Hence,  $Q$  is the union of two sets, one from the partition of  $V_1$  and one from the partition of  $V_2$ . It follows that the drawing induces a partition of size at least  $2\ell + 1 + 2(k - \ell) + 1 - 1 = 2k + 1$ . This completes the proof.  $\square$

**Lemma 2.15.** *The partition  $V_Q$  induces at most  $h - 1$  connections.*

*Proof.* By Observation 2.12 the sequence  $V$  consists of  $h$  groups of three consecutive points  $r_i, b_i, g_i$  that all have a difference color. Furthermore, the pixel polygons in  $Q$  correspond to a drawing of  $V$  in which only points of the same color are connected. Since the pixel polygons in  $Q$  are pairwise disjoint, this drawing is planar, and since  $Q \subset \mathcal{I}$  the points can be drawn on a line with the drawing of the connections (representing the sets  $Q$  themselves) below the line containing  $V$ . So, it now follows from Lemma 2.14 that the partition  $V_Q$  induced by (the drawing corresponding to)  $Q$

contains at least  $2h + 1$  connected components. Since  $V$  consists of  $3h$  elements, there are at most  $3h - (2h + 1) = h - 1$  connections as desired.  $\square$

As argued above, Lemma 2.15 (and its symmetrical counterpart for  $V'_Q$ ) imply that there are at least  $h + 2$  grid polygons in  $Q_R, Q_B,$  and  $Q_G$  that connect polygons in  $U_R, U_B,$  and  $U_G$  on both the left and the right part of the boundary of  $\mathcal{I}$ . Therefore,  $\mathcal{I}$  must have height more than  $2h + 1$ . By definition of  $\mathcal{I}$ , its height is less than  $2h + 1$  so we obtain a contradiction. We therefore obtain the following result:

**Theorem 2.16.** *For all integer  $h > 0$  there exist three regions  $\mathcal{R} = \{R_1, R_2, R_3\}$ , for which there is no valid assignment to grid polygons  $P_1, P_2, P_3$  so that all regions  $R_i \in \mathcal{R}$  have  $d'_H(R_i, P_i) < h$ .*

## 2.6 Conclusion

In this chapter we have shown what Hausdorff distance bounds can be attained when mapping disjoint simply connected regions to the unit grid. We expressed our bounds in terms of the number of regions and obtained different results depending on the shape and size characteristics of the regions, and showed that they are worst-case optimal. The result in Subsection 2.5.1 generalizes a result of Bouts et al. [26] and the result in Subsection 2.5.2 shows that a result by van Goethem et al. [64] cannot be generalized from two to three colors. Our results are slightly more general than we expressed them: for example, the bound for point regions in fact holds for any set of regions that each have constant diameter.

We assumed that our regions all had the same shape and size characteristics. In some cases it is interesting to see what happens in combinations. In particular, suppose we have one general region  $R_0$  and  $m$  point regions  $R_1, \dots, R_m$ ; what Hausdorff bounds can be attained? It turns out that we get a trade-off: we can realize a Hausdorff distance of  $O(\sqrt{m})$  for the point regions and for  $R_0$ , but we can also realize a Hausdorff distance of  $O(1)$  for  $R_0$  but then some point region will have a Hausdorff distance of  $\Theta(m)$ . Figure 2.15 illustrates this. We may map the points to the grid first using the  $O(\sqrt{m})$  bound, and then map  $R_0$ , or we can map the points to the grid in a constant width strip close to the boundary of  $R_0$ . Note that in the former case, we could have left a spacing of three pixels between the mappings of the point regions. Then the point regions still attain the  $O(\sqrt{m})$  bound, while  $d_H(R_0, P_0) = O(1)$  by using the extra space to allow  $P_0$  to reach every necessary place. However,  $d_H(\partial R_0, \partial P_0)$  will still be  $\Theta(\sqrt{m})$ , so we do not improve  $d'_H(R_0, P_0)$ .

While we concentrated on worst-case optimal bounds, our constructive proofs of the upper bounds will often give visually unfortunate output. Also, for a given instance we may not achieve  $O(1)$  Hausdorff distance for  $m$  point,  $\beta$ -fat convex, or convex regions even when constant would be possible for that instance. This leads to the following two open problems. Firstly, can we realize visually reasonable output

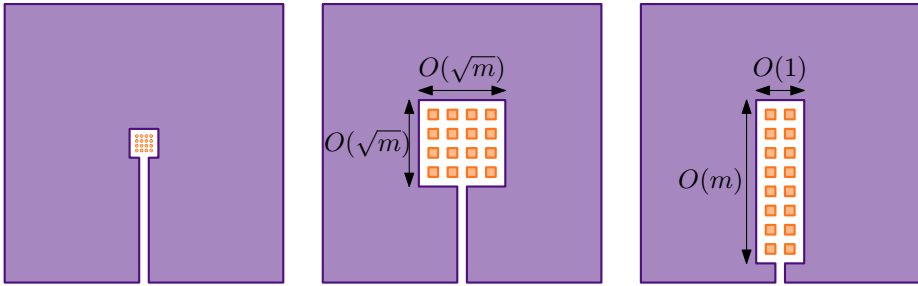


Figure 2.15: Left, an instance with one general region (light blue) and  $m$  point regions. Middle and right, two possible realizations for different Hausdorff bounds.

when this is possible for an instance (and how do we define this)? Secondly, can we realize a Hausdorff distance that is at most a constant factor worse than the best possible for each instance, in polynomial time? For point regions, we answer this question in the next chapter.



## Chapter 3

# Mapping Points to the Grid with Bounded Hausdorff Distance

We consider the problem of representing a set of  $m$  points using disjoint pixels on a unit grid with bounded Hausdorff distance. We prove that deciding if an appropriate set exists is NP-complete. Additionally, we present a constant-factor approximation algorithm with running time  $O(m^2 \log \delta^* / \log m)$ , where  $\delta^*$  is the Hausdorff distance in an optimal solution, as well as an approximation algorithm with a constant additive error and running time  $O(m^2 \delta^{*4} / \log m)$ . This chapter is based on the following publication:

M. Löffler and J. Urhausen. Mapping Points to the Grid with Bounded Hausdorff Distance. *33rd Canadian Conference on Computational Geometry (CCCG)*, pages 47–55, 2021.

### 3.1 Introduction

Similar to Chapter 2, we study a problem in the field of *digital geometry*. Recall that digital geometry concerns itself with the representation of geometric objects using pixels on a grid while preserving geometric properties. Examples are mapping convex regions to a similar-looking orthoconvex set of pixels or mapping lines to chains of pixels that still only intersect at most once. Digital geometry has application in image processing and storage. For related work concerning digital geometry in general and error bounds under the Hausdorff distance in the field of digital geometry in particular, see the introduction of Chapter 2.

This chapter is inspired by two papers: *Mapping Polygons to the Grid with Small Hausdorff and Fréchet Distance*, by Bouts et al. [26] and *Mapping Multiple Regions to the Grid with Bounded Hausdorff Distance* by van der Hoog et al. [80]; the latter is treated

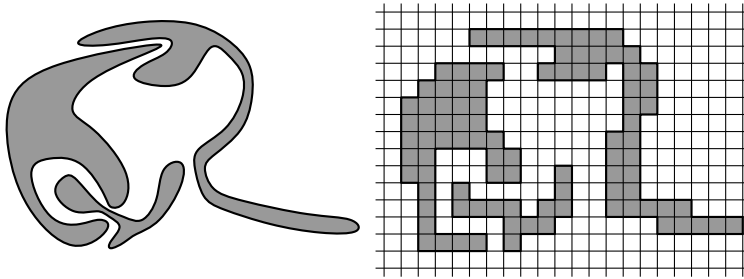


Figure 3.1: An example of a valid mapping of regions to grid polygons.

in Chapter 2. Intuitively the problem discussed in these papers is: for a given set of regions, find a set of pixels from the unit grid that best represents the input (see Figure 3.1). We use the Hausdorff distance to measure similarity. Thus, for each point, we determine a single pixel to represent that point as using multiple pixels to represent a single point would in no case reduce the minimal attainable Hausdorff distance.

In the following we use the notation  $d'_H(R, P)$  for the maximum of the Hausdorff distance between the sets  $R$  and  $P$  and the Hausdorff distance between their boundaries  $\partial R$  and  $\partial P$ . Amongst others, Bouts et al. [26] show that for a given connected region  $R$ , one can find a simply connected grid polygon  $P$  in the unit grid such that the Hausdorff distance  $d'_H(R, P)$  is at most a constant. Note that this result holds, no matter the resolution of  $R$ . On the other hand they show that, given a single region  $R$ , it is NP-hard to find a grid polygon  $P$  that minimizes the Hausdorff distance  $d'_H(R, P)$ . Here, we extend their results to the problem of mapping multiple point regions to the grid.

### 3.1.1 Computation

When considering the question of efficiency, we are faced with some additional modeling questions. The two main ones are:

1. How do we locate input features (vertices or edges) on the grid?
2. How do we compactly represent sets of pixels that correspond to output regions?

Regarding question (1), we note that traditionally, geometric algorithms are analysed in the *real RAM* computation model. In this model, one may work with arbitrary real numbers, but certain natural operations are not available; in particular the *floor* operation is known to be problematic [13]. In the context of digital geometry, where we have a natural underlying grid, and the whole objective is to map objects to a grid, the restriction on the floor operation has to be reevaluated. In this chapter, we will slightly deviate from the real RAM model and assume that the input coordinates are

all polynomial in the input size and that the floor function is available. Note that under this size restriction, if desired, the floor function on the input coordinates can also be implemented in logarithmic time on a real RAM using binary search.

Regarding question (2), we note that when mapping large regions (in size, not in description complexity) to a grid, an explicit representation of the output listing precisely which pixels are part of a set would necessarily be large as well, and may be unrelated to the input complexity. Alternatively, one could compactly represent output regions by providing only the coordinates of *vertices* and interpolating boundary edges onto the grid. Given the complexity of mapping lines to the grid, however, it is not clear how to do this in a consistent way. In this chapter, we make a first step towards understanding the computational aspect of the question by focusing on *point* regions. For a single point region and constant Hausdorff distance, the output is necessarily a set of only a constant number of pixels, and thus we avoid the issue.

To summarize, in this chapter, we make the following assumptions:

- There is a polynomial upper bound on the coordinate sizes of the input points, that is, there is a polynomial function  $f : \mathbb{R} \rightarrow \mathbb{R}$ , such that, if the input is a set  $\mathcal{R}$  of  $m$  points in  $\mathbb{R}^2$ , for every point  $R \in \mathcal{R}$  we have  $-f(m) < x_R < f(m)$  and  $-f(m) < y_R < f(m)$ .
- We have access to a *floor* operation, which can provide us with the integer part of any real number in the range  $[-f(m), f(m)]$ .

### 3.1.2 Related Work

For each point on the grid, placing unit squares in that grid within Hausdorff distance at most  $\sqrt{2}$  equals placing those squares in the grid such that they intersect the points. Furthermore, the problem of placing unit squares such that they intersect points is dual to the problem of placing points in unit squares. The problem of placing points in squares such that the points are not too close to each other has been introduced under the name of *distant representatives* [60] and was later also studied in the context of data *imprecision* [94]. Fiala et al. [60] prove that the problem is NP-hard (both for disk and square regions), and Cabello [40] proposes a constant-factor approximation algorithm.

We note that our problem is essentially different, since for us, valid placements are restricted to a *discrete* set of points (the unit grid). Neither the hardness proof nor the algorithmic result carry over directly to this discrete setting.

### 3.1.3 Contribution

In this chapter, we study the computational question of mapping point sets to disjoint pixels on a unit grid with small Hausdorff distance, as visualized in Figure 3.2. In Chapter 2 we observed that in general the solution constructed with our algorithms

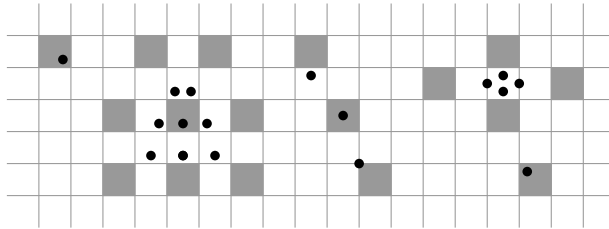


Figure 3.2: An example of a valid mapping of points to pixels on the grid.

might yield a “visually unfortunate” output. Formally, the algorithm might yield a solution with high Hausdorff distance, even in the case where the optimal solution has constant Hausdorff distance. In Section 3.2, we show that finding the solution with minimum Hausdorff distance to a given set of regions is NP-complete, even if the regions are just points. Then, in Section 3.3, we present an approximation algorithm for points that produces a solution with Hausdorff distance at most  $2\sqrt{2}(\lceil\delta^*\rceil + 1) \leq 6\sqrt{2}\delta^*$  and has a running time of  $O(m^2 \log \delta^* / \log m)$ , where  $\delta^*$  is the Hausdorff distance in an optimal solution. Finally, we present a second algorithm which produces a solution with Hausdorff distance at most  $\lceil\delta^*\rceil + \sqrt{2}$  in  $O(\delta^{*4}m^2 / \log m)$  time.

### 3.1.4 Notation and Definitions

We denote by  $\Gamma$  the (infinite) unit grid in two dimensions, whose unit squares are referred to as *pixels*.

Let  $\mathcal{R} = \{R_1, R_2, \dots, R_m\}$  be a set of  $m$  points in the plane. In this chapter, we treat the problem of how to assign a pixel  $P_i \in \Gamma$  to each point  $R_i \in \mathcal{R}$  such that different pixels do not meet in any edge or vertex of the grid. We call the set  $\mathcal{P} = \{P_1, P_2, \dots, P_m\}$  of such pixels a *valid mapping* for  $\mathcal{R}$ . Our goal is to find a valid mapping  $\mathcal{P}$  that minimizes  $\max_{i \in \{1, \dots, m\}} \{d_H(R_i, P_i)\}$ . See Figure 3.2 for an example.

Note that in contrast to [26] and Chapter 2, we disregard the Hausdorff distance between the boundaries  $d_H(\partial R_i, \partial P_i)$  because we have  $d_H(\partial R_i, \partial P_i) = d_H(R_i, P_i)$ , for convex  $R_i$  and  $P_i$ .

## 3.2 NP-Completeness

Bouts et al. [26] proved that for a single simply connected region  $R$  it is NP-complete to test if there is a grid-polygon within Hausdorff distance  $1/2$ . We extend this result to multiple point-regions. Formally, we show that for a set of points  $\mathcal{R}$  it is NP-complete to test if there is a valid mapping within Hausdorff distance  $\sqrt{2}$ . Our

proof is inspired both by the construction of Bouts et al. [26] and the proof by Fiala et al. [60] for a similar problem in a continuous setting.

We first prove containment in NP. For each point there are at most four options to place the corresponding pixel. An oracle can guess the correct placement and then just has to test that no two pixels share a common grid vertex.

We now show that the problem is NP-hard. We reduce from the NP-complete problem monotone rectilinear planar 3-SAT [20].

**Rectilinear Monotone Planar 3-SAT.** Input: a 3-SAT formula with only all positive or all negated variables per clause, embedded as a graph with rectilinear, non-crossing edges. The set of vertices consists of variable-, split- and clause-vertices; variable-vertices are drawn on a horizontal line that no edge crosses; clause-vertices for positive (negative) clauses are drawn above (below) this line; clauses are connected with the variables they contain with an edge or a path of edges and split-vertices. Output: “Yes” if there exists an assignment for the variables that satisfies the formula, “No” otherwise.

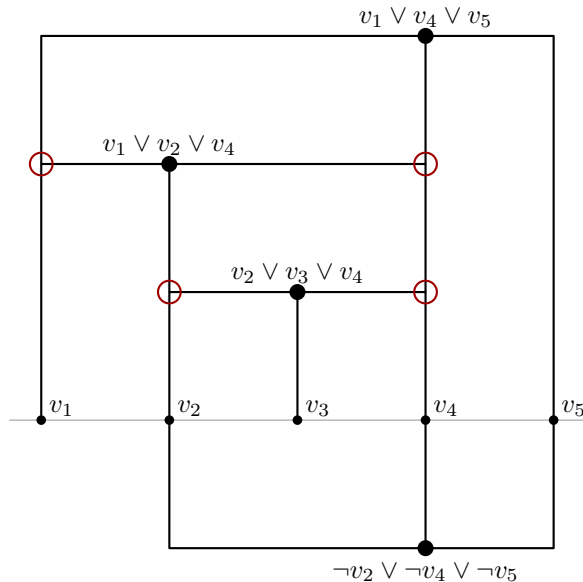


Figure 3.3: An example of the embedded formula  $(v_2 \vee v_3 \vee v_4) \wedge (v_1 \vee v_2 \vee v_4) \wedge (v_1 \vee v_4 \vee v_5) \wedge (\neg v_2 \vee \neg v_4 \vee \neg v_5)$ . The split vertices are highlighted in red.

Such a 3-SAT formula as an embedding of a graph is illustrated in Figure 3.3. Without loss of generality we can assume that the embedded graph has the following additional properties: each variable-vertex  $v$  has degree at most two and the incident

edges are vertical at  $v$ , split-vertices  $w$  have degree three and only one incident edge is horizontal at  $w$ , and for each variable  $a$ , all split-vertices corresponding to  $a$  are vertically aligned with the variable-vertex  $v_a$  of  $a$ . It is easy to realize these additional properties.

For a given monotone rectilinear planar 3-SAT instance that is embedded as described above, let  $\mathcal{G}$  be a drawing of the embedding. Without loss of generality we assume that  $\mathcal{G}$  is drawn on the unit grid and that the horizontal line containing all variables is the  $x$ -axis. We scale  $\mathcal{G}$  such that each vertex is on an even grid vertex  $(2x, 2y)$  and such that the distance between any two vertices, between any vertex and any bend, and between any two non-incident edges is at least 10.

**Construction.** We create a set of points  $\mathcal{R}$ . We only place points on grid vertices. In the end, we ask the question whether one can place pixels within Hausdorff distance  $\sqrt{2}$  from the points of  $\mathcal{R}$ , that is, we ask the question if for each point in  $\mathcal{R}$ , we can choose one of the four adjacent pixels so that no two chosen pixels of different points share a common vertex. We say a point  $R_i$  has a *top-left* (*top*, *top-right*, ...) *pixel* if  $P_i$  is to the top-left (*top*, *top-right*, ...) of  $R_i$ .

These following two observations are the main tools for the construction of the gadgets, depicted in Figure 3.4. When two horizontally aligned points are at distance 1, the leftmost point has a left pixel and the rightmost point has a right pixel. For two horizontally aligned points at distance 2, if the leftmost point has a right pixel, the rightmost point has a right pixel, too. Symmetrically, if the rightmost point has a left pixel, the leftmost point has a left pixel. This is symmetric for vertically aligned points.

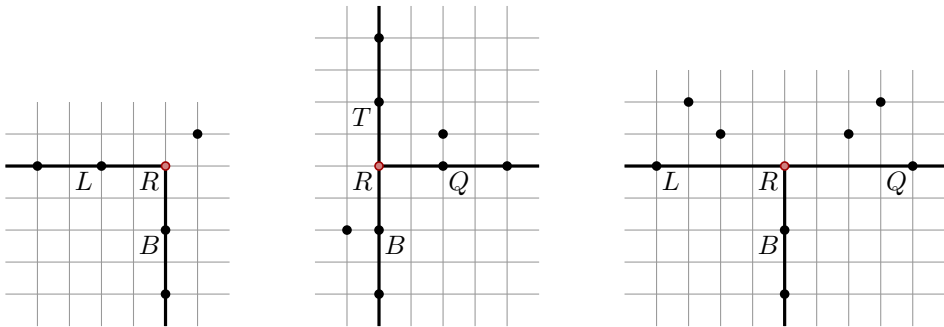


Figure 3.4: The bend, split and clause gadgets. The point  $R$  is highlighted in each gadget.

**Variable.** We first place a point  $R \in \mathcal{R}$  on each even grid vertex that is intersected by the drawing. For each variable  $a$  in the 3-SAT instance, there is a point  $R_a = (2x, 0)$  in  $\mathcal{R}$  where the variable-vertex  $v_a$  is drawn. We call that point  $R_a$  the *indicator* of  $a$ .

Intuitively, if  $R_a$  has a bottom (top) pixel,  $a$  is true (false). If the point  $(2x, 2)$  has a bottom (top) pixel we say it has a pixel *toward the variable* (*away from the variable*). Symmetrically, if the point  $(2x, -2)$  has a top (bottom) pixel we say it has a pixel *toward the variable* (*away from the variable*). This concept propagates throughout the points corresponding to  $a$ .

**Bend.** Let  $R = (2x, 2y) \in \mathcal{R}$  be a point at the corner of a bend of an edge  $e$ . We assume  $e$  connects a vertex to the left of  $R$  with a vertex below  $R$ . The other cases are symmetric. Thus, the points  $L = (2x - 2, 2y)$  and  $B = (2x, 2y - 2)$  are in  $\mathcal{R}$ . We add another point  $(2x + 1, 2y + 1)$  to  $\mathcal{R}$ . Now, if  $L$  has a right pixel,  $B$  has a bottom pixel and if  $B$  has a top pixel,  $L$  has a left pixel.

**Split.** Let  $R = (2x, 2y) \in \mathcal{R}$  be a point at a split-vertex. We assume that the horizontal edge incident to  $R$  is to its right and that the split vertex is above the  $x$ -axis and therefore its corresponding variable-vertex. Thus, the points  $T = (2x, 2y + 2)$ ,  $Q = (2x + 2, 2y)$  and  $B = (2x, 2y - 2)$  are in  $\mathcal{R}$ . The other cases are symmetric. We add the points  $(2x + 2, 2y + 1)$  and  $(2x - 1, 2y - 2)$  to  $\mathcal{R}$ . If  $T$  has a bottom pixel or if  $Q$  has a left pixel,  $B$  has a bottom pixel. That is, if a point on the top or right edges has a pixel toward the variable, the points on the bottom edge also have pixels toward the variable. Inversely, if the bottom edge has pixels away from the variable, both top and right edges also have pixels away from the variable.

**Clause.** Let  $R = (2x, 2y) \in \mathcal{R}$  be a point at a clause-vertex. We call  $R$  the *clause-point*. We assume the three edges connect to the left, right and bottom of  $R$  respectively, otherwise the situation is symmetric. As the distance between two gadgets is at least 8, the points  $(2x - 2, 2y)$ ,  $L = (2x - 4, 2y)$ ,  $(2x + 2, 2y)$ ,  $Q = (2x + 4, 2y)$  and  $B = (2x, 2y - 2)$  are in  $\mathcal{R}$ . We move the points  $(2x - 2, 2y)$  and  $(2x + 2, 2y)$  to  $(2x - 2, 2y + 1)$  and  $(2x + 2, 2y + 1)$  and add two points  $(2x - 3, 2y + 2)$  and  $(2x + 3, 2y + 2)$  to  $\mathcal{R}$ . It follows that if the clause-point  $R$  has a top-left pixel then  $L$  has a left pixel, if  $R$  has a top-right pixel then  $Q$  has a right pixel, and if  $R$  has a bottom pixel then  $B$  has a bottom pixel. That means that the points on at least one of the incident edges have pixels toward the variable.

Starting from the embedded 3-SAT formula shown in Figure 3.3, we show the points  $\mathcal{R}$  and a valid mapping  $\mathcal{P}$  in Figure 3.5.

**Proof of Correctness.** Let  $A$  be an assignment of variables to  $\{\text{true}, \text{false}\}$  such that the 3-SAT formula is satisfied. For each variable  $a$ , if  $a$  is true, first, we give the indicator  $R_a$  a bottom pixel. Second, we give each point  $R \in \mathcal{R}$  that is on an edge corresponding to  $a$  a pixel toward (away from) the variable, if  $p$  is above (below) the  $x$ -axis. For each variable assigned the value false we do the inverse. As in each positive (negative) clause there is a variable that is assigned to true (false),

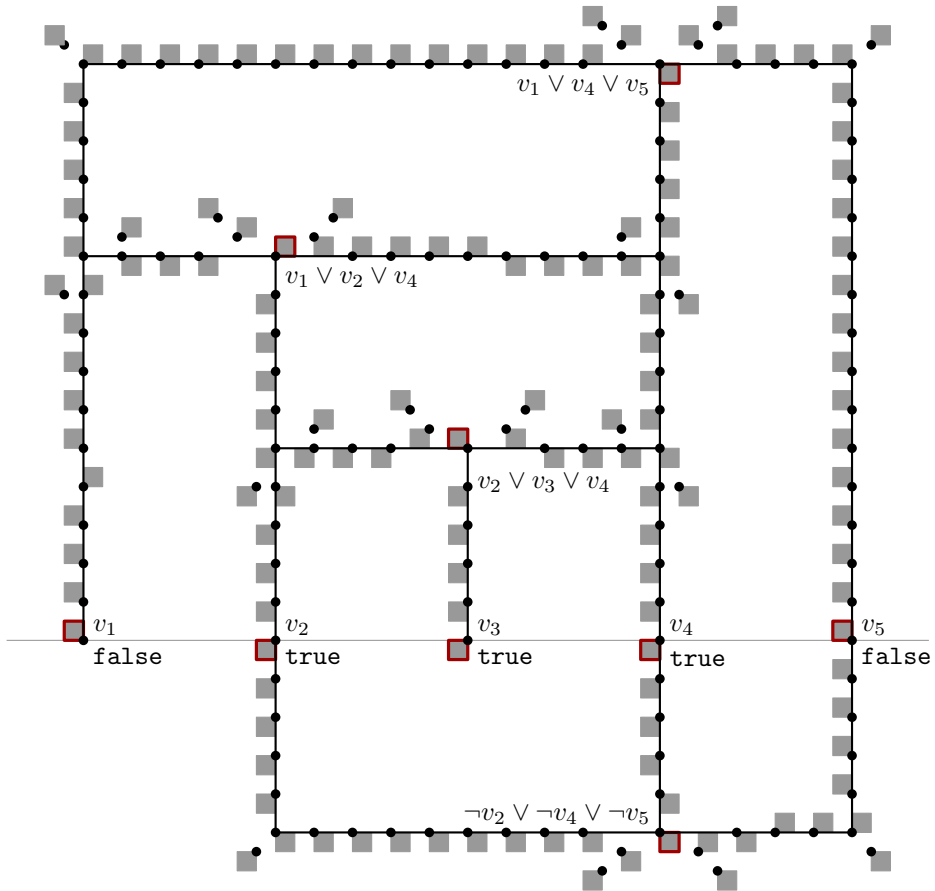


Figure 3.5: The complete construction of the NP-hardness reduction. The pixels corresponding to indicators or to clause-points are highlighted.



each clause-point can place its pixel in one of the four adjacent spots. So overall there is a valid mapping such that the Hausdorff distance between a point and its corresponding pixel is at most  $\sqrt{2}$ .

Inversely, let there be a set of pixels such that the Hausdorff distance between any point and its corresponding pixel is at most  $\sqrt{2}$ . For each variable  $a$ , if the indicator  $R_a$  has a bottom (top) pixel, we set  $a$  to true (false). We now prove that in each positive (negative) clause there is a variable that is assigned to true (false). Let  $c$  be a positive clause such that the incident edges are on the left, right and bottom of the clause point  $R$  of  $c$ . The other cases are symmetric. If  $R$  has a top-left (top-right, bottom) pixel, we know that the points on the left (right, bottom) edge have pixels toward the variable. Let  $e$  be an edge incident to  $R$  whose points have pixels toward the variable. If  $e$  connects to a split-vertex  $w$ , the points on the vertical edge  $e'$  incident at the bottom at  $w$  also have pixels toward the variable. We then set  $e = e'$ . This repeats until we have an edge  $e$  that connects to the variable-vertex of  $a$ . It follows that the indicator  $R_a$  has a bottom pixel and  $a$  has been assigned the value true. The theorem follows.

**Theorem 3.1.** *If  $\mathcal{R}$  is a set of  $m$  points, it is NP-complete to decide whether there exists a valid mapping such that for each point  $R_i \in \mathcal{R}$  with corresponding pixel  $P_i$ , we have  $d_H(R_i, P_i) \leq \sqrt{2}$ .*

### 3.3 Approximation Algorithms

We now turn our attention to approximation. We start by making some observations. Clearly, the optimal Hausdorff distance  $\delta^*$  for any instance is at least  $\frac{1}{2}\sqrt{2}$ , since the distance is taken between a point and (at least one) unit square. Therefore, a constant *additive* approximation in this case automatically translates to a constant *multiplicative* approximation. We also note that, due to the discrete nature of the output, we cannot hope to achieve a polynomial-time approximation scheme, that is, a  $(1 + \varepsilon)$ -approximation.

To illustrate the complexity of the problem, in Section 3.3.1 we first discuss some natural ideas which do *not* lead to a working approximation. In Section 3.3.2, we then present an algorithm that achieves a Hausdorff distance of at most  $2\sqrt{2}\lceil\delta^*\rceil + 2\sqrt{2}$ . In Section 3.3.3 we show how to improve the approximation to  $\lceil\delta^*\rceil + \sqrt{2}$ , at the cost of a slower runtime.

#### 3.3.1 A First Attempt

As discussed in the introduction, Cabello [40] presents a constant-factor approximation algorithm for placing  $n$  points into respective discs or squares. A first approach could be to dualize our problem and directly run their algorithm to place a set of

points—however, we would have no guarantee the points are placed at integer coordinates. We would have to snap the points to the grid before translating them back to squares. The Hausdorff distance itself would only increase by at most  $\frac{1}{2}\sqrt{2}$  in this way, which would still result in a constant-factor approximation, albeit with a slightly higher constant. However, the snapping procedure could also result in touching or even overlapping pixels, so the solution would not necessarily be valid.

Another approach would be to find a subdivision of the grid into cells such that for each cell all the points contained in it can be assigned separate pixels in that cell. Formally, let  $\Gamma_k$  be a coarsening of the grid  $\Gamma$  whose cells have  $k \times k$  pixels. We call these cells *superpixels*. The idea is to find a coarsening  $\Gamma_k$  where each superpixel contains at most  $\lfloor k/2 \rfloor^2$  points. Then, by assigning to each point a pixel within their superpixel, the Hausdorff distance between the points and their pixels would be at most  $k\sqrt{2}$ . Therefore, if the minimal size of the cells depends only on the Hausdorff distance between the points and an optimal valid mapping, this approach could lead to an approximation algorithm. The following lemma proves that this approach does not work. The minimal size of such a superpixel can be arbitrary large, even if the Hausdorff distance is a small constant for an optimal valid mapping.

**Lemma 3.2.** *There is a set of points  $\mathcal{R}$  and a point  $R \in \mathcal{R}$ , such that (1) there is a valid mapping  $\mathcal{P}$  within Hausdorff distance at most 3; (2) for any superpixel with side length  $s$  that contains  $R$  and at most  $\lfloor \frac{s}{2} \rfloor^2$  points from  $\mathcal{R}$ , we have  $s \geq \sqrt{m-1} + 2$ .*

*Proof.* A pixel is a set  $[i, i+1] \times [j, j+1]$ , for integers  $i, j$ . As shown in Figure 3.6, for an integer  $n \leq 1$ , we define  $R = (\frac{1}{4}, \frac{1}{4})$  and the set of points

$$\mathcal{R} = \{R\} \cup \left\{ \left( h \left( 2i + \frac{1}{2} \right), g \left( 2j + \frac{1}{2} \right) \right) \mid i, j \in \{0, \dots, n\}; g, h \in \{-1, 1\} \right\}.$$

We first prove that there is a valid mapping  $\mathcal{P}$  within Hausdorff distance at most 3. This valid mapping is also depicted in Figure 3.6. The point  $R$  is mapped to the pixel centered at  $(\frac{1}{2}, \frac{1}{2})$ . Each point  $(2i + \frac{1}{2}, \frac{1}{2})$  for  $i \in \{0, \dots, n\}$  is mapped to the pixel centered at  $(2i + \frac{1}{2} + 2, \frac{1}{2})$ . Each other point  $(h(2i + \frac{1}{2}), g(2j + \frac{1}{2}))$  for  $i, j \in \{0, \dots, n\}$  and  $g, h \in \{-1, 1\}$ , is mapped to  $(h(2i + \frac{1}{2} + 1), g(2j + \frac{1}{2} + 1))$ . The Hausdorff distance between the points and their respective pixels is  $\max\{\sqrt{26}/2, 3\sqrt{2}/2\} \simeq 2.55 < 3$ .

We move on to the second part of the proof. The set  $\mathcal{R}$  contains  $m = 4n^2 + 1$  points. Let  $S$  be a superpixel containing  $R$  and let  $(a, b), (-c, b), (-c, -d)$ , and  $(a, -d)$  be the coordinates for the four vertices of  $S$ , for  $a, b \geq 1$  and  $c, d \geq 0$ . The side length of the superpixel  $S$  is  $s = a + c = b + d$ . If  $s \leq 2n + 1$ ,  $S$  contains  $1 + \lceil a/2 \rceil \lceil b/2 \rceil + \lceil c/2 \rceil \lceil b/2 \rceil + \lceil c/2 \rceil \lceil d/2 \rceil + \lceil a/2 \rceil \lceil d/2 \rceil > \lfloor s/2 \rfloor^2$  points. Thus, a superpixel with side length  $s$  containing at most  $\lfloor s/2 \rfloor^2$  points has at least side length  $2n + 2 = \sqrt{m-1} + 2$ .  $\square$

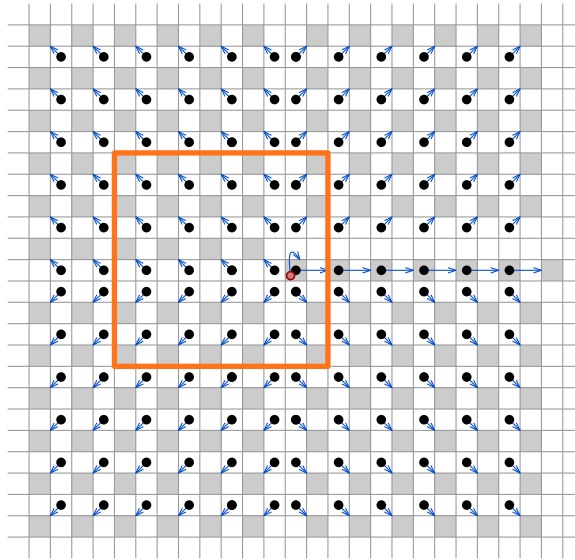


Figure 3.6: Any superpixel with side length  $s$  containing the red point  $R$  contains more than  $\lfloor s/2 \rfloor^2$  points. For example, the orange superpixel has side length 10 and contains 26 points. A valid mapping with Hausdorff distance  $\sqrt{26}/2$  is indicated.

### 3.3.2 A Constant-Factor Approximation Algorithm

We present an algorithm that, for a given set of  $m$  points  $\mathcal{R}$  in  $\mathbb{R}^2$  determines a valid mapping  $\mathcal{P}$ , such that the Hausdorff distance between a point and its corresponding pixel is at most  $2\sqrt{2}(\lceil \delta^* \rceil + 1)$ , for  $\delta^*$  being the minimal possible Hausdorff distance between  $\mathcal{R}$  and any valid mapping  $\mathcal{P}$ .

For a coarsening  $\Gamma_k$  of the grid  $\Gamma$ , let  $\mathcal{S}_k$  be the set of superpixels that either contain a point in  $\mathcal{R}$  or are adjacent to a superpixel that does.

**Observation 3.3.** *Let  $k \geq \delta^*$  and  $k \in \mathbb{N}$  even. Then, in an optimal solution, for each point  $R_i \in \mathcal{R}$  in a superpixel  $S \in \mathcal{S}_k$ , the pixel  $P_i$  is in  $S$  or in one of the eight superpixels adjacent to  $S$ . Additionally, each superpixel contains at most  $(k/2)^2$  pixels that are disjoint.*

Building on that idea, we create the following test  $f(\cdot)$ . For an even number  $i \in \mathbb{N}$ , we want  $f(i) = \text{true}$  if and only if for each coarsening  $\Gamma_i$ , there exists an assignment  $g : \mathcal{R} \rightarrow \mathcal{S}_i$  of points to superpixels such that:

1.  $\forall R \in \mathcal{R}$ ,  $g(R)$  is the superpixel containing  $R$  or one of the eight adjacent ones;
2.  $\forall S \in \mathcal{S}_i$ , there are at most  $(i/2)^2$  points  $R$  with  $g(R) = S$ .

We call such an assignment  $g$  a *correct assignment*. Otherwise, if for each coarsening  $\Gamma_i$  no correct assignment can be found, we want  $f(i) = \text{false}$ . Note that for some

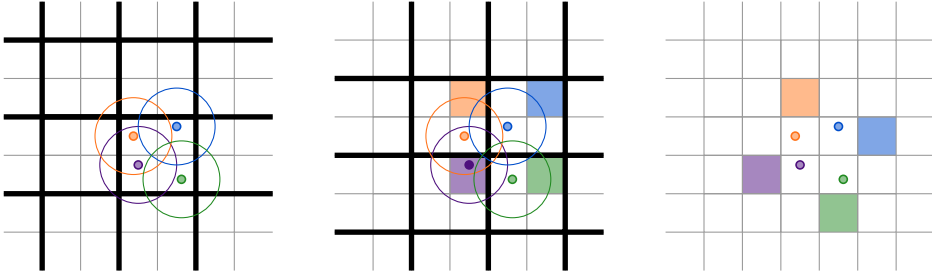


Figure 3.7: Depending on the coarsening the test  $f(1)$  can be either true or false here, as indicated by the circles of radius 1. The right figure shows the valid mapping with minimal Hausdorff distance. Since  $\delta^* \simeq 1,68$ , the test  $f(2)$  returns true.

values  $i$ , it depends on the “origin” of the coarsening if a correct assignment exists, as depicted in Figure 3.7. In that case,  $f(i)$  can either be true or false. We define  $f(0) = \text{false}$ .

**Binary Search.** We use exponential and binary search to find an even number  $i \in \mathbb{N}$  with  $f(i) = \text{true}$  and  $f(i-2) = \text{false}$ . We start at  $i = 2$ . We iterate calculating  $f(i)$ : if  $f(i) = \text{false}$ , we double  $i$  and continue, else we stop. Then we binary search normally between the last  $i$  and  $i/2$ . Note that from Observation 3.3 we get  $f(i) = \text{true}$ , for  $i \geq \delta^*$ . Therefore, this binary search algorithm results in a number  $i' \leq \lceil \delta^* \rceil + 1$  and has a running time of  $O(F \times \log \delta^*)$ , where  $F$  the the time to run the test  $f(\cdot)$ .

**Test.** The calculation of  $f(i)$  proceeds as follows. We use a flow algorithm [73] to determine if a correct assignment  $g$  exists. We choose a coarsening  $\Gamma_i$  and create a directed acyclic graph  $G = (V, E)$  as illustrated in Figure 3.8. We set  $V = \{s, t\} \cup \{S_{in}, S_{out} \mid S \in \mathcal{S}_i\}$  as the set of vertices. We define  $(a, b, c) \in E$  as the edge between the vertices  $a \in V$  and  $b \in V$  with capacity  $c \in \mathbb{N} \cup \{\infty\}$ . We set  $E = \{(s, S_{in}, |S \cap \mathcal{R}|) \mid S \in \mathcal{S}_i\} \cup \{(S_{in}, S'_{out}, \infty) \mid S, S' \in \mathcal{S}_i \wedge (S = S' \vee S \text{ adjacent to } S')\} \cup \{(S_{out}, t, (i/2)^2) \mid S \in \mathcal{S}_i\}$  as the set of edges. Intuitively, the capacity of the edge  $(s, S_{in})$  from  $s$  to the superpixel  $S$  equals the number of points in  $S$ , and the capacity of the edge  $(S_{out}, t)$  from the superpixel  $S$  to  $t$  is the number of disjoint pixels that can be placed in  $S$ .

We then calculate a maximal flow from  $s$  to  $t$ . We can assume that the flow in each edge is a natural number. As  $|E| \in O(|V|)$  and  $|V| \in O(m)$ , the flow algorithm runs in  $O(m^2/\log m)$  time [103]. If  $G$  admits a flow from  $s$  to  $t$  with flow rate  $m$ , the flow induces a correct assignment  $g$  as follows: we repeat the following for every superpixel  $S \in \mathcal{S}_i$ . Let  $\{S_1, \dots, S_9\} = \{S' \mid S' \in \mathcal{S}_i \wedge (S = S' \vee S \text{ adjacent to } S')\}$  be the set containing  $S$  and the superpixels adjacent to  $S$ . Let  $U_1 \cup \dots \cup U_9 = S \cap \mathcal{R}$  be any partition of the points in  $S$ , where, for  $j, k \in \{1, \dots, 9\}$ , we have  $j \neq k \implies U_j \cap U_k = \emptyset$  and  $|U_j|$  is the flow from  $S_{in}$  to  $(S_j)_{out}$ . For each  $j \in \{1, \dots, 9\}$  and each

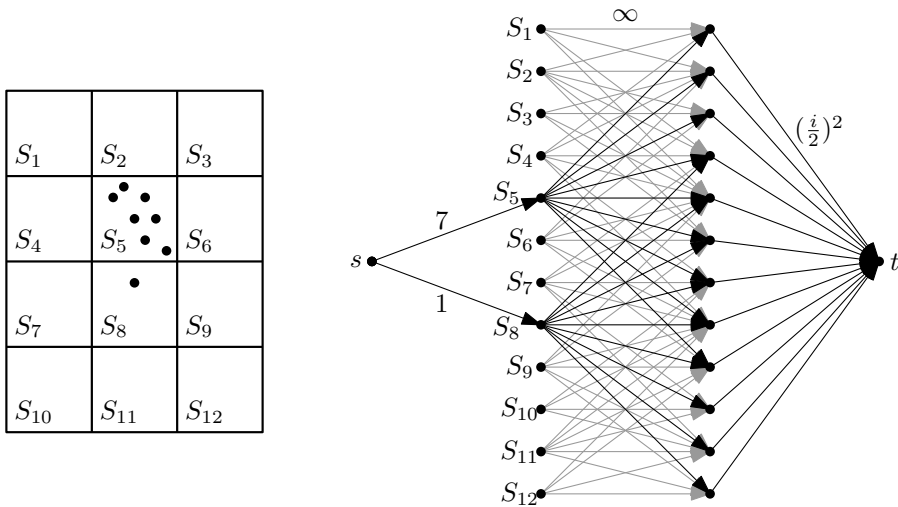


Figure 3.8: A set  $\mathcal{R}$  of 8 points and the corresponding graph  $G$  produced by the algorithm in Section 3.3.2. The edges are annotated with their respective capacities. Edges with capacity 0 are omitted; edges starting at  $(S_i)_{in}$  are grayed out if  $S_i$  is empty.

point  $R \in U_j$ , we set  $g(R) = S_j$ . This results in a correct assignment  $g$ .

Thus, if  $G$  admits a flow from  $s$  to  $t$  with flow rate  $m$ , we set  $f(i) = \text{true}$ . Otherwise  $f(i) = \text{false}$ . This test performs as requested and has a running time of  $O(m^2 / \log m)$ .

**Placing the Pixels.** Let  $i'$  be the even number that is the result from the binary search, that is,  $f(i') = \text{true}$  and  $f(i' - 2) = \text{false}$ . Let now  $\Gamma_{i'}$  be a coarsening and let  $g$  be a correct assignment, calculated by the test  $f(i')$ . We place the pixels  $\mathcal{P}$  as shown in Figure 3.9: for each superpixel  $S \in \mathcal{S}_{i'}$ , let  $\{R_1^S, R_2^S, \dots\} = \{R \in \mathcal{R} \mid g(R) = S\}$  be the points assigned to  $S$ . We set the pixel corresponding to  $R_j^S$  as  $S \left[ \left( (j - 1) \bmod \frac{i'}{2} \right) + 1, \lceil \frac{2j}{i'} \rceil \right]$ , where  $S[x, y]$  is the pixel that is the  $(2x)^{\text{th}}$  from the left and  $(2y)^{\text{th}}$  from the bottom within  $S$ . That way no two pixels in  $\mathcal{P}$  touch and for each point  $R$  its corresponding pixel is in the superpixel assigned to  $R$ . It follows that the Hausdorff distance between a point and its corresponding pixel is at most  $2\sqrt{2}i'$ . Since  $i' \leq \lceil \delta^* \rceil + 1$  from the binary search, and since  $i'$  is an even number and  $\delta^* \geq \frac{1}{2}\sqrt{2}$ , we get:

**Theorem 3.4.** *Given a set of  $m$  points  $\mathcal{R}$ , we can determine a valid mapping  $\mathcal{P}$  such that for each point  $R_i$  with corresponding pixel  $P_i$ , we have  $d_H(R_i, P_i) \leq \min\{2\sqrt{2}(\lceil \delta^* \rceil + 1), 8\delta^*\}$ , in  $O(m^2 \log \delta^* / \log m)$  time.*

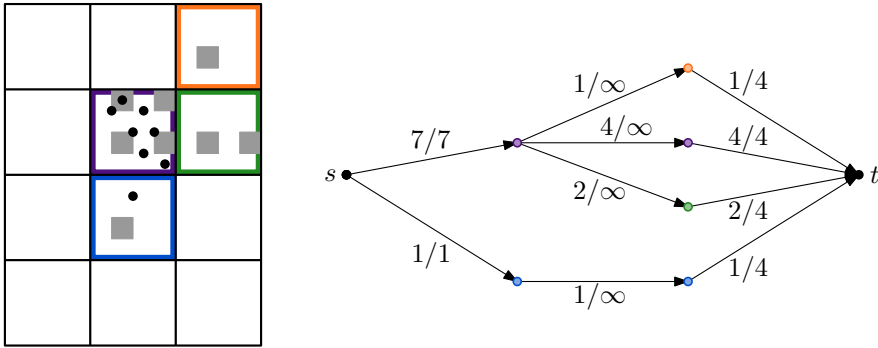


Figure 3.9: A set  $\mathcal{R}$  of eight points and the graph  $G$  with a maximal  $s$ - $t$ -flow where edges with flow 0 are omitted, produced by the algorithm in Section 3.3.2. The superpixels and their respective vertices in  $G$  are color matched. The pixels  $\mathcal{P}$  induced by the flow are show in gray.

The approximation factor of 8 is determined as follows: for large  $\delta^*$ , the additive constant of  $2\sqrt{2}$  is negligible. Inversely, the ratio  $d_H(R_i, P_i)/\delta^*$  is largest for small  $\delta^*$ . The two extreme cases to consider are:  $\delta^* = \frac{1}{2}\sqrt{2}$  and  $i' = 2$ , and  $\delta^* = 2 + \varepsilon$  and  $i' = 4$  for  $0 < \varepsilon \ll 1$ . In the first case, we have  $d_H(R_i, P_i) \leq 4\sqrt{2} = 8\delta^*$ . In the second case, we have  $d_H(R_i, P_i) \leq 8\sqrt{2} \leq 4\sqrt{2}\delta^*$ .

### 3.3.3 An Algorithm with Constant Additive Error

Contrary to the constant-factor algorithm presented in the previous section, we now present an approximation algorithm with *only* a constant additive error: that is, for a given set  $\mathcal{R}$ , we determine a valid mapping  $\mathcal{P}$ , with  $d_H(R_i, P_i) \leq \delta^* + c$  for each  $R_i \in \mathcal{R}$ , where  $c$  is a constant and  $\delta^*$  is the minimal possible Hausdorff distance between  $\mathcal{R}$  and any valid mapping  $\mathcal{P}$ . The algorithm uses similar ideas to the algorithm in Section 3.3.2. Let  $\Gamma_2$  be a coarsening, that is, each superpixel only contains four pixels forming a square. It follows that when placing a pixel in each superpixel in the same position, the pixels are disjoint. We define  $d_H^*(R, S) = \min_{\text{pixel } P \in S} d_H(R, P)$  as the Hausdorff distance between the point  $R \in \mathcal{R}$  and its closest pixel in the superpixel  $S$ . For  $i \in \mathbb{N}$ , let  $\mathcal{S}_i$  be the set of superpixels  $S$ , such that there is a point  $R \in \mathcal{R}$  with  $d_H^*(R, S) \leq i$ . Note that here, the size of the set  $\mathcal{S}_i$  is in  $\Theta(i^2m)$  instead of just  $\Theta(m)$  like in Section 3.3.2.

**Observation 3.5.** *Let  $k \geq \delta^*$  and  $k \in \mathbb{N}$ . For a given valid mapping  $\mathcal{P}$  that minimizes the Hausdorff distance, for each point  $R_i \in \mathcal{R}$ , the pixel  $P_i \in \mathcal{P}$  is in a superpixel  $S \in \mathcal{S}_k$ , with  $d_H^*(R_i, S) \leq k$ . Additionally, each superpixel contains at most one pixel that is mapped to a point.*

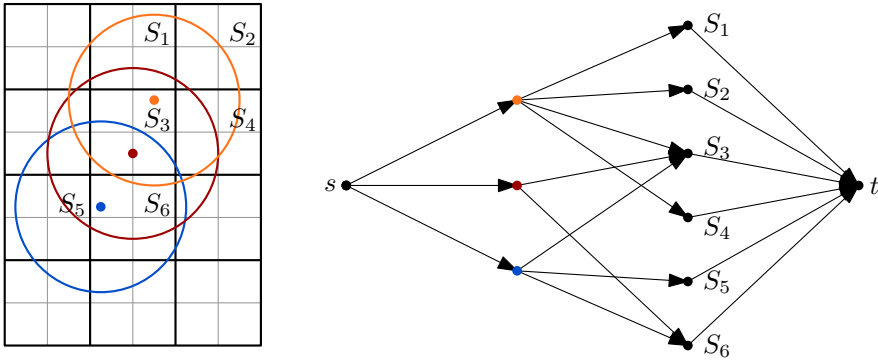


Figure 3.10: A small set  $\mathcal{R}$  of 3 points and the graph  $G = (V, E)$  produced by the algorithm in Section 3.3.3 for  $i = 2$ . All edges have a capacity of 1. The points in  $\mathcal{R}$  and their respective vertices in  $G$  are color matched.

The observation leads us to create the following test  $f(\cdot)$ : we want  $f(i) = \text{true}$ , if there exists an assignment  $g : \mathcal{R} \rightarrow \mathcal{S}_i$  of points to superpixels such that:

1.  $\forall R \in \mathcal{R}, d_{\text{H}}^*(R, g(R)) \leq i$ ;
2.  $\forall S \in \mathcal{S}_i$ , there is at most one  $R$  with  $g(R) = S$ .

We again call such an assignment  $g$  a *correct assignment*. Otherwise, we want  $f(i) = \text{false}$ . Note that  $f(0) = \text{false}$ .

**Binary Search.** Similarly to Section 3.3.2, we use exponential and binary search to find a number  $i \in \mathbb{N}$  with  $f(i) = \text{true}$  and  $f(i - 1) = \text{false}$ . Note that unlike in the previous section,  $i$  does not need to be even. We start at  $i = 1$ . We iterate calculating  $f(i)$ : if  $f(i) = \text{false}$ , we double  $i$  and continue, else we stop. Then we binary search normally between the last  $i$  and  $i/2$ . Due to Observation 3.5, we have  $f(i) = \text{true}$ , for  $i \geq \delta^*$ . This binary search algorithm results in a number  $i' \leq \lceil \delta^* \rceil$  and has a running time of  $O(F \times \log \delta^*)$ , where  $F$  the the time to run the test  $f(\cdot)$ .

**Test.** The calculation of  $f(i)$  proceeds very similarly to Section 3.3.2. We use a flow algorithm [73] to determine if a correct assignment  $g$  exists. We create a directed acyclic graph  $G = (V, E)$ , as illustrated in Figure 3.10. We set  $V = \{s, t\} \cup \mathcal{R} \cup \mathcal{S}_i$  as the set of vertices. We set  $E = \{(s, R) \mid R \in \mathcal{R}\} \cup \{(R, S) \mid R \in \mathcal{R}, S \in \mathcal{S}_i \wedge d_{\text{H}}^*(R, S) \leq i\} \cup \{(S, t) \mid S \in \mathcal{S}_i\}$  as the set of edges. We set the capacity of all those edges to 1.

We then calculate a maximal flow from  $s$  to  $t$ . We can assume that the flow in each edge is either 0 or 1. As  $|V|, |E| \in O(i^2 m)$ , the flow algorithm runs in  $O(i^4 m^2 / (\log i \log m))$  time [103]. If  $G$  admits a flow from  $s$  to  $t$  with flow rate  $m$ , the

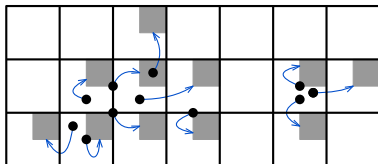


Figure 3.11: A set  $\mathcal{R}$  of 11 points and the valid mapping  $\mathcal{P}$  respecting a correct assignment  $g$  produced by the algorithm in Section 3.3.3.

flow induces a correct assignment  $g$  as follows: for each point  $R \in \mathcal{R}$ , there is exactly one superpixel  $S \in \mathcal{S}_i$  where the edge  $(R, S)$  has flow 1. We set  $g(R) = S$ . This results in a correct assignment  $g$ .

Thus, if  $G$  admits a flow from  $s$  to  $t$  with flow rate  $m$ , we set  $f(i) = \text{true}$ . Otherwise  $f(i) = \text{false}$ . This test  $f(i)$  performs as requested and has a running time of  $O(i^4 m^2 / (\log i \log m))$ .

**Placing the Pixels.** Let  $i'$  be the number that is the result from the binary search, that is,  $f(i') = \text{true}$  and  $f(i' - 1) = \text{false}$ . Let  $g$  be a correct assignment, calculated by the test  $f(i')$ . We place the pixels  $\mathcal{P}$  as shown in Figure 3.11: for each superpixel  $S \in \mathcal{S}_{i'}$ , if there is a point  $R$  assigned to  $S$ , we place the pixel corresponding to  $R$  in the top-right pixel of  $S$ . That way no two pixels in  $\mathcal{P}$  touch and for each point  $R$  its corresponding pixel is in the superpixel assigned to  $R$ . It follows that the Hausdorff distance between a point and its corresponding pixel is at most  $i' + \sqrt{2}$ . Since  $i' \leq \lceil \delta^* \rceil$  from the binary search, we have:

**Theorem 3.6.** *Given a set of  $m$  points  $\mathcal{R}$ , we can determine a valid mapping  $\mathcal{P}$  such that for each point  $R_i$  with corresponding pixel  $P_i$ , we have  $d_H(R_i, P_i) \leq \lceil \delta^* \rceil + \sqrt{2}$ , in  $O(m^2 \delta^{*4} / \log m)$  time.*

The running time is calculated as follows: the algorithm runs a binary search over  $i$ , where each step takes  $O(m^2 i^4 / (\log i \log m))$  time. In the end we get a value  $i' \leq \lceil \delta^* \rceil$ . Therefore, the running time is  $O(F \times \log \delta^*)$ , where  $F$  the the time to run one step.

### 3.4 Conclusion

We extended the previously known results on mapping regions to the grid with bounded Hausdorff distance. Where Bouts et al. [26] showed that, for a given set of regions, it is NP-hard to find the set of grid polygons that minimizes the Hausdorff distance  $d'_H$ , even if for just one region, we show that this is hard, even if all regions are points. On the other hand, where we focused on constructions that produce mappings



---

with Hausdorff distance asymptotically tight to the worst-case in Chapter 2, here we present the first approximation algorithms for mapping regions to the grid.

An interesting open question is whether the concepts presented in this chapter can be extended to an approximation algorithm that maps convex regions to the grid.



## Chapter 4

# Querying the Hausdorff Distance of a Line Segment

We consider the problem of preprocessing a set of points or segments  $R$  such that we can quickly determine the Hausdorff distance between a query segment and  $R$ . For  $|R| = n$  and parameters  $k \in [1 \dots n]$ ,  $\delta \in (0, 1)$  and  $\varepsilon > 0$ , we can store  $R$  in a data structure of size  $O(nk^{1+\varepsilon} + n^{1+\delta})$  in  $O(nk^{1+\varepsilon} + n^{1+\delta})$  expected time such that given a query line segment  $b$  we can compute the Hausdorff distance  $d_H(b, R) = \max \left\{ \vec{d}_H(b, R), \vec{d}_H(R, b) \right\}$  in  $O((n/k) \log k + \log^3 k + 2^{1/\delta} \log n)$  time. This chapter is based on the following presentation:

F. Staals, J. L. Vermeulen and J. Urhausen. Querying the Hausdorff Distance of a Line Segment. *Abstracts of the 38th European Workshop on Computational Geometry (EuroCG)*, pages 60:1–60:8, 2022.

### 4.1 Introduction

We are interested in computing the Hausdorff distance efficiently. Given a set  $R$  of “red” points, and a set  $B$  of “blue” points in  $\mathbb{R}^2$ , of sizes  $n$  and  $m$ , respectively, it is easy to compute the Hausdorff distance in  $O((n + m) \log(n + m))$  time. We simply build the Voronoi diagram of one set, and query it with the other. In case  $R$  and  $B$  are sets of disjoint line segments in  $\mathbb{R}^2$  the problem can be solved in the same time [7]. When  $R$  and  $B$  are convex polygons, their Hausdorff distance can even be computed in linear time [12].

The above algorithms are very good if  $B$  and  $R$  have similar sizes. However, when we wish to compute the Hausdorff distance between a possibly large set, say  $R$ , and multiple individual smaller sets  $B_1, \dots, B_k$  one by one, we wish to avoid the costly linear dependence on  $|R|$  when comparing  $R$  with each  $B_i$ . That is, we wish to

build a data structure on  $R$  that can be efficiently queried for the Hausdorff distance between  $R$  and some query object  $B$ . This setting appears naturally in a range of applications, for example when querying a shape database (for instance, to recognize a hand-written character by comparing it to block letters), trajectory clustering (in which a cluster is represented by a single representative curve) [102], or polyline simplification (test if some candidate shortcut segment is “good enough”) [28, 88]. Furthermore, efficiently computing the Hausdorff distance between subsets of varying sizes is one of the (many) challenging subproblems that arise if one is interested in maintaining the Hausdorff distance between two sets of line segments subject to updates.

**Problem Statement and Results.** Let  $R$  be a set of  $n$  points or line segments in  $\mathbb{R}^2$ , and let  $b$  be a single query line segment in  $\mathbb{R}^2$ . We develop a data structure storing  $R$  that can efficiently be queried for the Hausdorff distance between  $b$  and  $R$ . The hard part when computing  $d_H(b, R)$  is in determining  $\vec{d}_H(b, R) = \max_{p \in b} \min_{r \in R} d(p, r)$ . We first approach that part of the question assuming  $R$  is a set of points and later extend our solution to line segments. Observe that  $\vec{d}_H(b, R)$  is the maximum of two separate terms: (1) the distance from the endpoints of  $b$  to the corresponding closest point in  $R$  and (2) the maximum distance from an intersection of  $b$  and the Voronoi diagram of  $R$  to the corresponding closest point in  $R$  [7], see also Figure 4.1. While the first part is easy to compute, the second part needs a more involved data structure. We start in Section 4.2.1 by storing  $R$  using  $O(n^{2+\varepsilon})$  space and expected preprocessing time in a way that allows computation of said intersection-distance for a given query *line* in  $O(\log n)$  time. Here, and throughout the rest of the chapter,  $\varepsilon > 0$  is an arbitrarily small constant. Building on that idea, we extend the data structure in Section 4.2.2 to allow line segment queries at the cost of increasing the query time to  $O(\log^3 n)$ .

In Section 4.3, we then generalize this solution to when  $R$  is a set of line segments without increasing the space needed. The asymptotic running time does not change. Then, in Section 4.4, we show how, for any parameter  $k \in [1 \dots n]$ , we can decrease the space usage to  $O(nk^{1+\varepsilon})$  at the cost of increasing the query time to roughly  $O(n/k)$ .

Computing  $\vec{d}_H(R, b)$  turns out to be relatively straightforward in comparison, as the maximum distance from  $R$  to  $b$  is realized by an endpoint of a line segment in  $R$ . Using known results, for example farthest-point Voronoi diagrams, this yields an  $O(n^{1+\delta})$  space  $O(2^{1/\delta} \log n)$  time solution, for some parameter  $\delta \in (0, 1)$ , see Section 4.5. Overall, we can store a set  $R$  of line segments using  $O(nk^{1+\varepsilon} + n^{1+\delta})$  space and expected preprocessing time, such that given a line segment  $b$  we can determine the Hausdorff distance  $d_H(b, R)$  in  $O((n/k) \log k + \log^3 k + 2^{1/\delta} \log n)$  time.

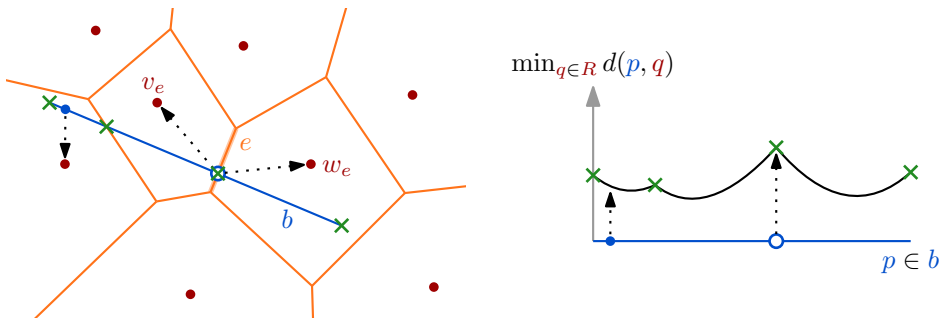


Figure 4.1: Left: the blue line segment  $b$  and the Voronoi diagram  $\text{Vor}_R$  of the red points  $R$ . An edge  $e \in \text{Vor}_R$  and the corresponding points  $v_e, w_e \in R$  are highlighted. Right: for each point  $p \in b$  we show the distance to the closest red point. The green crosses mark the points of interest.

### 4.1.1 Preliminaries

For a set of points or line segments  $R$  in  $\mathbb{R}^2$ , let  $\text{Vor}_R$  be their Voronoi Diagram. We consider  $\text{Vor}_R$  as a set of edges. When  $R$  is a point set, each edge is a line segment (or a ray). When  $R$  is a set of line segments on the other hand,  $\text{Vor}_R$  contains line segments and parabolic arcs. Each edge  $e \in \text{Vor}_R$  is equally close to two objects  $v_e$  and  $w_e$  of  $R$  that induce  $e$ . For a line segment  $b$ , the directed Hausdorff distance  $\vec{d}_H(b, R)$  is realized either by an endpoint of  $b$ , or a point on the intersection  $b \cap e$ , for an edge  $e \in \text{Vor}_R$  [7], see Figure 4.1. That is,

$$\vec{d}_H(b, R) = \max\{d_{\text{END}}(b, R), d_{\text{INT}}(b, R)\}$$

with the following definitions: for  $b = \overline{b_1 b_2}$ ,

$$d_{\text{END}}(b, R) = \max_{i \in \{1, 2\}} \vec{d}_H(b_i, R) = \max_{i \in \{1, 2\}} \min_{r \in R} d(b_i, r)$$

is the *endpoint-distance* and

$$d_{\text{INT}}(b, R) = \max_{e \in \text{Vor}_R} \{d(u, v_e) \mid u \in b \cap e\}$$

is the *intersection-distance*. We can easily compute the endpoint-distance using two  $O(\log n)$  time nearest neighbor queries on  $R$ . Hence, our main task is to develop a data structure that allows us to efficiently compute  $d_{\text{INT}}(b, R)$ .

## 4.2 A Data Structure for Red Points

We first explore how to determine  $d_{\text{INT}}(b, R)$ , when  $R$  is a set of points. We start with the case where  $b$  is a line, and then extend to the case where  $b$  is a line segment.

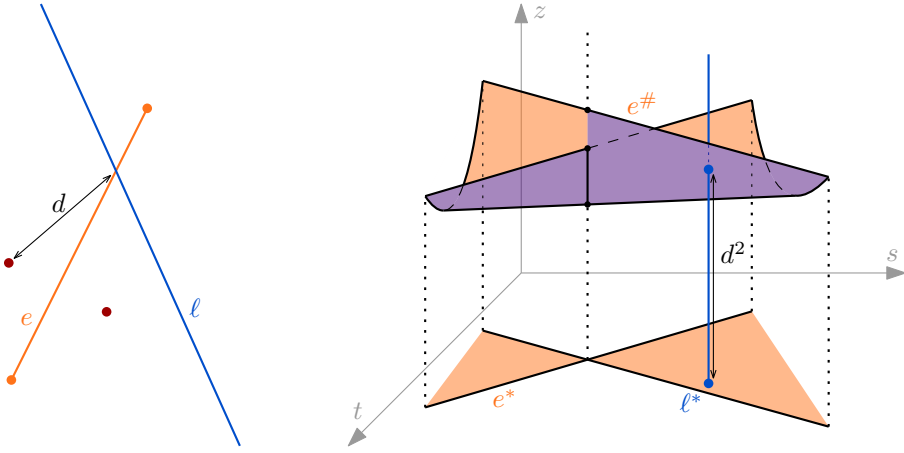


Figure 4.2: The Voronoi edge  $e$ , its corresponding dual  $e^*$  and its corresponding lifted surface  $e^\#$ . The surface  $e^\#$  is colored orange on the side that is visible from the top and purple on the side that is visible from the bottom. For a line  $\ell$  intersecting  $e$ , the squared distance between the intersection  $\ell \cap e$  and a corresponding red point  $v_e$  equals the height at which a vertical line through  $\ell^*$  punctures  $e^\#$ .

### 4.2.1 Querying with a Line

We want to store  $R$  so that we can efficiently compute  $d_{\text{INT}}(b, R)$  for a query line  $b$ . For each edge  $e$  in the Voronoi Diagram  $\text{Vor}_R$ , we lift the double wedge  $e^*$  to the surface  $e^\#$  such that a point  $\ell^* \in e^*$  is lifted to a height equal to the squared distance between the red Voronoi site  $v_e$  and the intersection of  $\ell$  and  $e$ , as shown in Figure 4.2. We square to simplify the derivations. This surface is also shown in Figure 1.3 in Chapter 1 (Introduction). Formally,  $e^\# = \{(s, t, d^2(v_e, \ell \cap e)) \mid \ell^* := (s, t) \in e^*\}$ . For a set  $E$ , we set  $E^\# = \{e^\# \mid e \in E\}$ . As a result, for a double wedge  $e^*$  and a line  $p^*$  through the center of the double wedge, all points  $\ell^*$  on  $p^*$  will be lifted to the same height, meaning  $e^\#$  is a ruled surface, see Figure 4.3. As we want to determine the directed Hausdorff distance, for each line  $\ell$ , we are interested in  $\max\{z \mid \exists e \in \text{Vor}_R : (s, t) = \ell^* \wedge (s, t, z) \in e^\#\}$ , that is, we care about the upper envelope of these surfaces.

Sharir [112] proves that the complexity of this upper envelope is  $O(n^{2+\epsilon})$  and that it can be constructed using a randomized algorithm in  $O(n^{2+\epsilon})$  expected time. We use their construction and project the upper envelope down to  $\mathbb{R}^2$  along the  $z$ -axis. We label each area in the arrangement induced by the projection with the surface  $e^\#$  on the upper envelope above that area. As a line parallel to the  $z$ -axis intersects each surface  $e^\# \in E^\#$  at most once, this labeled projection also has a complexity of  $O(n^{2+\epsilon})$ . Finally, we use Chapter 6 of [18] to allow point location queries on this projection in  $O(\log n)$  time. Thus, for a query line  $b$ , we query the point location data

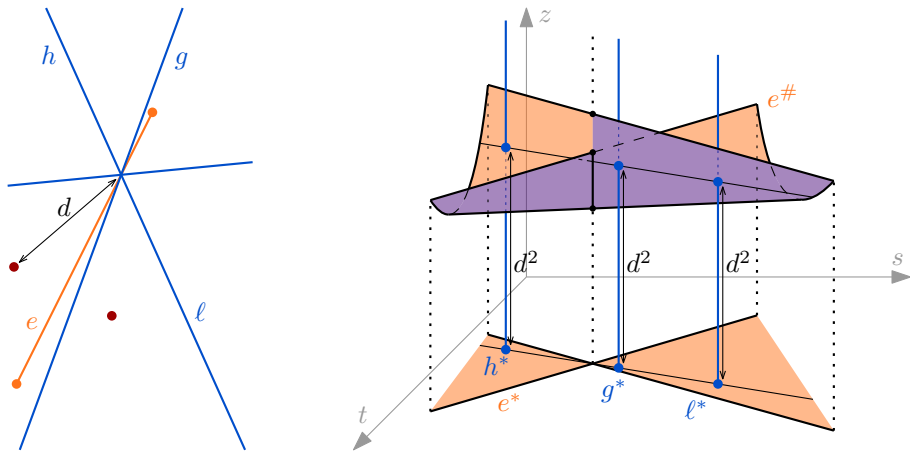


Figure 4.3: The Voronoi edge  $e$ , its corresponding dual  $e^*$  and its corresponding lifted surface  $e^\#$ . The points dual to lines intersecting a Voronoi edge  $e$  in the same point are aligned and are lifted to the same height in  $e^\#$ .

structure using  $b^*$ , resulting in the surface  $e^\#$  that is on the upper envelope of  $E^\#$  above  $b^*$ . Thus we can calculate the height of  $e^\#$  above  $b^*$ . As previously stated, this determines  $d_{\text{INT}}(b, R)$ .

Note that when  $b$  is a vertical line,  $b^*$  is not defined. That is why we additionally compute the same data structure over the set  $R$  rotated around the origin by  $\pi/2$  radians. We get:

**Lemma 4.1.** *Let  $R$  be a set of  $n$  points in  $\mathbb{R}^2$ . In  $O(n^{2+\epsilon})$  expected time we can build an  $O(n^{2+\epsilon})$  size data structure that can compute  $d_{\text{INT}}(b, R)$  for a query line  $b$  in  $O(\log n)$  time.*

Lemma 4.1 is illustrated in Figure 4.4. For a segment  $s$ , let  $\ell_s$  be the supporting line of  $s$ . We also state the following corollary:

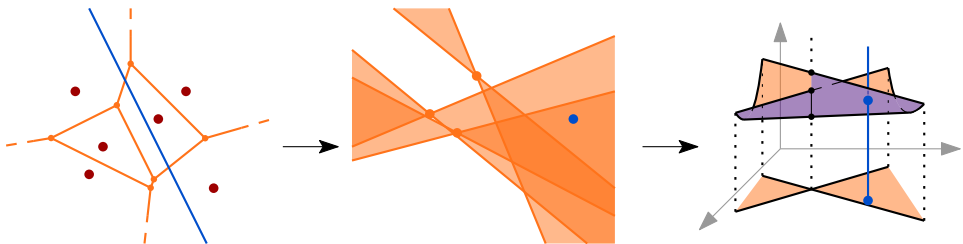


Figure 4.4: The overview of Section 4.2.1. To query the intersection distance for a line  $\ell$ , we query the upper envelope above the point  $\ell^*$ .

**Corollary 4.2.** *Let  $E \subseteq \text{Vor}_R$  be a set of  $k$  Voronoi edges. In  $O(k^{2+\epsilon})$  expected time we can build an  $O(k^{2+\epsilon})$  size data structure that can compute  $\max\{z \mid \exists e \in E : (s, t) = \ell_b^* \wedge (s, t, z) \in e^\#\}$  for a query line segment  $b$  intersecting all edges  $E$  in  $O(\log k)$  time.*

## 4.2.2 Querying with a Line Segment

We now extend the data structure to support queries with a line segment, see Figure 4.5. For a set of  $n$  hyperplanes  $H$  in  $\mathbb{R}^d$  and  $r \in [1 \dots n]$ , a  $1/r$ -cutting of  $H$  is a partition of  $\mathbb{R}^d$  into cells with disjoint interiors, each of which is intersected by at most  $n/r$  hyperplanes from  $H$  [42]. Furthermore, each cell is a connected region bounded by a simple curve, or in higher dimension, a not self-intersecting surface. The *conflict list* of a cell  $\nabla$  is the set of the hyperplanes intersecting the interior of  $\nabla$ .

**Construction.** We can cut  $\mathbb{R}^2$  into  $O(r^2)$  cells where each cell is intersected by at most  $n/r$  boundaries of double wedges in  $\text{Vor}_R^*$  in  $O(nr)$  time [42]. Figure 4.6 shows an example of such a cutting. This algorithm from [42] also computes the conflict list for each cell. Let  $E_\nabla^{\text{int}} = \{e \in \text{Vor}_R \mid \emptyset \subsetneq e^* \cap \nabla \subsetneq \nabla\}$  be the edges whose dual double wedges have boundaries in the conflict list of  $\nabla$ .

For each cell  $\nabla$ , we compute the following: let  $E_\nabla^{\text{elem}}$  be the set of edges whose dual double wedges contain  $\nabla$ , that is,  $E_\nabla^{\text{elem}} = \{e \in \text{Vor}_R \mid \nabla \subseteq e^*\}$ . We sort the edges  $E_\nabla^{\text{elem}}$  by the order in which a line  $\ell$  with  $\ell^* \in \nabla$  intersects them. Lemma 4.3 ensures that a unique sorting exists. We then build a balanced binary search tree  $T_\nabla$  on  $E_\nabla^{\text{elem}}$ , where each node has a set of edges associated with it. The set of a leaf contains one edge and the set of an inner node contains the edges contained in its children's sets, as illustrated in Figure 4.7. For each node with set of edges  $E$  we calculate the upper envelope of  $E^\#$ , as stated in Corollary 4.2. The tree with the upper envelope at each node uses  $T(n)$  space and construction time, where

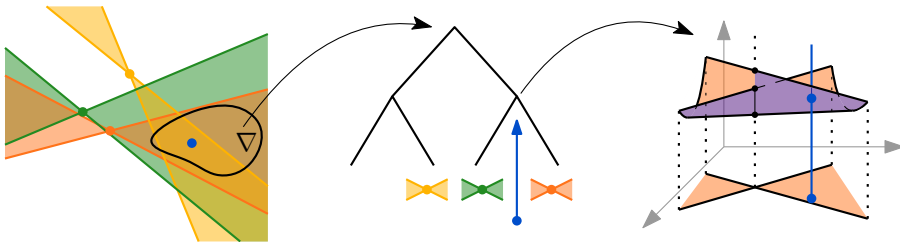


Figure 4.5: The overview of the data structure of Section 4.2.2. Each cell of the cutting stores a balanced binary search tree whose internal nodes store an upper envelope.



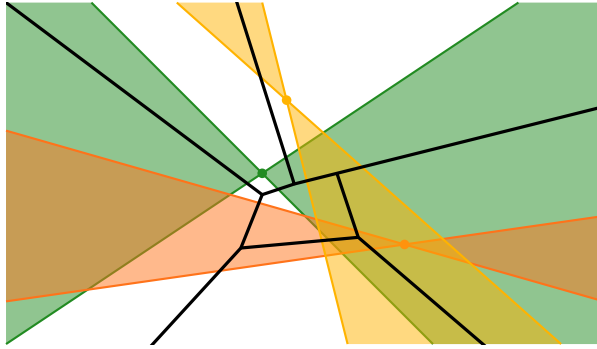


Figure 4.6: A cutting of three wedges into six cells where each cell is intersected by at most three boundaries.

$T(n) = 2T(n/2) + O(n^{2+\epsilon})$ . This means

$$T(n) \in O\left(\sum_{i=0}^{\log n} 2^i \left(\frac{n}{2^i}\right)^{2+\epsilon}\right) = O\left(n^{2+\epsilon} \sum_{i=0}^{\log n} \left(\frac{1}{2^i}\right)^{1+\epsilon}\right) = O(n^{2+\epsilon}).$$

**Lemma 4.3.** *Let  $E$  be the edges of a planar subdivision. Let  $\nabla \subseteq \mathbb{R}^2$  be a connected region bounded by a simple curve and let  $E_{\nabla}^{\text{elem}}$  be the edges in  $E$  dual to the double wedges that contain  $\nabla$ . Any two lines with duals within  $\nabla$  will intersect all edges of  $E_{\nabla}^{\text{elem}}$  in the same order.*

*Proof.* This proof is similar to the proof of Lemma 4.2 of [2]. However, as that proof is abbreviated, we will prove the statement here in its entirety. Let  $a$  be a line with dual point inside  $\nabla$ . For each edge  $e \in E_{\nabla}^{\text{elem}}$ , we have  $a^* \in e^*$ , thus  $a$  intersects all edges  $E_{\nabla}^{\text{elem}}$ . It remains to show that any two lines with duals inside  $\nabla$  intersect those edges in the same order.

Let  $a$  and  $c$  be two lines with dual points within the interior of  $\nabla$  and assume for the sake of contradiction that  $a$  and  $c$  intersect the edges  $E_{\nabla}^{\text{elem}}$  in a different order. Let  $u : [0, 1] \rightarrow \mathbb{R}^2$  be a curve with  $u(0) = a^*$  and  $u(1) = c^*$  whose interior does not touch the boundary of  $\nabla$ . Such a curve exists as  $\nabla$  is a region bounded by a simple curve. Each point  $u(i)$  on  $u$  corresponds to a line  $u(i)^*$  in the primal that intersects all edges  $E_{\nabla}^{\text{elem}}$ . Increasing  $i$  from 0 to 1 corresponds to moving  $u(i)$  along  $u$  from  $a^*$  to  $c^*$  and it corresponds to continuously transforming the line  $u(i)^*$  from  $a$  to  $c$ . Let  $j = \inf_{i \in [0, 1]} \{u(i)^* \text{ intersects } E_{\nabla}^{\text{elem}} \text{ in a different order than } a\}$ . It follows that  $u(j)^*$  intersects two edges of  $E_{\nabla}^{\text{elem}}$  in the same point. As  $E$  is a set of interior-disjoint edges,  $u(j)^*$  intersects the endpoint of at least one edge  $e \in E_{\nabla}^{\text{elem}}$ . That means  $u(j)$  is on the boundary of the double wedge  $e^*$ . As  $\nabla \subseteq e^*$ ,  $u(j)$  is also on the boundary of  $\nabla$ , a contradiction.  $\square$

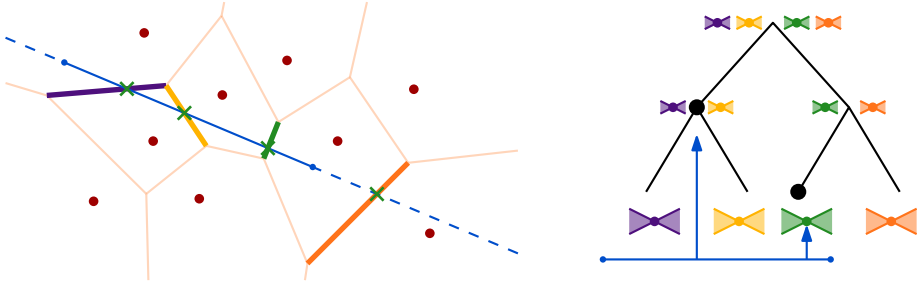


Figure 4.7: Left: a query segment whose supporting line intersects four Voronoi edges. Right: a binary search tree build on those four edges. A query determines the two nodes containing the three edges intersected by the query segment.

We process each cell of the cutting recursively, as described by [42]. That is, for each cell, if it is intersected by  $k > 0$  boundaries of double wedges, we again cut it into  $O(r^2)$  cells where each cell is intersected by at most  $k/r$  boundaries of double wedges. We again determine  $E_{\nabla}^{\text{int}}$  for each cell  $\nabla$  using the conflict list. Also, for each cell  $\nabla$  with parent cell  $\nabla'$ , we define  $E_{\nabla}^{\text{elem}} = \{e \in \text{Vor}_R \mid \nabla \subseteq e^* \wedge \nabla' \not\subseteq e^*\}$ , that is,  $E_{\nabla}^{\text{elem}} = \{e \in E_{\nabla}^{\text{int}} \mid \nabla \subseteq e^*\}$ . We compute the abovementioned balanced binary search tree  $T_{\nabla}$  and upper envelope data structure on the edges  $E_{\nabla}^{\text{elem}}$ . In the end, the space needed is  $S(n) = cr^2(n^{2+\varepsilon} + S(n/r))$ , for some constant  $c$ . For a sufficiently large constant  $r$ , we get  $S(n) \in O(n^{2+\varepsilon})$ . The same holds for the construction time.

**Query.** When querying  $\ell$  with a line segment  $b$  with supporting line  $\ell_b$ , we proceed as follows. We determine the cell  $\nabla$  containing  $\ell_b^*$  and recurse within  $\nabla$ . This results in the set  $C$  of nested cells all containing  $\ell_b^*$ . Note that for each edge  $e$  intersecting  $\ell_b$ , that is,  $\ell_b^* \in e^*$ , we have one cell  $\nabla \in C$ , where  $e \in E_{\nabla}^{\text{elem}}$ . Within each tree  $T_{\nabla}$  with  $\nabla \in C$ , we compute the consecutive range of edges intersecting the line segment  $b$  and not just the supporting line  $\ell_b$ . This gives us a set of  $O(\log^2 n)$  nodes that together represent all edges from  $\text{Vor}_R$  intersected by  $b$ . For each of those nodes we query the height of the upper envelope at  $(s, t) = \ell_b^*$ . The result of the query is the maximum over those heights. This query takes  $O(\log^3 n)$  time and returns the intersection distance  $d_{\text{INT}}(b, R)$ . We finally perform two point location queries in the Voronoi Diagram—one for each of the endpoints of  $b$ —to determine  $d_{\text{END}}(b, R)$ .

**Theorem 4.4.** *Let  $R$  be a set of  $n$  points in  $\mathbb{R}^2$ . In  $O(n^{2+\varepsilon})$  expected time we can build an  $O(n^{2+\varepsilon})$  size data structure that can compute  $\vec{d}_H(b, R)$  for a query line segment  $b$  in  $O(\log^3 n)$  time.*

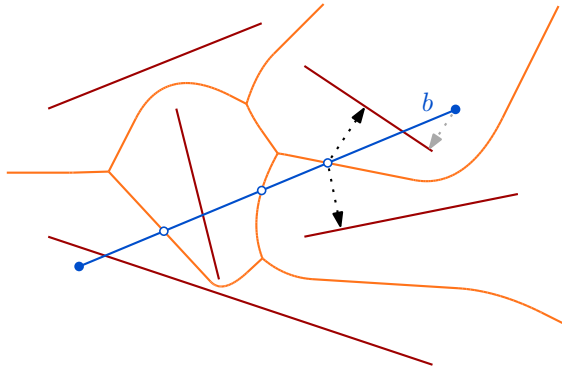


Figure 4.8: The directed Hausdorff distance  $\vec{d}_H(b, R)$  is realized either at an endpoint of  $b$ , or at an intersection of  $b$  with an edge of the Voronoi diagram of  $R$ , even when  $R$  contains line segments and not just points.

### 4.3 A Data Structure for Red Line Segments

Now we consider the problem when  $R$  is a set of line segments instead of a set of points and  $b$  is still a line segment. The main difference is that the edges of the Voronoi Diagram  $\text{Vor}_R$  may be parabolic arcs, as shown in Figure 4.8. Nonetheless, the results from Section 4.2 directly translate here. Note that we consider each parabolic arc and each line segment a separate edge. The main question in this section concerns the shape of the dual  $e^*$  of a parabolic arc  $e$  and the shape of the lifted surface  $e^\#$ . To be precise, we define the *pseudo-double wedge*  $e^*$  as the set of all points  $\ell^*$ , where  $\ell$  intersects the edge  $e$ , and  $e^\# = \{(s, t, \max_{a \in \ell \cap e} d^2(a, v_e)) \mid \ell^* = (s, t) \in e^*\}$ . We use the maximum, as  $\ell$  might intersect  $e$  twice. See Figure 4.9 for an example. The following two lemmata describe the shape of  $e^*$ . A conic is a curve described by an equation of the form  $ax^2 + bxy + cy^2 + dx + ey + f = 0$ , for some  $a, b, c, d, e, f \in \mathbb{R}$ . A conic is *nondegenerate* if it is a circle, ellipse, parabola, or hyperbola, that is, if it contains at least two points and no three collinear points.

**Lemma 4.5.** *The points dual to the tangents of a parabola form a nondegenerate conic.*

*Proof.* We make use of projective geometry [104]. This proof goes through the following steps:

1. We define points and (parameters of) lines in homogeneous coordinates. They are three-dimensional vectors.
2. We define projective transformations on those points and lines. They correspond to multiplying all vectors with a matrix.
3. For any conic  $\mathcal{C}$ , we define a matrix  $A_{\mathcal{C}}$  corresponding to that conic.

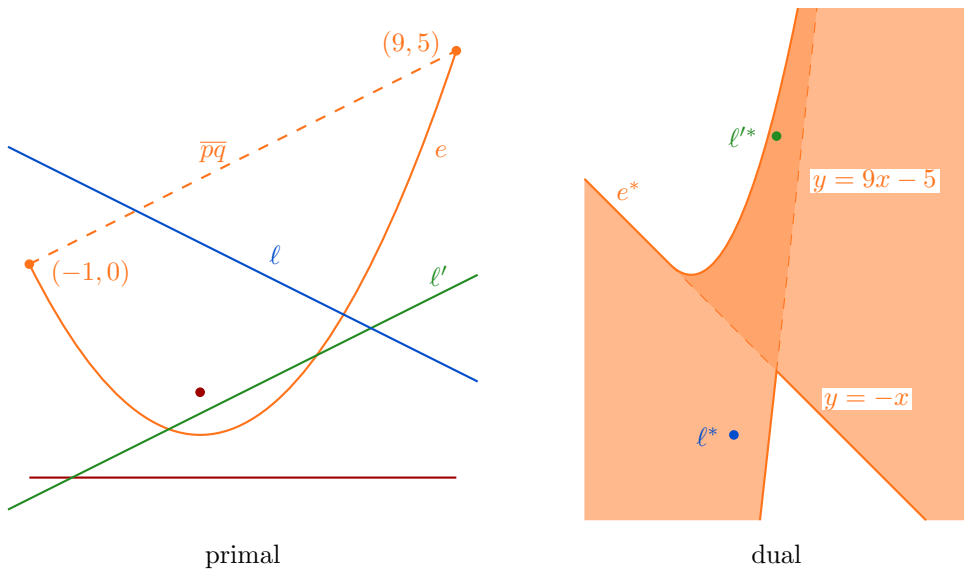


Figure 4.9: A Voronoi edge  $e$  and the corresponding dual pseudo-double wedge  $e^*$ . A line  $\ell$  intersecting  $e$  once also intersects the line segment  $\overline{pq}$  between the endpoints of the parabolic arc and thus  $\ell^* \in \overline{pq}^*$ . A line  $\ell'$  intersecting  $e$  twice has its dual point  $\ell'^*$  between the double wedge and a conic.

4. We prove that a conic is nondegenerate if and only if its matrix is invertible.
5. We observe that generating the parameters of the line tangent to a conic  $\mathcal{C}$  in a point corresponds to a matrix multiplication using  $A_{\mathcal{C}}$ .
6. We observe that the dual transformation is a projective transformation involving an invertible matrix  $D$  where, after the transformation, points are now regarded as if they were lines and vice versa.
7. We finish by stating that, for a parabola  $\mathcal{C}$ , the matrix  $A_{\mathcal{C}'} = D^{-1T} \cdot A_{\mathcal{C}}^{-1T} \cdot D^{-1}$  corresponds to the conic  $\mathcal{C}'$  describing the points dual to the tangents of  $\mathcal{C}$ . As  $A_{\mathcal{C}}$  is invertible  $\mathcal{C}'$  is a nondegenerate conic.

To start, we present homogeneous coordinates. Each point  $p = (x, y) \in \mathbb{R}^2$  corresponds to the equivalence class

$$[p] = \left\{ \begin{pmatrix} \lambda x \\ \lambda y \\ \lambda \end{pmatrix} \mid \lambda \in \mathbb{R} \setminus \{0\} \right\} \text{ with representative } \hat{p} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}.$$

We use the notation  $\hat{p}$  to describe a point in homogeneous coordinates;  $\hat{p}$  is a column-vector. A line  $\ell \equiv ax + by + c = 0$  corresponds to the line  $\hat{\ell} \equiv ax + by + cz = 0$  in projective geometry. The parameters of  $\hat{\ell}$  are described by the equivalence class

$$[\hat{\ell}_{\text{para}}] = \left\{ \begin{pmatrix} \lambda a \\ \lambda b \\ \lambda c \end{pmatrix} \mid \lambda \in \mathbb{R} \setminus \{0\} \right\} \text{ with representative } \hat{\ell}_{\text{para}} = \begin{pmatrix} a \\ b \\ c \end{pmatrix}.$$

To test if a point  $p$  is on a line  $\ell$ , we verify if  $\hat{p}^T \hat{\ell}_{\text{para}} = 0$ . Thus,  $\hat{\ell} = \left\{ \hat{p} \in \frac{\mathbb{R}^3 \setminus \{(0,0,0)\}}{\mathbb{R} \setminus \{0\}} \mid \hat{p}^T \hat{\ell}_{\text{para}} = 0 \right\}$ . Observe that in projective geometry points and lines are similar.

We can apply a projective transformation to the equivalence classes. A projective transformation is described by an invertible  $3 \times 3$  matrix  $M$ . Applying a projective transformation corresponds to multiplying each point with  $M$  and the parameters of each line with  $M^{-1T}$ . Recall that a conic is described by an equation of the form  $ax^2 + bxy + cy^2 + dx + ey + f = 0$ , for some  $a, b, c, d, e, f \in \mathbb{R}$ , which is equivalent to  $\hat{p}^T \cdot A_C \cdot \hat{p} = 0$ , where  $A_C = \begin{pmatrix} a & b/2 & d/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & f \end{pmatrix}$  is the matrix corresponding to conic  $\mathcal{C}$ . A projective transformation  $M$  transforms  $\mathcal{C}$  to the conic with matrix  $M^{-1T} \cdot A_C \cdot M^{-1}$ , implying that whether the matrix of a conic is invertible is invariant under projective transformations.

Now, in step 4, we prove that a conic is nondegenerate if and only if its corresponding matrix is invertible, that is, if the matrix has nonzero determinant. Richter-Gebert [104] states in Theorem 9.3 that "Every planar conic is projectively equivalent to a conic of the form  $\sigma_1 x^2 + \sigma_2 y^2 + \sigma_3 z^2 = 0$  with  $\sigma_i \in \{+1, -1, 0\}$  for  $i \in \{1, 2, 3\}$ ." This means that each conic can be transformed into a conic of the above form using a projective transformation. We now analyze the different types of shape which the conic  $\mathcal{C}$  described by the homogeneous equation  $\sigma_1 x^2 + \sigma_2 y^2 + \sigma_3 z^2 = 0$  can have:

- $\sigma_1 = \sigma_2 = \sigma_3 = 0$ : the conic is the entire space  $\frac{\mathbb{R}^3 \setminus \{(0,0,0)\}}{\mathbb{R} \setminus \{0\}}$ .
- exactly one sigma is nonzero: the conic is a line.; for example for  $\sigma_2 \neq 0$ , the conic equals the line  $y = 0$  in the Euclidean plane.
- one sigma is zero and the other two have the same sign: the conic is a point, as two coordinates are set to 0.
- one sigma is zero and the other two have different signs: the conic consists of two lines; for example for  $\sigma_3 = 0$ , the conic equals the lines  $y = x$  and  $y = -x$  in  $\mathbb{R}^2$ .
- $\sigma_1 = \sigma_2 = \sigma_3 \neq 0$ : no point satisfies this equation, as  $(0, 0, 0) \notin \frac{\mathbb{R}^3 \setminus \{(0,0,0)\}}{\mathbb{R} \setminus \{0\}}$ .
- otherwise, no sigma is zero and not all sigma have the same sign:  $\mathcal{C}$  is the unit hyperbola, its conjugate, or the unit circle in  $\mathbb{R}^2$ , meaning  $\mathcal{C}$  is nondegenerate.

The determinant of the matrix  $A_C$  is zero if and only if there is a  $\sigma_i$  that is zero, meaning  $A_C$  is invertible only in the last case. Richter-Gebert [104] states in Theorem 3.2 “A projective transformation maps collinear points to collinear points.” Note that only the conics in the last case have multiple points and no three collinear points. Therefore, any nondegenerate conic is projectively equivalent to a conic that is in the last case and is thus described by an invertible matrix. Conversely, if the matrix of a conic is invertible, it is again projectively equivalent to conic that is in the last case and thus it is nondegenerate.

Moving on to step 5 of the proof, Richter-Gebert [104] says in Theorem 9.1 that for  $p \in \mathcal{C}$ ,  $\hat{\ell}_{\text{para}} = A_C \hat{p}$  are the parameters of the tangent at  $\hat{p}$  to  $\mathcal{C}$  in the projective plane. For step 6, observe that for a line  $\hat{\ell} = \begin{pmatrix} -s \\ 1 \\ -t \end{pmatrix} \equiv y = sx + t$ , its dual point is  $\hat{\ell}^* = \begin{pmatrix} s \\ -t \\ 1 \end{pmatrix} = D \cdot \hat{\ell}_{\text{para}}$ , where  $D = \begin{pmatrix} -1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$ .

Finally, let  $\mathcal{C}$  be a parabola. We have  $\mathcal{C} = \{p \mid \hat{p}^T \cdot A_C \cdot \hat{p} = 0\}$  in projective geometry (step 3). The matrix  $A_C$  is invertible (step 4). The parameters of the tangents to  $\mathcal{C}$  are described by  $\hat{\ell}_{\text{para}}^T \cdot A_C^{-1T} \cdot \hat{\ell}_{\text{para}} = 0$  (step 5). Thus, the points  $\hat{\ell}^*$  dual to the tangents to  $\mathcal{C}$  satisfy  $\hat{\ell}^{*T} \cdot D^{-1T} \cdot A_C^{-1T} \cdot D^{-1} \cdot \hat{\ell}^* = 0$  (step 6). The matrix  $D^{-1T} \cdot A_C^{-1T} \cdot D^{-1}$  is invertible meaning the points form a nondegenerate conic.  $\square$

**Lemma 4.6.** *For an edge  $e$ , the pseudo-double wedge  $e^*$  is an area bounded by three algebraic curves each of degree at most two.*

*Proof.* We show that the curves bounding  $e^*$  are roots of bivariate polynomials of degree at most two. Let  $e_s$  be the supporting parabola of  $e$  and let  $p$  and  $q$  be the endpoints of  $e$ . Let  $\ell \equiv y = sx + t$  be a line intersecting  $e$ .

Let  $\ell_{\min}$  ( $\ell_{\max}$ ) be the bottommost (topmost) line parallel to  $\ell$  intersecting  $e$ . Every line parallel to  $\ell$  between  $\ell_{\min}$  and  $\ell_{\max}$  intersects  $e$ . Then,  $\ell_{\min}$  ( $\ell_{\max}$ ) contains  $p$  or  $q$ , or is tangent to  $e$ . Similarly, let  $s_{\inf} = \inf_{s' \in \mathbb{R}} \{s' \mid e \text{ intersects } y = s'x + t\}$ . We symmetrically define  $s_{\sup}$ . For  $s_{\inf} < s' < s_{\sup}$ , the line  $y = s'x + t$  intersects  $e$ . Again, if  $s_{\inf} \neq -\infty$  ( $s_{\sup} \neq \infty$ ), the line  $y = s_{\inf}x + t$  ( $y = s_{\sup}x + t$ ) contains  $p$  or  $q$ , or is tangent to  $e$ .

Thus, when  $\ell^*$  is a point on the boundary of  $e^*$ ,  $\ell$  intersects  $p$  or  $q$ , or is tangent to  $e$ . As the points dual to the lines tangent to  $e_s$  form a nondegenerate conic  $\mathcal{C}$  (Lemma 4.5),  $e^*$  is an area bounded by the conic  $\mathcal{C}$  and the lines  $p^*$  and  $q^*$ .  $\square$

From Lemma 4.6 we deduce that the arrangement of pseudo-double wedges  $E_R^*$  has complexity  $O(n^2)$ . Thus, we can extend the line query data structure from Section 4.2.1. Furthermore, the (vertical decomposition of the) arrangement of a random sample of  $\Omega(r^2 \log^2 r)$  such pseudo-double wedges is expected to be a  $1/r$ -cutting of  $E_R^*$  [72]. It then follows we can compute a  $1/r$ -cutting of  $E_R^*$  of size  $O(r^2)$  in  $O(nr)$  expected time [22]. This enables to extend the line segment query data structure in Section 4.2.2.

Next, we are interested in the shape of the surface  $e^\#$ , so that we can prove an upper bound on its complexity. A surface in  $\mathbb{R}^3$  is a *terrain* if any line parallel to the  $z$ -axis intersects the surface at most once. The definition of when a surface is algebraic of constant degree is beyond the scope of this thesis; we use here that, for a surface to be *algebraic of constant degree*, it is sufficient if the surface is the zero set of a polynomial whose degree is bounded by a constant. We prove that each  $e^\#$  is an algebraic terrain of constant degree, which means their upper envelope can be stored for  $O(\log n)$  time point location queries using  $O(n^{2+\epsilon})$  space [112].

**Lemma 4.7.** *The surface  $e^\#$  is an algebraic terrain of degree at most eight or the maximum over two algebraic terrains each of degree at most eight.*

*Proof.* Recall that  $e^\#$  is defined as  $\{(s, t, \max_{a \in \ell \cap e} d^2(a, v_e)) \mid \ell^* = (s, t) \in e^*\}$ .

We start by expressing the intersection  $\ell \cap e$  depending on the parameters  $s$  and  $t$  of the line  $\ell$ . Let  $\ell \equiv y = sx - t$  and let  $f_e(\cdot)$  be the polynomial of degree at most two that describes the supporting line or parabola of the edge  $e$ . Let  $a \in \ell \cap e$ , implying  $a_y = sa_x - t \wedge f_e(a) = 0 \implies f_e((a_x, sa_x - t)) = 0$ . We solve that last equation for  $a_x$  resulting in at most two solutions  $x_1(s, t)$  and  $x_2(s, t)$  that are functions of  $s$  and  $t$ .

If  $e$  is a line segment, there is only one solution  $x_1(s, t)$ . Then  $e^\# = \{(s, t, z) \mid z = d^2((x_1(s, t), sx_1(s, t) - t), v_e)\}$ . Then, the equation  $z = d^2((x_1(s, t), sx_1(s, t) - t))$  can be rewritten as  $z = \frac{f_1(s, t)}{f_2(s)}$ , where  $f_1$  and  $f_2$  are polynomials of degree two. This is equivalent with  $f_1(s, t) - zf_2(s) = 0$ , proving  $e^\#$  is an algebraic terrain of degree at most three in this case.

Otherwise,  $e$  is a parabolic arc, so we get two solutions  $x_1(s, t)$  and  $x_2(s, t)$ . So,  $e^\#$  is the maximum over the two surfaces  $\{(s, t, z) \mid z = d^2((x_i(s, t), sx_i(s, t) - t), v_e)\}$  for  $i \in \{1, 2\}$ . For each  $i \in \{1, 2\}$ , the equation  $z = d^2((x_i(s, t), sx_i(s, t) - t), v_e)$  can be rewritten as  $z = \frac{f_1(s, t) + f_2(s, t) \sqrt{f_3(s, t)}}{f_4(s)}$ , where  $f_1, f_2, f_3$  and  $f_4$  are polynomials of degree at most four. This in turn can be rewritten as  $f(s, t, z) = 0$ , where  $f$  is a polynomial of degree at most eight. So,  $e^\#$  is the maximum of two algebraic terrains of constant degree.  $\square$

Thus, we can compute arrangements of pseudo-double wedges, compute cuttings of these arrangements and finally we can lift pseudo-double wedges to 3D and compute their upper envelope. Hence, we can extend the approach from Section 4.2 to line segments. Below we restate Lemma 4.1 and Theorem 4.4. The asymptotic running time remains unchanged.

**Lemma 4.8.** *Let  $R$  be a set of  $n$  disjoint line segments in  $\mathbb{R}^2$ . In  $O(n^{2+\epsilon})$  expected time we can build an  $O(n^{2+\epsilon})$  size data structure that can compute  $d_{\text{Int}}(b, R)$  for a query line  $b$  in  $O(\log n)$  time.*

**Theorem 4.9.** *Let  $R$  be a set of  $n$  disjoint line segments in  $\mathbb{R}^2$ . In  $O(n^{2+\epsilon})$  expected time we can store  $R$  in a data structure of size  $O(n^{2+\epsilon})$  such that given a query line segment  $b$  we can compute  $\overrightarrow{d_H}(b, R)$  in  $O(\log^3 n)$  time.*

## 4.4 A Space-Time Tradeoff

We describe how to adapt our data structure to reduce the space used, at the cost of increasing the query time. Let again  $R$  be a set of  $n$  red line segments. Using a parameter  $k \in [1 \dots n]$ , we reduce the space used to  $O(nk^{1+\varepsilon} + n \text{ polylog } n)$ , while increasing the query time to  $O((n/k) \text{ polylog } k)$ . For  $k = n$  the results in this section match the results stated in Theorem 4.9. The main idea is to partition  $\mathbb{R}^2$  into  $O(n/k)$  cells, such that in each cell  $\nabla$ , there are only  $O(k)$  line segments from  $R$  that contribute to  $\text{Vor}_R$ . We then build our data structure from Theorem 4.9 on each such set. A query line segment  $b$  may now intersect all  $O(n/k)$  cells, which ultimately yields a query time of  $O((n/k) \log k + \log^3 k)$ .

**Points.** Again, we first describe our data structure for a set of points  $R$  and extend it later to line segments. We lift a point  $r \in R$  to a plane in  $\mathbb{R}^3$  using the standard lifting transformation (Chapter 8 in [18]). Formally this lifts a point  $r = (r_x, r_y)$  to  $r^\wedge = \{(x, y, z) \mid z = 2xr_x + 2yr_y - (r_x^2 + r_y^2)\}$ . Note that the projection of the lower envelope of these surfaces corresponds to the Voronoi diagram of  $R$ . Let  $R^\wedge$  denote the resulting set of planes, and let  $\Lambda$  be a vertical  $k$ -shallow cutting on  $R^\wedge$  [41]. Such a cutting  $\Lambda$  consists of  $O(n/k)$  vertical constant complexity prisms that together cover the  $\leq k$ -levels of the arrangement of  $R^\wedge$ . Moreover, each prism is intersected by only  $O(k)$  planes. Computing  $\Lambda$  takes  $O(n \log(n/k))$  time [41].

For each cell (prism)  $\nabla \in \Lambda$ , we consider the points  $R_\nabla$  corresponding to the  $O(k)$  planes in the conflict list of  $\nabla$ . We build the Voronoi diagram  $\text{Vor}_\nabla$  of  $R_\nabla$  and clip it to the downward projection  $\underline{\nabla}$  of  $\nabla$ . Observe that due to the relation between the Voronoi diagram of  $R$  and the lower envelope of  $R^\wedge$ , for any point  $q \in \underline{\nabla}$  the site among  $R_\nabla$  closest to  $q$  is actually the closest site overall. So if we were to glue together all (clipped) Voronoi diagrams  $\text{Vor}_\nabla$  over all cells we obtain the Voronoi diagram of  $R$ .

**Observation 4.10.** *Let  $p \in \underline{\nabla}$ . Then  $\vec{d}_H(p, R) = \vec{d}_H(p, R_\nabla)$ . Also, for a segment  $b$  intersecting  $\nabla$ , we have  $d_{\text{INT}}(b \cap \underline{\nabla}, R) = d_{\text{INT}}(b \cap \underline{\nabla}, R_\nabla)$ .*

For each cell  $\nabla \in \Lambda$  we now store the edges of its clipped Voronoi diagram once in the line data structure of Lemma 4.1 and once in the data structure of Theorem 4.4. Since each such a data structure requires  $O(k^{2+\varepsilon})$  space and there are  $O(n/k)$  cells, the total size of our data structure is  $O(nk^{1+\varepsilon})$ .

To answer a query with a line segment  $b = \overline{b_1 b_2}$  we naively compute the subset of cells from  $\Lambda$  that intersect  $b$ . For the at most two cells  $\nabla_1, \nabla_2$  of  $\Lambda$  whose downward projections contain the endpoints  $b_1, b_2$  of  $b$ , we query the data structure belonging to  $\nabla_i$  described in Theorem 4.4 to compute the directed Hausdorff distance from the part of  $b$  that lies inside the cell  $\nabla_i$  to  $R_{\nabla_i}$ . Formally, using Observation 4.10, this part of the query returns

$$\max\{d_{\text{END}}(b, R), \max_{i \in \{1, 2\}} \{d_{\text{INT}}(b \cap \underline{\nabla}_i, R)\}\}.$$



For the other intersected cells we use the data structure from Lemma 4.1 to make queries in  $O(\log k)$  time per cell. Again, for an intersected cell  $\nabla$ , this calculates  $d_{\text{INT}}(b \cap \nabla, R)$ . By taking the maximum over these queries we get  $\vec{d}_{\text{H}}(b, R) = \max\{d_{\text{END}}(b, R), d_{\text{INT}}(b, R)\}$ . Since  $q$  intersects at most  $O(n/k)$  cells the query time is  $O((n/k) \log k + \log^3 k)$ . Hence, we obtain:

**Theorem 4.11.** *Let  $R$  be a set of  $n$  points in  $\mathbb{R}^2$ , and let  $k \in [1 \dots n]$  be a parameter. In  $O(nk^{1+\varepsilon} + n \log(n/k))$  time we can store  $R$  in a data structure of size  $O(nk^{1+\varepsilon})$  such that given a query line segment  $b$  we can compute  $\vec{d}_{\text{H}}(b, R)$  in  $O((n/k) \log k + \log^3 k)$  time.*

**Line segments.** When  $R$  is a set of line segments we use a similar approach. However, we do not use the standard lifting, since an extension of this lifting operation to line segments does not result in flat surfaces. Instead, for each line segment  $r \in R$  we consider the (graph of the) function that expresses the distance from a point  $q \in \mathbb{R}^2$  to  $r$ . Formally, for a line segment,  $r \in R$ , we define the surface  $r^\vee = \{(x, y, z) \mid z = d((x, y), r)\}$ . We build a vertical  $k$ -shallow cutting  $\Lambda$  of those surfaces (in this case cells of  $\Lambda$  are constant complexity pseudo-prisms). Observe that, as the line segments  $R$  do not intersect, the Voronoi diagram has linear complexity, and thus such a cutting exists. The resulting cutting has size  $O(n/k)$  and can be constructed in  $O(n \log^3(n) \lambda_{s+2}(\log n)) \in O(n \log^5 n)$  expected time [93]. The term  $\lambda_s(n)$  is the maximum length of a Davenport-Schinzel sequence of  $n$  symbols of order  $s$ , and  $s$  is a constant. For each cell  $\nabla$  we then proceed in the same way as before: we compute the Voronoi diagram of the segments in the conflict list of  $\nabla$ , and preprocess its edges for directed Hausdorff distance queries for lines (Lemma 4.8) and line segments (Theorem 4.9).

This yields the following result:

**Theorem 4.12.** *Let  $R$  be a set of  $n$  disjoint line segments in  $\mathbb{R}^2$  and let  $k \in [1 \dots n]$  be a parameter. In  $O(n \log^3(n) \lambda_{s+2}(\log n) + nk^{1+\varepsilon})$  expected time we can store  $R$  in a data structure of size  $O(nk^{1+\varepsilon})$  such that given a query line segment  $b$  we can compute  $\vec{d}_{\text{H}}(b, R)$  in  $O((n/k) \log k + \log^3 k)$  time, where  $s$  is a constant.*

## 4.5 The Directed Hausdorff Distance from Red to Blue

We compute the directed Hausdorff distance from a set of  $n$  red line segments to a blue query line segment  $b$ . First observe that the maximum distance from the red line segments to  $b$  is realized by an endpoint of a red line segment. Hence, we can immediately reduce the problem to computing the directed Hausdorff distance  $\vec{d}_{\text{H}}(R, b)$  from a set of  $O(n)$  red points  $R$  to  $b$ .

We use the method by Buchin et al. [37], which improved and extended the result from de Berg et al. [21]. Assume for ease of description that  $b = \overline{b_1 b_2}$  is horizontal

with  $b_1$  left of  $b_2$ . The directed Hausdorff distance  $\overrightarrow{d}_H(R, b)$  is then the maximum of: (i) the Euclidean distance between  $b_1$  and the farthest point from  $R$  left of  $b_1$ , (ii) the Euclidean distance between  $b_2$  and the farthest point from  $R$  right of  $b_2$ , or (iii) the maximum difference in  $y$ -coordinates between  $b$  and the points in  $R$  in the vertical slab between  $b_1$  and  $b_2$  (that is,  $\max\{|r_y - b_y| \mid r \in R \wedge (b_1)_x \leq r_x \leq (b_2)_x\}$ ). To determine  $\overrightarrow{d}_H(R, b)$  for a general segment  $b$  we therefore perform three queries: two queries search for the farthest point in  $R$  from a point  $b_i$  within a halfplane and one query searches for the farthest point to the supporting line of  $b$  within a slab perpendicular to  $b$ . Aronov et al. [11] present a data structure that uses  $O(n^{1+\delta})$  space and preprocessing time, and can be queried for the point in a halfplane farthest from  $b_i$  in  $O(2^{1/\delta} \log n)$  time, where  $\delta \in (0, 1)$  is an arbitrary constant. By storing the convex hull of  $R$  we can compute the point realizing the third query in  $O(\log n)$  time as well. Hence, we conclude:

**Lemma 4.13.** *Let  $R$  be a set of  $n$  line segments in  $\mathbb{R}^2$ , and let  $\delta$  be an arbitrary constant in the range  $(0, 1)$ . Using  $O(n^{1+\delta})$  time we can build a data structure of size  $O(n^{1+\delta})$  so that given a query line segment  $b$  we can compute  $\overrightarrow{d}_H(R, b)$  in  $O(2^{1/\delta} \log n)$  time.*

Together with Theorem 4.12, as  $O(n \log^3(n) \lambda_{s+2}(\log n)) \in O(n \log^5 n) \in O(n^{1+\delta})$  for any parameter  $\delta \in (0, 1)$  and constant  $s$ , we can formulate the main theorem of this chapter:

**Theorem 4.14.** *Let  $R$  be a set of  $n$  disjoint line segments in  $\mathbb{R}^2$ , and let  $k \in [1 \dots n]$  and  $\delta \in (0, 1)$  be parameters. In  $O(nk^{1+\varepsilon} + n^{1+\delta})$  expected time we can store  $R$  in a data structure of size  $O(nk^{1+\varepsilon} + n^{1+\delta})$  such that given a query line segment  $b$  we can compute  $d_H(b, R)$  in  $O((n/k) \log k + \log^3 k + 2^{1/\delta} \log n)$  time.*

## 4.6 Conclusion

We presented a data structure on a set of segments that allows queries of the following type: for a given segment, we can quickly determine the Hausdorff distance between the query segment and the set of segments the data structure is built on. We provided a mechanism to balance between space usage and preprocessing time for that data structure.

Note that Aronov et al. [11] also prove the existence of a data structure on  $n$  points with  $O(n \log^3 n)$  space that can be computed in polynomial time and answers halfplane proximity queries in  $O(\log n)$  time. This result could improve the size of the data structure in Theorem 4.14 to  $O(nk^{1+\varepsilon} + n \log^3 n)$  and the query time to  $O((n/k) \log k + \log^3 k)$ . However, there is no polynomial time algorithm known that constructs that data structure.

The next step is to work this data structure into a way to determine the Hausdorff distance between two sets of segments and update this distance while the sets change. As a first step, one goal is to make the query data structure dynamic.

# Chapter 5

## The $k$ -Fréchet Distance

We describe the  $k$ -Fréchet distance, a similarity measure that bridges between the Fréchet distance and the Hausdorff distance, by testing similarity between curves that resemble each other only piecewise. The parameter  $k$  denotes the number of subcurves into which we divide the input curves. The  $k$ -Fréchet distance allows two variants: the cover and the cut distance. We show that computing the cover variant of  $k$ -Fréchet is NP-hard, which is interesting since both (weak) Fréchet and Hausdorff distance are computable in polynomial time. We then show two algorithms for the cover variant: a polynomial time 2-factor approximation and an exact algorithm with exponential running time. The cut variant is not just NP-hard, but also APX-hard [38]. We present a polynomial time algorithm for the case  $k = 2$  in the cut variant. This chapter is based on two articles:

H. A. Akitaya, M. Buchin, L. Ryvkin and J. Urhausen. The  $k$ -Fréchet Distance: How to Walk Your Dog While Teleporting. *30th International Symposium on Algorithms and Computation (ISAAC)*, pages 50:1–50:15, 2019.

M. Buchin, L. Ryvkin and J. Urhausen. Computing the Cut Distance of Two Curves. *Abstracts of the 36th European Workshop on Computational Geometry (EuroCG)*, 2020.

Note that the dissertation of Leonie Ryvkin [106] is also in part based on that paper from Akitaya et al. [6].

### 5.1 Introduction

The Fréchet distance and the Hausdorff distance are two well-studied distance measures. As demonstrated in the introduction in Chapter 1, both are helpful in several applications. The Hausdorff distance can be computed more efficiently. However,

compared to the Fréchet distance, the Hausdorff distance provides us with less information by taking only the overall positioning of curves into consideration, not how they are traversed. We introduce the  $k$ -Fréchet distance as a distance measure in-between Hausdorff and (weak) Fréchet distance. As the name implies, it is closely related to the Fréchet distance but detects similarities between curves that resemble each other only piecewise. That is, we subdivide both curves into subcurves and compare the subcurves pairwise. There are two variants depending on whether the subcurves are allowed to overlap (cover distance) or not (cut distance). The parameter  $k$  in  $k$ -Fréchet denotes the number of subcurves into which we divide the input curves.

The new measure allows us to find similarities between curves that need to be cut and reordered to be similar under the Fréchet distance. For instance, this could be objects of rearranged pieces such as a set of curves of tourists visiting several sights in a city. If the  $k$ -Fréchet distance of two curves is small, the respective tourists used similar routes to get to the same sights. For  $k$  smaller than the number of sights, we can also conclude that the tourists visited some sights in the same order. Other examples would be chemical structures or handwritten characters and symbols. An example is displayed in Figure 5.1, where we compare two variants of writing the letter  $k$  by hand. Note that we deal with disconnected curves by concatenating the respective subcurves. Of course, we can easily identify that both of them are  $k$ s by using the Hausdorff distance to compare them to a “generic”  $k$ , but the  $k$ -Fréchet distance provides us with more information: the 2-Fréchet distance between the  $k$ s is large because the strokes are set differently. Those  $k$ s are unlikely to be written by the same person. The 3-Fréchet distance, however, is small, because the letter consists of at most three strokes in general.

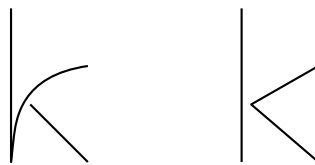


Figure 5.1: Two  $k$ s written in a different way. Their 2-Fréchet distance is large and their 3-Fréchet distance is small.

Characterizing the mentioned variants of the Fréchet distance next to the Hausdorff distance intuitively shows that the new distance measure bridges between weak Fréchet and Hausdorff distance. As is common for the Fréchet distance, we use the following analogy: we interpret our input curves as two paths, which have to be traversed by a man and a dog, each of them walking on one of the paths. For the (weak) Fréchet distance we ask for the length of the shortest leash so that man and dog can traverse their respective curves. They may choose their speeds independently. For the weak Fréchet distance, man and dog are allowed to backtrack.

The Hausdorff distance finds for each point on either curve the closest point on the other curve and takes the largest of the obtained distances. In terms of man and dog we do not ask for a traversal as such, we simply need that for any fixed position on either path there is a position on the other one such that man and dog can stand on their respective positions using a leash of fixed length. One could say they may teleport on their curves any number of times as long as both man and dog can reach all positions on their respective curves without exceeding the given maximum distance, that is, the leash length. The  $k$ -Fréchet distance limits this number of teleports by a constant, that is, we allow  $k - 1$  simultaneous teleports. In short, we want man and dog to traverse their paths continuously on  $k$  pieces and reach every position. Note that we use the weak Fréchet distance as the underlying distance measure. Recall that for the weak Fréchet distance we do not require the endpoints of the respective (sub)curves to be matched. For the cover distance, we allow that subcurves are traversed multiple times, for the cut variant we require that after a teleport, neither man nor dog come back to a position that was visited before that teleport.

**Related work.** Efficient algorithms were presented for computing the Fréchet distance and the weak Fréchet distance by Alt and Godau in 1995. They first introduced the concept of the free-space diagram, which is key to computing this distance measure and its variants [8]. Following their work, numerous variants and extensions have been considered. Here we mention only a few results related to our work. Alt, Knauer and Wenk compared Hausdorff to Fréchet distance and discussed  $\kappa$ -bounded curves as a special input instance [9]. In particular, they showed that for convex closed curves Hausdorff distance equals Fréchet distance. For curves in one dimension Buchin et al. [30] proved the equality of Hausdorff and weak Fréchet distance using the Mountain climbing theorem [65]. For computing the Hausdorff distance, Alt et al. [7] gave a thorough overview. Buchin [35] gave the characterization of these measures in free space, which motivated our study of  $k$ -Fréchet distance. Inspired by our research [6], Akitaya et al. [4] investigated a variant in which both agents traverse their respective curves moving alternately.

For  $c$ -packed curves, Driemel, Har-Peled and Wenk presented a  $(1 + \varepsilon)$ -approximation algorithm, which determines the Fréchet distance in  $O(cn/\varepsilon + cn \log n)$  time [53]. For the Fréchet distance between general polygonal curves, Colombe and Fox [47] recently presented a strongly subquadratic time algorithm with subexponential approximation ratio. Buchin et al. [31] slightly improved the original algorithm of Alt and Godau, while Bringmann [27] showed that unless SETH fails no strongly subquadratic algorithm for the Fréchet distance exists. An interesting variant bridging between the Fréchet distance and the weak Fréchet distance was presented by Gheibi et al.: they studied minimizing the length of the subcurves on which backtracking is necessary [63]. Buchin, Buchin and Wang studied partial curve matching, where they presented a polynomial-time algorithm to compute the “partial Fréchet similarity” [32], and a variation of this similarity was presented by Scheffer in [108]. Also,

Driemel and Har-Peled defined a Fréchet distance with shortcuts [52], which was proven to be the first NP-hard variant of the Fréchet distance by Buchin, Driemel and Speckmann [36].

**Overview.** Both Hausdorff and (weak) Fréchet distance are computable in polynomial time. Interestingly, computing both variants of the  $k$ -Fréchet distance, as distance measures that bridge between the two of them, proves to be NP-hard for general values of  $k$ . Note that the cut variant is even APX-hard. Nevertheless, we give several possibilities to deal with the hardness of the  $k$ -Fréchet distance. We start by formally defining the cover and cut distance. We then prove NP-hardness of a simpler auxiliary problem to gain some intuition in Section 5.2, followed by the main proof of this chapter showing that deciding the cover distance is NP-complete in Section 5.3. Then, we give a simple exponential-time algorithm in Section 5.4, and a factor 2 approximation in Section 5.4.1, both for the cover distance. For the cut distance, we present a polynomial-time decision algorithm for  $k = 2$  in Section 5.5.

### 5.1.1 Preliminaries

For the computation of the Fréchet distance, the concept of the free-space diagram was introduced by Alt and Godau [8]. For curves  $P, Q : [0, 1] \rightarrow \mathbb{R}$  and a value  $\varepsilon > 0$ , we define

$$F_\varepsilon(P, Q) = \{(s, t) \in [0, 1]^2 \mid d(P(s), Q(t)) \leq \varepsilon\}.$$

So, the *free space* is the set of all pairs of positions, one on each curve, that are at distance at most  $\varepsilon$  from each other. For piecewise-linear curves  $P$  and  $Q$ , the *free-space diagram* puts this information into an  $(n \times m)$ -grid where  $n$  and  $m$  are the numbers of segments in  $P$  and  $Q$  respectively. The free space is oriented as follows (see Figure 5.2): the point corresponding to the starting positions of both curves is in the bottom left, and moving horizontally (vertically) in the diagram corresponds to

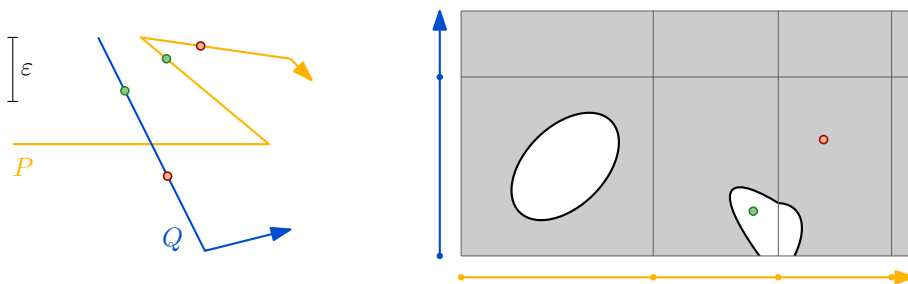


Figure 5.2: Two curves and the corresponding free-space diagram. Each point in the free-space diagram corresponds to a pair of positions, one on each curve.

moving along the curve  $P(Q)$ . For the rest of this chapter we assume that  $m \in \Theta(n)$  to simplify runtime expressions.

The Fréchet distance of two curves  $P$  and  $Q$  is at most a given value  $\varepsilon$  if and only if there exists a monotone path through the free space  $F_\varepsilon(P, Q)$  in the free-space diagram connecting the bottom left to the top right corner. For the weak Fréchet distance, this path need not be monotone and it may also start and end somewhere other than the corners of the diagram, as long as it touches all four boundaries.

We now define further terms: First, when we want to distinguish between the parameter spaces of  $P$  and  $Q$ , we use the notation  $[0, 1]_P$ , respectively  $[0, 1]_Q$ , instead of just  $[0, 1]$ . A *component* of a free space is a connected subset  $c \subseteq F_\varepsilon(P, Q)$ . We call a component *maximal* if it is inclusion maximal. A set  $S$  of components *covers* a set  $I \subseteq [0, 1]_P$  of the parameter space (corresponding to the curve  $P$ ) if  $I$  is a subset of the projection of  $S$  onto that parameter space, that is, if  $\forall x \in I: \exists c \in S, y \in [0, 1]_Q: (x, y) \in c$ . Covering on the second parameter space is defined analogously. This means the weak Fréchet distance is smaller than  $\varepsilon$  if there is one component in  $F_\varepsilon(P, Q)$  that covers both parameter spaces. Similarly, the Hausdorff distance can be tested by checking whether the set of all components covers both parameter spaces. In this chapter we extend this concept to also account for the number of components needed to cover the parameter spaces with the  $k$ -Fréchet distance.

**Definition.** We define two variants of the  $k$ -Fréchet distance: the *cover distance* and the *cut distance*. The cover distance  $d_{\text{cover}}(k, P, Q)$  is the minimum  $\varepsilon$  such that there is a set of at most  $k$  components of  $F_\varepsilon(P, Q)$  covering both parameter spaces. The intervals covered by the components are allowed to overlap, thus the cover distance between  $P$  and  $Q$  can also be defined as the minimum  $\varepsilon$  with a set of at most  $k$  *maximal* components of  $F_\varepsilon(P, Q)$  covering both parameter spaces. In contrast, the cut distance  $d_{\text{cut}}(k, P, Q)$  is the minimum  $\varepsilon$  such that there is a set of at most  $k$  components of  $F_\varepsilon(P, Q)$  that *uniquely* cover both parameter spaces, that is, each point in  $[0, 1]_P$  or  $[0, 1]_Q$  is covered by exactly one of the components in the selection.

By definition, the  $k$ -Fréchet distance lies in-between the Hausdorff and the (weak) Fréchet distances. As the cut variant is more restrictive than the cover variant, we have the following:

$$d_{\text{H}}(P, Q) \leq d_{\text{cover}}(k, P, Q) \leq d_{\text{cut}}(k, P, Q) \leq d_{\text{wF}}(P, Q) \leq d_{\text{F}}(P, Q).$$

Figure 5.3 shows a comparison. Also, the  $k$ -Fréchet distances decrease as  $k$  increases: for  $k = 1$  they equal the weak Fréchet distance, whereas for  $k$  sufficiently large they equal the Hausdorff distance.

Recall that in the introduction in Chapter 1, we defined both the (weak) Fréchet distance and the Hausdorff distance without using the definition of the free space. We can do the same for the variants of the  $k$ -Fréchet distance. We display definitions for all five distance measures below in a similar style to facilitate their comparison.

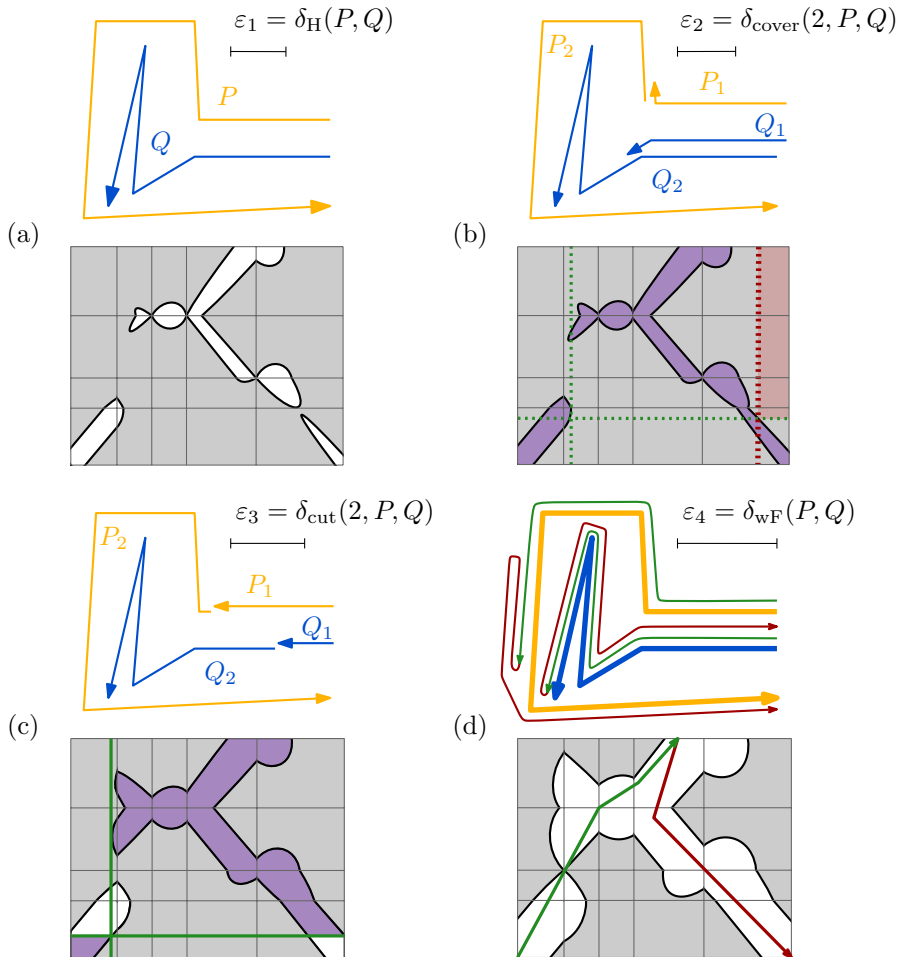


Figure 5.3: Comparison of distance measures. (a) The curves  $P$  and  $Q$  and their free-space diagram for the Hausdorff distance. Both parameter spaces can be covered by a component. (b) The subcurves  $P_1$  and  $P_2$  cover  $P$ ;  $Q_1$  and  $Q_2$  cover  $Q$ . We have  $d_{\text{wF}}(P_1, Q_1) \leq \varepsilon_2$  and  $d_{\text{wF}}(P_2, Q_2) \leq \varepsilon_2$ . Two components are sufficient to cover the parameter spaces, but cutting does not work, because by choosing the bottom left and top right cell the red section on the bottom parameter space would not be covered. (c) The subcurves  $P_1$  and  $P_2$  partition  $P$ ;  $Q_1$  and  $Q_2$  partition  $Q$ . We have  $d_{\text{wF}}(P_1, Q_1) \leq \varepsilon_3$  and  $d_{\text{wF}}(P_2, Q_2) \leq \varepsilon_3$ . Two components cover the parameter spaces without overlap. (d) A man and a dog can walk along  $P$  and  $Q$  while staying within distance  $\varepsilon_4$  by first following the green arrow and then the red arrow. Their movements correspond to a walk in the free space.



For curves  $P, Q$  and integer  $k > 0$ :

- Fréchet distance:  $d_F(P, Q) = \inf_{\sigma, \tau} \max_{t \in [0, 1]} d(P(\sigma(t)), Q(\tau(t)))$ , where  $\sigma, \tau : [0, 1] \rightarrow [0, 1]$  are orientation-preserving, bijective, and continuous functions.
- weak Fréchet distance:  $d_{wF}(P, Q) = \inf_{\sigma, \tau} \max_{t \in [0, 1]} d(P(\sigma(t)), Q(\tau(t)))$ , where  $\sigma, \tau : [0, 1] \rightarrow [0, 1]$  are surjective and continuous functions.
- Hausdorff distance:  $d_H(P, Q) = \inf_{\sigma, \tau} \max_{t \in [0, 1]} d(P(\sigma(t)), Q(\tau(t)))$ , where  $\sigma, \tau : [0, 1] \rightarrow [0, 1]$  are surjective functions.
- Cover distance:  $d_{\text{cover}}(k, P, Q) = \inf_{\sigma, \tau} \max_{t \in [0, 1]} d(P(\sigma(t)), Q(\tau(t)))$ , where  $\sigma, \tau : [0, 1] \rightarrow [0, 1]$  are surjective functions that are continuous everywhere except at the  $k - 1$  values  $\frac{1}{k}, \frac{2}{k}, \dots, \frac{k-1}{k}$ .
- Cut distance:  $d_{\text{cut}}(k, P, Q) = \inf_{\sigma, \tau} \max_{t \in [0, 1]} d(P(\sigma(t)), Q(\tau(t)))$ , where  $\sigma, \tau : [0, 1] \rightarrow [0, 1]$  are surjective functions that are continuous everywhere except at the  $k - 1$  values  $\frac{1}{k}, \frac{2}{k}, \dots, \frac{k-1}{k}$  and where we have  $\sigma(s) = \sigma(t) \vee \tau(s) = \tau(t) \implies \exists i \in \{1, \dots, k - 1\} : s, t \in [\frac{i}{k}, \frac{i+1}{k}]$ .

## 5.2 Gaining Intuition: The Box Problem

To give some intuition for the later proof that the cover variant of  $k$ -Fréchet is NP-complete, we first present a reduction from the well-known 3-SAT problem to the problem of covering two sides of a rectangle by selecting a number of smaller rectangles, or boxes, that are situated inside. This problem—we call it the box problem—mimics selecting the maximal components in the free space to cover the parameter spaces. However, in this section, we do not ask to find curves that realize this specific free space. Also, even though the intuition translates over to next section, this section and Section 5.3 are self-contained.

We want to reduce from the following classical NP-hard satisfiability problem [62]:

**3-SAT.** Input: a Boolean formula with  $n$  variables written as a conjunction of  $m$  clauses, where a clause is a disjunction of at most three literals. A literal is  $a$  or  $\neg a$ , where  $a$  is a variable. Output: “Yes” if there exists an assignment for the variables that satisfies the formula, “No” otherwise.

**Box problem.** Input: a set  $A$  of aligned, interior-disjoint rectangles  $b_i$ , their bounding box  $B$ , and  $k \in \mathbb{N}$ . Output: “Yes” if there exists a selection of at most  $k$  rectangles from  $A$  such that their union surjectively projects onto the bottom and left boundary of  $B$ , “No” otherwise.

Given any instance of a 3-SAT formula, we build a bounding box  $B$  containing a number of boxes  $b_i$  such that we can find a covering selection of size  $k$  if and only if there is an assignment that satisfies the formula. A *covering selection* of boxes is a subset of the boxes  $b_i$  that projects surjectively onto the bottom and left boundaries of  $B$ . For this we build boxes  $b_i$  that correspond to the variables such that any satisfying assignment of the variables implies a covering selection of the  $b_i$ .

First, note that we assume that no clause contains duplicates, that is, no clause is of the form  $v \vee v \vee w$ . The duplicates can be deleted without changing the boolean function induced by the formula. Additionally, we require that throughout the formula each literal appears at least once, that is, each variable appears at least once in its positive and in its negated form. Otherwise, we could simply define the occurring literal to be true (or false, respectively) and omit the clauses the literal appears in.

Now we give the detailed construction of our box problem instance derived from some 3-SAT formula: Let  $V = \{v_1, \dots, v_n\}$  be the set of variables and let  $C = \{c_1, \dots, c_m\}$  be the set of clauses. For each variable  $v_i$ , let  $a_i^+$  (respectively  $a_i^-$ ) be the number of clauses in which  $v_i$  appears positive (respectively negative), and let  $\{c_{i,1}^+, c_{i,2}^+, \dots, c_{i,a_i^+}^+\}$  (respectively  $\{c_{i,1}^-, \dots, c_{i,a_i^-}^-\}$ ) be the set of clauses in which  $v_i$  appears positive (respectively negative). Additionally we define the sums  $s_i^+ = \sum_{j=1}^i a_j^+$  and  $s_i^- = \sum_{j=1}^i a_j^-$ .

In the following we describe the placement of boxes, which is depicted in Figure 5.4. The number of rows and columns needed for the different gadgets is indicated in the figure. A box  $(x, y, w, \ell)$  designates the axis-aligned rectangle with unit height and width  $w$  whose bottom left corner has coordinates  $(x, y) \in \mathbb{R}^2$  with label  $\ell$ . The labels are literals and are later used in the proof of correctness.

**Variable gadget.** For each variable  $v_i$ , we place two boxes  $(i, i, 1, \neg v_i)$  and  $(i, i + n + s_n^+, 1, v_i)$ , and no other boxes are placed over the interval  $(i, i + 1)$  of the bottom boundary. That way, in order to cover that interval, at least one of those two boxes has to be chosen.

**Split gadget.** The split gadget ensures that we can propagate the assignment of a variable onto all clauses the variable takes part in. We build the splits used for the positive occurrences of the variables first. For each variable  $v_i$ , we place the box  $(1 + n + s_{i-1}^+, i, a_i^+, v_i)$  and the boxes  $(n + s_{i-1}^+ + j, n + s_{i-1}^+ + j, 1, \neg v_i)$ , for  $j \in \{1, \dots, a_i^+\}$ . For negated occurrences of  $v_i \in V$  we place the box  $(1 + n + s_n^+ + s_{i-1}^-, n + s_n^+ + i, a_i^-, \neg v_i)$  and the boxes  $(n + s_n^+ + s_{i-1}^- + j, 2n + s_n^+ + s_{i-1}^- + j, 1, v_i)$ , for  $j \in \{1, \dots, a_i^-\}$ .

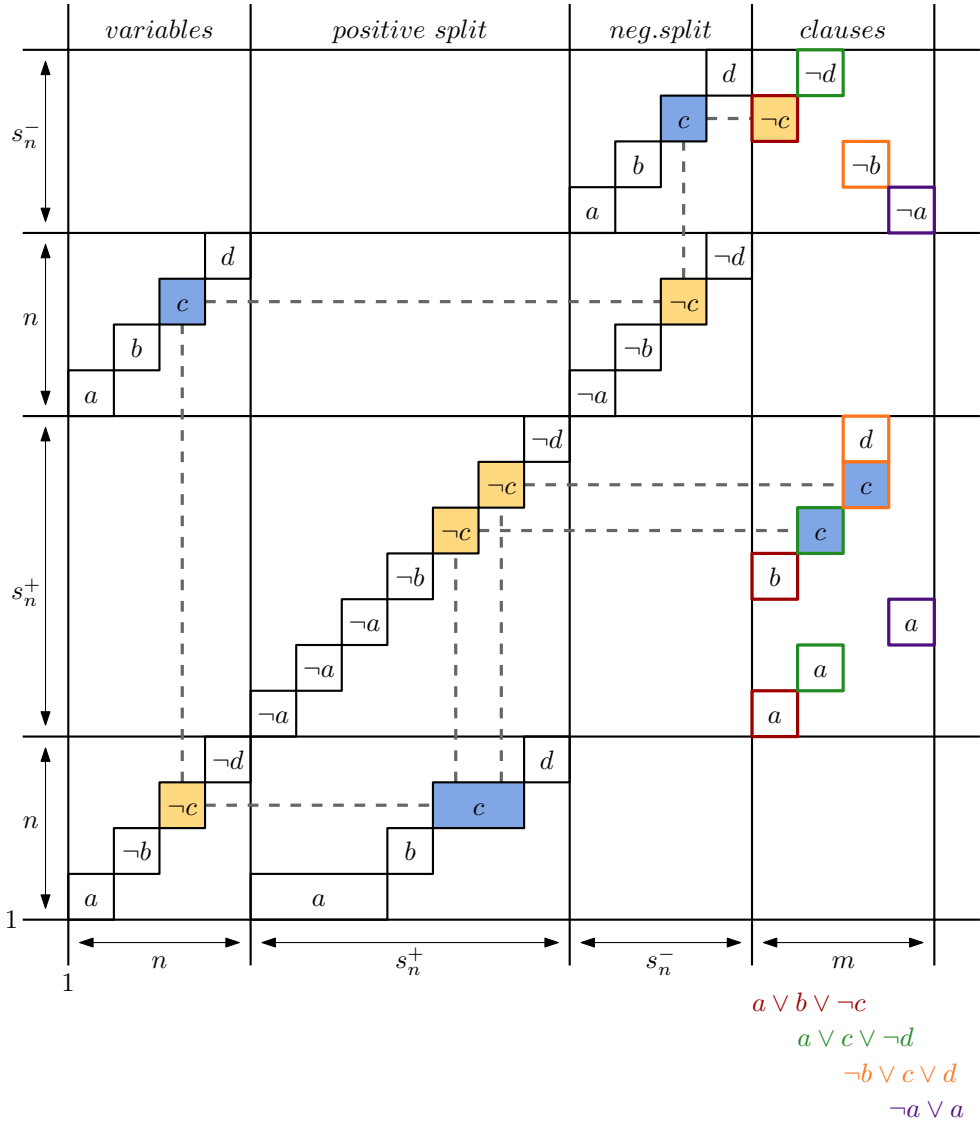


Figure 5.4: Construction of the box problem instance and propagation of assignment. All boxes labeled with  $c$  or  $\neg c$  are highlighted to show the staircase structure.

**Clause gadget.** We assign to each clause  $c_i$  the unit interval on the bottom boundary of  $B$  starting at  $I(c_i) = n + s_n^+ + s_n^- + i$ . For each literal of a clause  $c_i$  we place a box labeled with the respective literal above the unit interval  $[I(c_i), I(c_i) + 1]$ . To be precise, for each  $v_i \in V$  we place the boxes  $(I(c_{i,j}^+), n + s_{i-1}^+ + j, 1, v_i)$ , for  $j \in \{1, \dots, a_i^+\}$ , and  $(I(c_{i,j}^-), 2n + s_n^+ + s_{i-1}^- + j, 1, \neg v_i)$ , for  $j \in \{1, \dots, a_i^-\}$ .

Overall, we have  $4n + 2(m_1 + 2m_2 + 3m_3)$  boxes, where  $m_i$  is the number of clauses with  $i$  variables (and therefore  $m_1 + m_2 + m_3 = m$ ). Each unit interval  $(i, i + 1)$  with  $i \in \{1, \dots, 2n + s_n^+ + s_n^-\}$  on the left boundary of  $B$  can be covered by exactly two different boxes. The same holds for every unit interval  $(i, i + 1)$  with  $i \in \{1, \dots, n + s_n^+ + s_n^-\}$  on the bottom boundary. Note that for all these unit intervals, one of the boxes is labeled with a variable and the other one is labeled with the negated version of that variable, that is, one box is labeled  $v$  and the other one  $\neg v$ . Each interval  $[I(c), I(c) + 1]$  on the bottom boundary can be covered by as many boxes as the clause  $c$  contains literals. The labels of these boxes correspond to the variables contained within this clause. We set the bounding box  $B$  as the axis-aligned rectangle spanned by the points  $(1, 1)$  and  $(1 + n + s_n^+ + s_n^- + m, 1 + 2n + s_n^+ + s_n^-)$  and we set  $k = 2n + m_1 + 2m_2 + 3m_3$  so only half the boxes can be chosen. For a given boolean formula in conjunctive normal form, the set of boxes defined above can be determined in polynomial time.

**Theorem 5.1.** *The box problem is NP-complete.*

*Proof.* The box problem is in NP since for a given subset  $S$  of boxes one can test if the bounding box  $B$  is covered by simply marking the covered intervals, which can be done in polynomial time. To show NP-hardness, we prove that the box problem as constructed above has a solution if and only if the input 3-SAT formula has a variable assignment such that it evaluates to true.

' $\Leftarrow$ '. Let  $f : V \rightarrow \{\text{true}, \text{false}\}$  be an assignment of the variables that satisfies the 3-SAT formula. We set  $S = \{\text{box}(x, y, w, v) \mid f(v) = \text{true}\} \cup \{\text{box}(x, y, w, \neg v) \mid f(v) = \text{false}\}$ . The set  $S$  projects surjectively onto the bottom and left boundary of the bounding box  $B$  because each unit interval on the left boundary is covered by exactly one box. For most of the bottom boundary we also have that each interval is uniquely covered, but for the clauses columns we allow that more than one box per unit interval is chosen.

' $\Rightarrow$ '. Let  $S$  be a minimal set of boxes that covers the boundaries of the bounding box  $B$  with  $|S| = k$ . Because all boxes in the construction have height 1, this means that each unit interval on the left boundary of  $B$  is covered by exactly one box. For each variable  $v_i$ , look at the interval  $(i, i + 1)$  on the bottom boundary. If it is covered by a box labeled  $v_i$ , we set  $v_i$  to true, else it is covered by a box labeled  $\neg v_i$  and we set  $v_i$  to false. Now, let  $c$  be a clause. Its interval  $[I(c), I(c) + 1]$  is covered by at least one box  $b$ . We assume without loss of generality that  $b$  is labeled with a positive

literal  $v_i$ , as the other case is symmetric. Recall that each unit interval on the left boundary of  $B$  is uniquely covered by one box. Thus, the box  $b'$  labeled  $\neg v_i$  that is horizontally aligned with  $b$  is not part of the set  $S$ . Propagating further in the positive split gadget, the interval on the bottom boundary below  $b'$  is then covered by the box labeled  $v_i$  below it. Again, this implies that in the variables gadget, the box labeled  $\neg v_i$  is not in  $S$ , and the box labeled  $v_i$  is in  $S$ . It follows that  $v_i$  was assigned true, as desired.  $\square$

We can interpret the box problem as the problem of finding a selection of components in the free space that cover the parameter spaces. The small boxes can be seen as bounding boxes of maximal components (for the projection there is no difference) and the bottom and left boundaries of the large box  $B$  correspond to the respective parameter spaces. The above hardness proof, especially the construction of the boxes, provides us with the key ideas to prove hardness of the cover variant of  $k$ -Fréchet distance. Next, we construct actual curves where certain intervals on the parameter spaces of the free-space diagram each have two maximal components that could cover them. As with the box problem, the choice we make for one of those intervals determines the choices for other intervals as we still need to ensure that the selection size is minimal in the end. The propagation of choices works in the same manner for the box problem as for the cover distance problem.

### 5.3 NP-Completeness for the Cover Distance

Buchin and Ryvkin [38] proved that deciding the cut variant of  $k$ -Fréchet is NP-hard. In this section, we prove that deciding the cover variant for fixed  $\varepsilon$  is NP-hard. Like in Chapter 3, Section 3.2, we reduce from the NP-complete problem monotone rectilinear planar 3-SAT [20], albeit with different requirements for the shape of the drawing of the formula. Since for the cover distance, we can restrict our view to *maximal* components, in this section, we will omit the term “maximal” and just say “component”.

**Rectilinear Monotone Planar 3-SAT.** Input: a 3-SAT formula with only all positive or all negated variables per clause, embedded as a graph with rectilinear, non-crossing edges. Note that the requirements on the 3-SAT formula are different in this section compared to the formula needed to reduce from for the box problem in Section 5.2. The set of vertices consists of variable-, split- and clause-vertices; variable-vertices are drawn on a horizontal line that no edge crosses; clause-vertices for positive (negative) clauses are drawn above (below) this line; clauses are connected with the variables they contain with an edge or a path of edges and split-vertices. Output: “Yes” if there exists an assignment for the variables that satisfies the formula, “No” otherwise.

We can draw a graph corresponding to such a 3-SAT formula on a grid, see, for example, Figure 5.5. We assume that each variable appears in at least one positive and

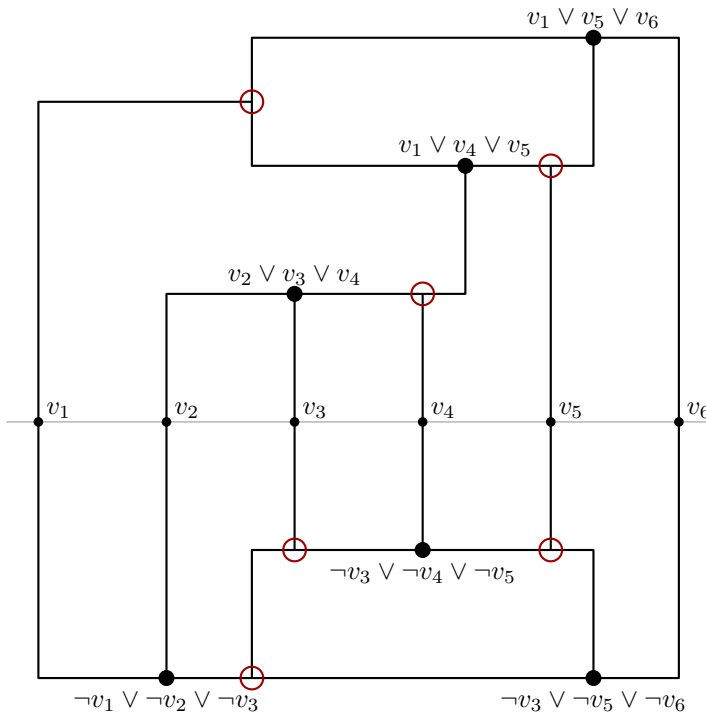


Figure 5.5: Instance of rectilinear monotone planar 3-SAT.

one negative clause. Otherwise, we could define the occurring literal to be true (or false, respectively) and omit the clauses the literal appears in. Also, we assume that each clause has exactly three literals. If there are clauses with less literals, we change them by repeating literals, for example the clause  $a \vee b$  can be changed into  $a \vee b \vee b$  without changing the satisfiability of the formula. This change can be made while preserving the fact that the formula is monotone, rectilinear and planar. Without loss of generality we can assume that the embedded graph has the following additional properties: each variable-vertex  $v$  has degree two and the incident edges are vertical at  $v$ . Also, split-vertices have degree three and the one edge perpendicular to both others connects to the variable vertex (possibly via other split vertices), while the other two edges connect to clauses.

For a given monotone rectilinear planar 3-SAT instance that is embedded as described above, let  $G$  be a drawing of the embedding. Without loss of generality we assume that  $G$  is drawn on the unit grid. We scale  $G$  such that vertices and bends are on grid vertices of the type  $(16x, 16y)$  and such that the distance between any two vertices, between any vertex and any bend, and between any two non-incident edges is at least 160. We translate  $G$  such that the variable vertices have  $y$ -coordinate 0.

**Cover Distance Problem.** Input: Two polygonal curves  $P$  and  $Q$ , a distance  $\varepsilon$  and a natural number  $k$ . Output: “Yes” if there exists a selection of at most  $k$  components in the free-space diagram  $F_\varepsilon$  such that their union projects surjectively onto both parameter spaces, “No” otherwise.

Our goal is to construct two curves  $P$  (red) and  $Q$  (blue) that mimic any input instance of a rectilinear monotone planar 3-SAT graph and show that in the free space resulting from these curves we can find a covering selection of size  $k$  if and only if there exists a satisfying assignment for the formula. The intuition of the construction is given first.

### 5.3.1 Intuition

Overall we create wire and clause gadgets to represent variables and clauses, where wires correspond to the edges of the input graph. Wire gadgets allow a boolean choice that is propagated consistently throughout the wire. Clause gadgets test whether at least one incoming wire carries an appropriate choice. An example can be seen in Figure 5.6.

We construct a wire alongside an edge of the 3-SAT formula as follows: The curves  $P$  and  $Q$  run parallel to the edge on both sides with some distance and have protrusions towards the edge. We construct  $P$  and  $Q$  as polylines with vertices embedded on a unit grid. We call the line segments of the curves on the outside of and parallel to the wire the *base parts*. We call the line segments protruding into the wire *spikes*. The base parts are not particularly relevant for the analysis because the segments forming them can only be covered by larger components (called *compulsory*) that are always part of any covering selection. The value  $\varepsilon = 10$  is chosen such that it is a bit larger than the distance between two adjacent spikes. It follows that the spikes induce components that are similar to the boxes of Subsection 5.2. compulsory We say that a spike  $s$  is *covered* by an adjacent spike  $t$  of the other curve if the component of the free-space diagram that covers the two intervals induced by these spikes is chosen for the covering selection. Each spike is covered by a single spike of the other curve (Lemma 5.2 below). In the end, we choose  $k$  equal the number of compulsory components plus the number of blue spikes. Since no components covers multiple spikes of the same color, each blue spike in any gadget can only be covered once (Lemma 5.3 below). The choice for blue spikes must be consistent along the wire to preserve minimality of  $k$ , and it encodes the assignment of the corresponding variable.

As displayed in Figure 5.6, the clause gadget features one yellow spike that can be covered by either one of the three blue spikes within its  $\varepsilon$ -neighborhood. Which one of their neighboring yellow spikes the blue ones cover is determined by the variable assignment and propagated throughout the wire, so if at least one of the variables is set to the correct value, the yellow spike at the center of the clause is covered.

Next, we need a number of other gadgets, too. As mentioned, the wires correspond

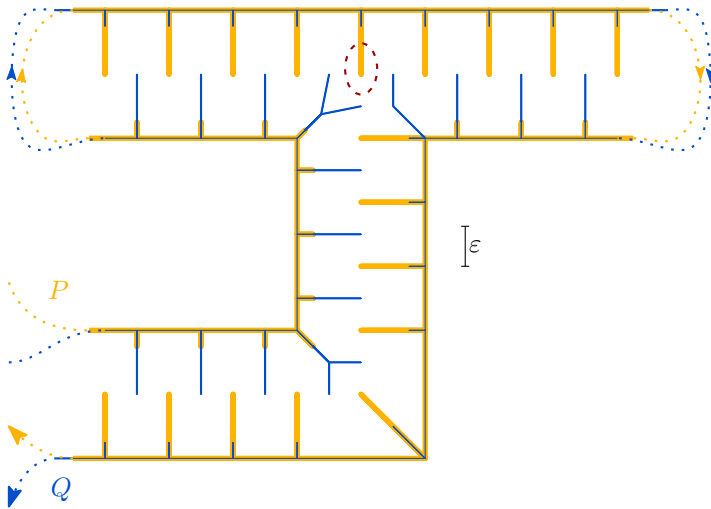


Figure 5.6: Three wires that connect to a clause gadget. The bottom wire has a bend. Each spike can be covered by one of the two adjacent spikes, with the exception of the clause spike (red) which can be covered by any of the three adjacent spikes.

to edges in the rectilinear monotone planar 3-SAT instance. To draw them coherently we need to make sure we can make 90 degree turns (so-called *bends*) and do T-crossings, that is, *split* a wire into two.

We need to treat remaining difficulties: First, we may need to change which of the curves has spikes on a specific side to draw the other gadgets consistently, so we also build a *color gadget* to switch the color pattern of the spikes. Second, there is a *connection gadget* that enables us to connect the opposite base parts of  $P$  and  $Q$ , respectively. The resulting two curves are closed, which we solve by using the *end gadget*. Finally, it remains to prove that our construction works in the sense that the curves have  $k$ -Fréchet distance  $\varepsilon$  if and only if the specific 3-SAT instance can be satisfied.

### 5.3.2 Gadgets

In this subsection we first describe the basic gadgets we need for our reduction. The gadgets are intricate, which is why we also need some other gadgets to be able to draw the entire construction using only two curves. The gadgets are build in a way such that no component covers more than one blue spike.

Assigning a variable to be true or false corresponds to choosing between one of two options in the free-space diagram, just as choosing between two boxes per row for the box problem. Therefore, we do not build specific variable gadgets but



encode the variables in the wires, which work as edges, that is, connections between other gadgets. Wires produce similar staircase structures as in Figure 5.4 in order to provide a choice between two components per uncovered interval of either parameter space.

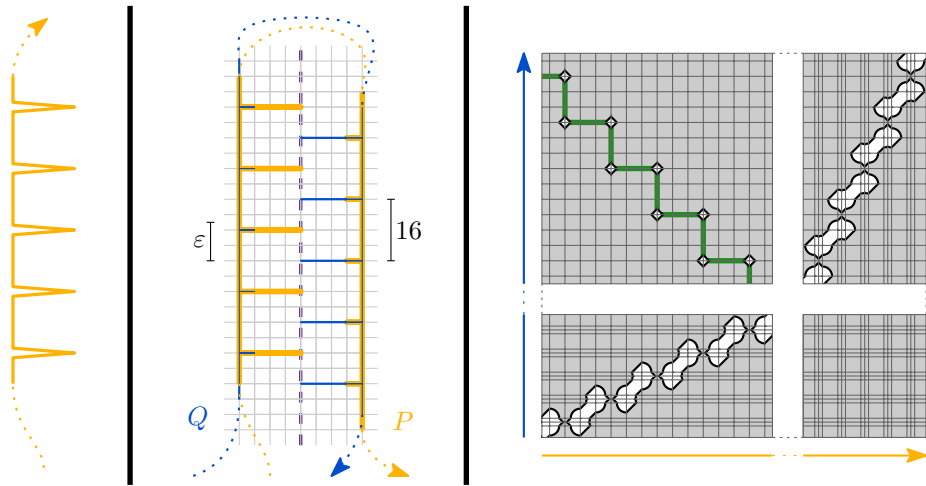


Figure 5.7: (Left) A perturbed part of the wire gadget with (long) spikes. (Middle) Wire gadget drawn on a grid where only each fourth line of the grid is drawn. The edge of the 3-SAT formula corresponding to the wire is dashed and purple. (Right) The resulting free-space diagram. It includes a staircase structure.

**Wire Gadget.** Figure 5.7 displays a wire gadget: we build the parallel base parts (shown in gray on the left side) that are studded with spikes of different lengths: each curve consists of two parallel base parts (in this figure they are drawn vertically) with shorter spikes on one and longer spikes on the other side (horizontal segments that are pointing inwards). Short spikes of one curve lie on top of the longer spikes of the other one. In the following the term *spike* refers to long spikes. Recall that vertices are embedded on a unit grid and that we set  $\varepsilon = 10$ . The distance between two spike tips of the same color is 16. We also show an underlying grid on which we draw our gadget, but note that we only drew every fourth line to improve readability. Two tips of spikes that are at distance at most  $\varepsilon$  are called *adjacent*. The spikes generate the small components forming a *staircase*, as each spike can be covered by one of two components induced by the two adjacent spikes of the opposite color. When we choose a component that covers the interval induced by a spike we say that the spike is *covered* by this component, respectively by the adjacent spike inducing this component.

Note that the segments of the base parts generate components that we definitely have to take. These larger components that are the only ones covering a certain interval on either parameter space are called *compulsory* components throughout the rest of this chapter. Sometimes smaller components are generated that are never part of an optimal selection of components, so called *unnecessary* components. An component is unnecessary, if for example it covers intervals that are covered by compulsory components.

At the end of the construction the value  $k$  is chosen to be equal to the number of compulsory components plus the number of blue spikes. It follows that each blue spike (on curve  $Q$ ) can only be covered by one single yellow spike (on curve  $P$ ). The spikes of  $Q$  determine the variable assignment: we define the *central spike* in each wire that encodes a variable: the central spike is the blue spike with  $y$ -coordinate 8, just above the corresponding variable vertex. If the central spike is covered by its upper yellow neighbor, the variable corresponding to this wire is set to `true`; if the central spike is covered by its lower neighbor, the variable is set to `false`. The choice made for the central spike is propagated throughout the rest of its wire. The fact that blue spikes have two choices, and that this choice can be propagated if each blue spike is uniquely covered, also holds for the other gadgets.

(Left) A perturbed part of the wire gadget with (long) spikes. (Middle) Wire gadget drawn on a grid where only every fourth line of the grid is drawn. The edge of the 3-SAT formula corresponding to the wire is purple. (Right) The resulting free-space diagram. It includes a staircase structure.

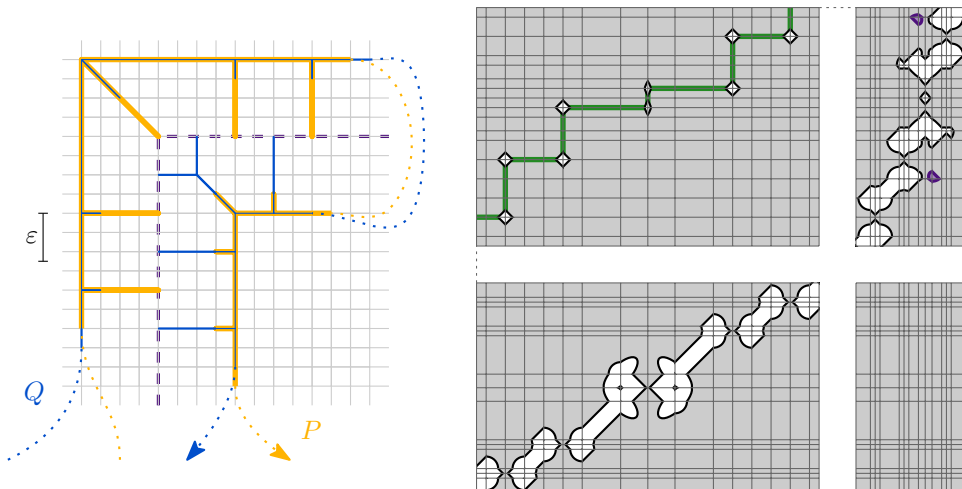


Figure 5.8: Bend gadget drawn on a (coarse) grid with its resulting free-space diagram. The two unnecessary components are purple.

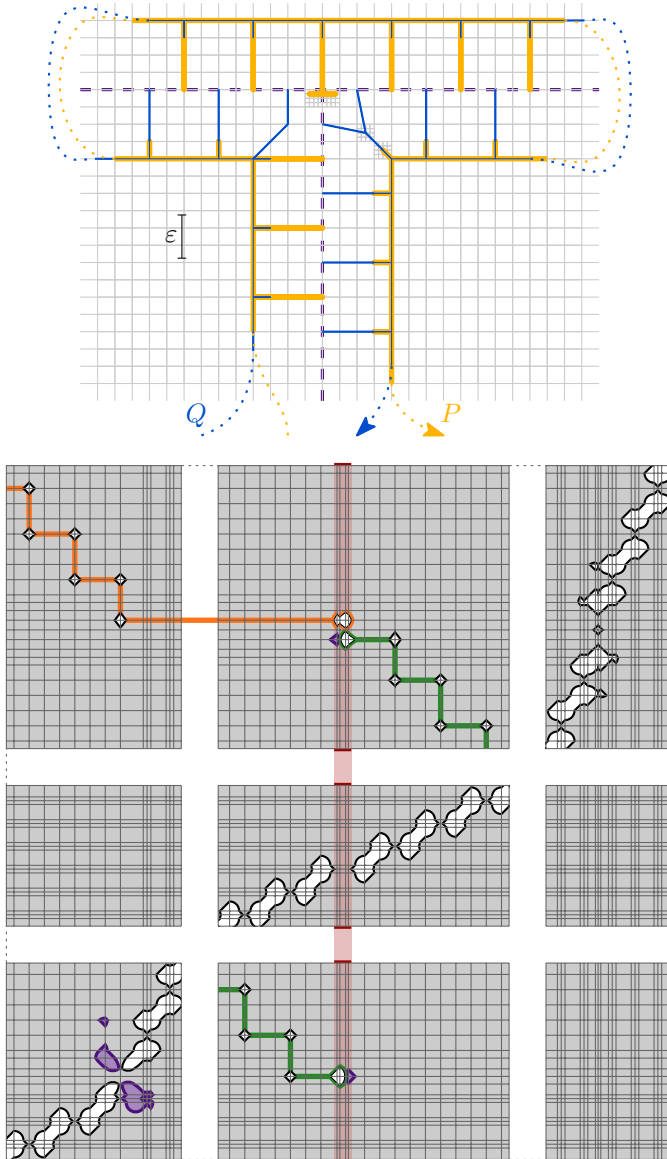


Figure 5.9: Split gadget drawn on a grid with its resulting free-space diagram displaying the three staircases corresponding to the two exit wires (green) and the entry wire (yellow). The red interval corresponding to the split spike can be covered by the orange or by both green components. The unnecessary components are purple.

**Bend Gadget.** Next we want to build 90 degree turns in order to ensure that we can form curves to represent the rectilinear edges of the input graph. We simply bend a wire to a right angle and insert a long diagonal spike at the outer corner as well as a “fork” with two spikes at the inner one, as displayed in Figure 5.8. The bend gadget ensures that the choice how to cover the spikes made for the wire on one side propagates to the other side of the bend.

**Split Gadget.** Now we focus on how to split a wire into two. Figure 5.9 shows the gadget, as well as its free-space diagram. The split is the most intricate gadget to draw, which is why we drew the underlying grid in its actual density for specific parts of the gadget and used the coarser grid, that is, only every fourth line, for the rest of the gadget. The  $T$ -shaped yellow spike in the middle of the split is called the *split spike*. Note that splits are directed in the sense that one wire, the *entry wire*, points toward the central spike while the other two, called *exit wires*, point toward clauses. In Figure 5.9 the entry wire is the bottom one.

The split spike can be covered by the adjacent blue spike from the entry wire or from both adjacent blue spikes from the exit wires at the same time. Thus, when the blue spike directly left of the split spike covers the yellow spike to its left, the split spike must be covered by the blue spike below it. The same holds for the blue spike directly right of the split spike. This observation is needed for the proof of Lemma 5.4 below.

**Clause Gadget.** The clause gadget in Figure 5.10 looks similar to the split gadget, the  $T$ -shaped spike is simply replaced by a regular spike, which we call *clause spike*. As a result, the clause spike can be covered by any of the three adjacent blue spikes. The three staircases in the free-space diagram correspond to the same wire parts of the curves as for the split gadget. As a result of omitting the  $T$ -shape we get one interval, corresponding to the clause spike, that can be covered by three components.

**Color Gadget.** Note that throughout all gadgets, the tips of yellow spikes are all on points of the type  $(16x, 16y)$ , for integers  $x$  and  $y$ . Thus, the bend, clause and split gadgets drawn on the corresponding places of the drawing  $G$  of the input graph can be connected with wires while alternating blue and yellow spikes. However, it is possible that when connecting a wire to a gadget, the yellow spikes are on different sides of the base curves. The color gadget flips the sides on which the yellow and blue spikes occur.

As displayed in Figure 5.11, the  $x$ -shaped component, which corresponds to the slightly tilted spike being within  $\varepsilon$ -distance of blue spike above it, connects the two staircases that stem from the wires to the sides of it. So, concerning coverage, the tilted spike has the same properties as a normal spike.

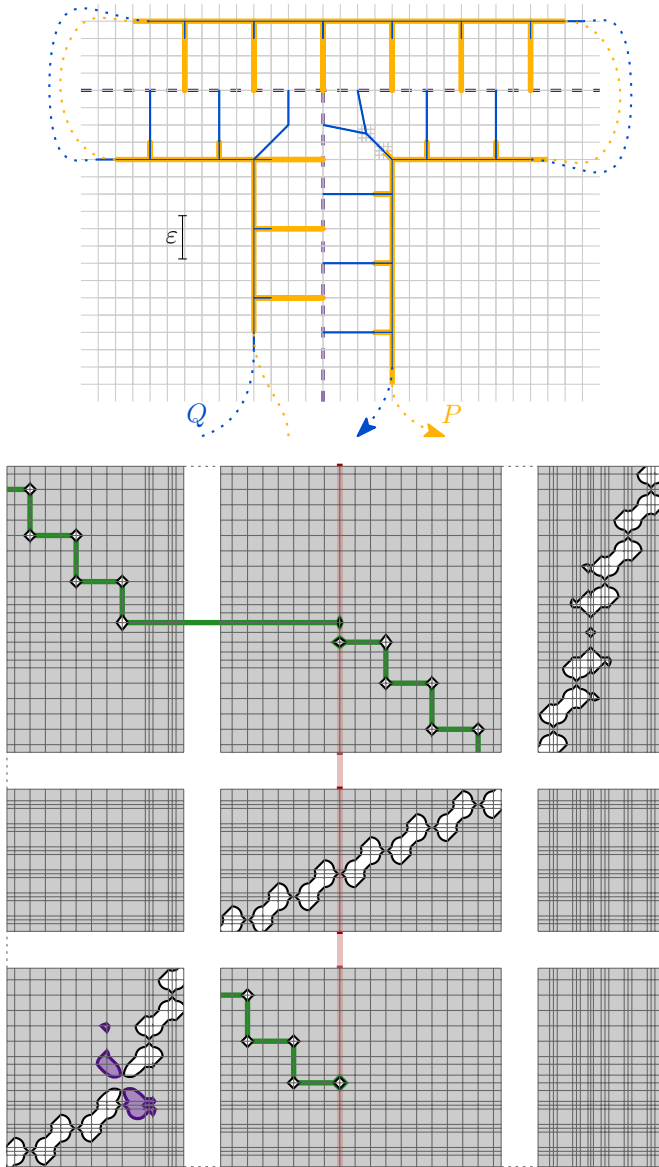


Figure 5.10: Clause gadget drawn on a grid with its resulting free-space diagram displaying the three staircases corresponding to the three connected wires. The red interval corresponding to the clause spike can be covered by any of the three green components. The unnecessary components are purple.

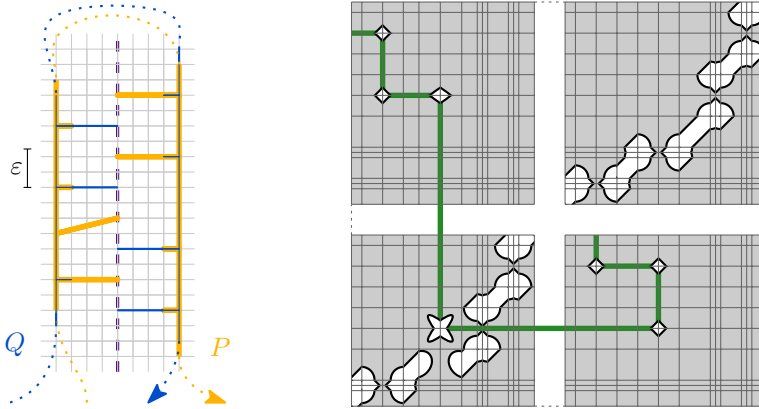


Figure 5.11: Color gadget drawn on a (coarse) grid with its resulting free-space diagram.

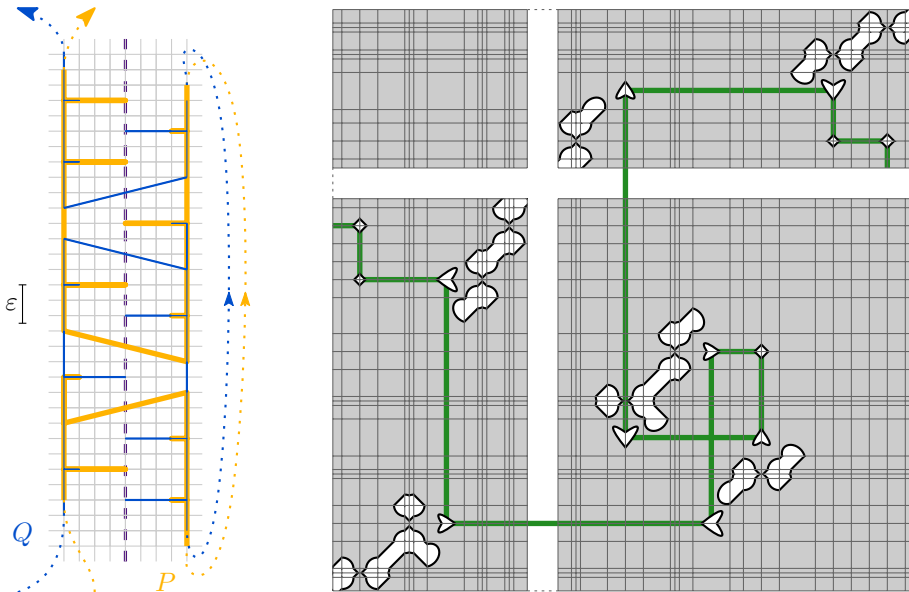


Figure 5.12: Connection gadget drawn on a (coarse) grid with its resulting free-space diagram.

**Connection Gadget.** Note that in all gadgets described above, the base curves for both sides are not connected. The connection gadget enables us to connect the base parts along a wire without interrupting the alternating spikes, that is, the staircase structures in the free-space diagram. Figure 5.12 shows the connection gadget: the four diagonal segments are also called spikes as they display the same behavior, that is, they need to be covered by one of the two adjacent spikes, and no component can cover more than one blue spike.

**End Gadget.** When combining these gadgets, we obtain two closed curves. As we set out to prove NP-hardness for non-closed curves, we need the end gadget. When comparing the end gadget in Figure 5.13 with a normal wire in the free-space diagram, the staircase is still intact and it has one compulsory component less.

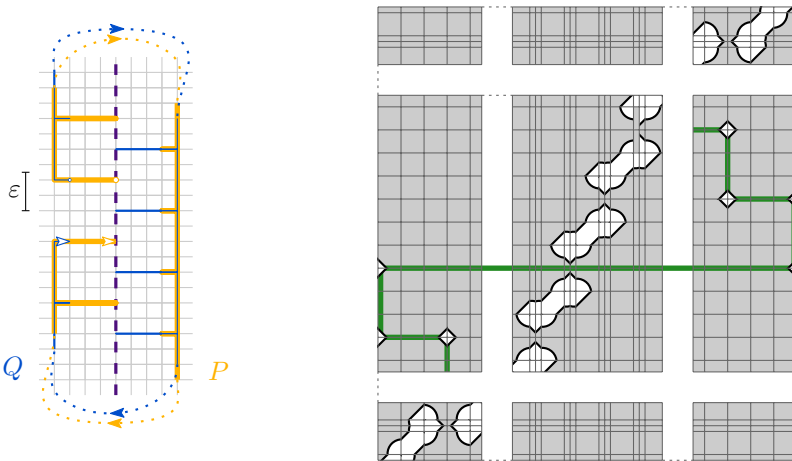


Figure 5.13: End gadget drawn on a (coarse) grid with its resulting free-space diagram.

### 5.3.3 Reduction

Now, given a drawing  $G$  of a 3-SAT formula as specified in Subsection 5.3.2. We start the construction of the Cover Distance Problem instance by placing a clause (split, bend) gadget at each clause (split, bend) of  $G$ . We then connect these gadgets along the edges using the wire gadgets. On edges where this is not possible because the colors of the spikes on the sides of the wire do not match, we first place a color gadget before connecting the gadgets. This results in multiple closed curves.

Now, let  $e$  be an edge such that the base curves along the wire drawn on  $e$  are not connected. We add a connection gadget on  $e$ . Since the distance between

clause/split/bend gadgets is at least 160, there is enough space on  $e$  to place one color gadget and one connection gadget without interfering with the gadgets at both ends of the edge. We repeat this step until we just have two closed curves, one yellow and one blue. Lastly, we replace a piece of wire with the end gadget resulting in our two curves  $P$  and  $Q$ . We observe that the following holds:

**Lemma 5.2.** *A spike can only be covered by an adjacent spike of the other curve and no component covers two spikes of the same curve.*

We set  $k = k_b + k_c$ , where  $k_b$  is the number of blue spikes and  $k_c$  is the number of compulsory components. Thus, together with Lemma 5.2, we can make an even stronger statement for blue spikes:

**Lemma 5.3.** *In a selection of  $k$  components covering both parameter spaces, each blue spike is covered by exactly one component.*

### 5.3.4 Proof of Correctness

We prove that  $d_{\text{cover}}(k, P, Q) \leq \varepsilon$  if and only if the input 3-SAT instance can be satisfied.

' $\Rightarrow$ '. Let  $g : V \rightarrow \{\text{true}, \text{false}\}$  be an assignment of the variables  $v \in V$  that satisfies the 3-SAT formula. Now we explain how to cover the parameter spaces with  $k$  components. First, for each interval of the parameter spaces that can only be covered by a single component, we choose that component. After this step, by definition, we have chosen all the compulsory components. It follows that only spikes remain to be covered. Then, for each variable  $v$ , if  $g(v)$  is true, we cover the central spike of the corresponding wire by the adjacent yellow spike  $y$  above it. We then propagate this choice: the blue spike  $b$  above that yellow spike  $y$  is covered by the yellow spike above  $b$  and so on. For each variable set to false we do the inverse, that is, we cover the central spike by the yellow spike below it and then propagate.

Per assignment, each blue spike is covered once. As we propagate the choices made at the central spikes, all yellow spikes not in clauses are covered. Lastly, each clause spike is covered as for each clause, one of the variables is set to the correct value. This means that both parameter spaces are completely covered by our selection.

' $\Leftarrow$ '. In the following, we assume that we have a selection of components  $S$  with  $|S| = k$  that covers the parameter spaces. By Lemma 5.3, each central spike is covered once. We now define an assignment of the variables  $g : V \rightarrow \{\text{true}, \text{false}\}$  as follows: for each variable  $v$ , if the (blue) central spike  $b(v)$  of  $v$  is covered by the yellow spike above it, we set  $g(v)$  to true, else  $b(v)$  is covered by the yellow spike below it and we set  $g(v)$  to false. This assignment satisfies the 3-SAT formula, as proven below.

Let  $c$  be a clause. The (yellow) clause spike  $y(c)$  of  $c$  is covered by at least one of the three adjacent blue spikes, by Lemma 5.2. Let  $b(c)$  be one of those blue spikes that covers  $y(c)$ , and let  $v$  be the variable that corresponds to the wire of  $b(c)$ .



We define the *staircase distance* of two spikes  $a, b$  of a variable  $v$  to be the number of spikes corresponding to that variable in-between them plus one. We define that a spike has distance zero to itself. Two adjacent spikes have staircase distance 1, two spikes of the same color have an even staircase distance. A spike  $a$  is *closer* to a spike  $b$  (within the same variable) than a third spike  $c$  if the staircase distance of  $a$  and  $b$  is smaller than the staircase distance of  $b$  and  $c$ . Intuitively, the staircase distance counts the number of “steps” between two spikes on the staircase corresponding to a wire in the free space.

**Lemma 5.4.** *The central spike  $b(v)$  is covered in such a way that the induced assignment value  $g(v)$  of  $v$  fulfills  $c$ .*

*Proof.* We prove by induction that the central spike  $b(v)$  is covered by the adjacent yellow spike that is closer to the clause spike  $y(c)$ . Let  $2j$  be the distance between  $b(c)$  and  $b(v)$ . For  $i \in \{0, \dots, j\}$ , let  $b_i$  be the (blue) spike between  $b(c)$  and  $b(v)$  whose distance to  $b(c)$  is  $2i$ . The induction hypothesis is that  $b_i$  is covered by the adjacent yellow spike that is closer to the clause spike  $y(c)$ . For  $i = 0$  this is true, because  $b_0 = b(c)$  is covered by  $y(c)$ .

Now assume that the hypothesis holds for any  $i \in \{0, \dots, j\}$ . Let  $y$  be the yellow spike between  $b_i$  and  $b_{i+1}$ . Together with Lemma 5.3 the induction hypothesis implies that  $b_i$  does not cover  $y$ . If  $y$  is simply part of a wire, it can only be covered by  $b_i$  or  $b_{i+1}$ , meaning  $y$  is covered by  $b_{i+1}$ . If  $y$  is a split spike, we know that it can either be covered by the spike within the entry wire, which is  $b_{i+1}$ , or  $y$  can be covered by both other adjacent spikes simultaneously, that is, by  $b_i$  and another blue spike. Again, it follows that  $b_{i+1}$  covers  $y$ , finishing the proof by induction. Since  $b_j = b(v)$ , due to the monotonicity of the formula, the central spike is covered in such a way that the induced assignment value  $g(v)$  of  $v$  fulfills  $c$ .  $\square$

**Theorem 5.5.** *It is NP-hard to decide whether  $d_{\text{cover}}(k, P, Q) \leq \varepsilon$  for given polygonal curves  $P$  and  $Q$ , integer  $k$ , and  $\varepsilon > 0$ .*

We can test in polynomial time whether the union of a selection of maximal components covers the parameter spaces. Thus the problem of deciding the cover variant of the  $k$ -Fréchet distance lies in NP.

## 5.4 Algorithms for the Cover Variant

We present two algorithms. For given curves  $P$  and  $Q$ , integer  $k$  and  $\varepsilon > 0$ , the first algorithm tests whether  $d_{\text{cover}}(k, P, Q) \leq \varepsilon$  in exponential time. If the inequality holds, the algorithm returns a covering selection of at most size  $k$ . Note that this algorithm can be extended to an optimization algorithm that minimizes  $k$  for a fixed  $\varepsilon$ , for example using binary search.

For given curves  $P$  and  $Q$ , and  $\varepsilon > 0$ , the second algorithm is a 2-approximation algorithm which runs in  $O(n^2 \log n)$  time. It calculates a covering selection of size  $k'$ , where  $k'$  is at most  $2 \min_k \{d_{\text{cover}}(k, P, Q) \leq \varepsilon\}$ .

Both algorithms start by computing the free-space diagram  $F_\varepsilon(P, Q)$  in  $O(n^2)$  time [8]. Such a free-space diagram has  $n^2$  cells and therefore at most  $n^2$  components.

The brute force approach simply checks for all selections of  $k$  components of the free space whether their joint projections cover both parameter spaces surjectively. That means we have to check at most  $\binom{n^2}{k}$  possible combinations of components resulting in a runtime of  $O(k \cdot n^{2k})$  for fixed  $k$ , which is of course only feasible for small  $k$ . Therefore, we can compute the answer to the decision problem for the cover distance with  $k = 2$  in  $O(n^4)$ . Since  $\binom{m}{k} \leq 2^m$  holds for any  $m > k$ , our runtime is upper-bounded by  $O(n \cdot 2^{n^2})$  for general  $k$ .

**Lemma 5.6.** *For given polygonal curves  $P$  and  $Q$ , integer  $k$ , and  $\varepsilon > 0$ , we can test in  $O(k \cdot n^{2k})$  time whether  $d_{\text{cover}}(k, P, Q) \leq \varepsilon$  holds.*

### 5.4.1 Approximation Algorithm

We can also approximate the size of an optimal solution. The main idea of our algorithm is to greedily find minimal covering selections for each parameter space individually and combine those selections into an overall solution in the end. Given the free-space diagram, we first project all components onto the parameter spaces. This results in two set of intervals, the set  $I_P$  covering the first parameter space and the set  $I_Q$  for the second parameter space, see Figure 5.14.

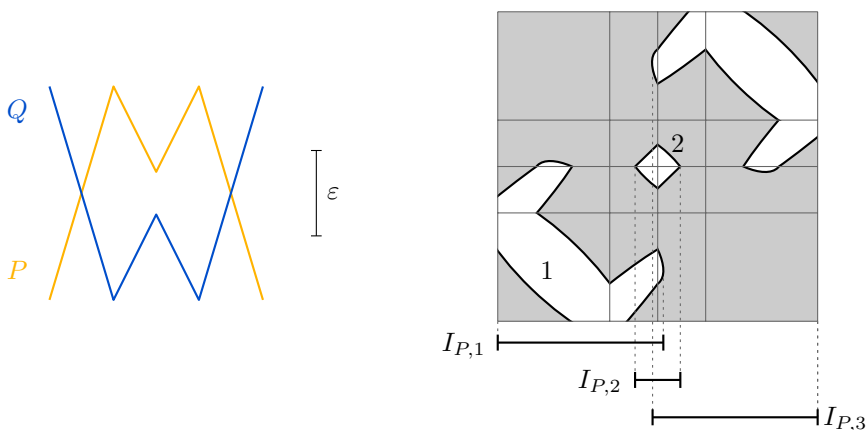


Figure 5.14: The projection onto the first parameter space and the resulting elements of  $I_P$ .

Now we determine a minimal subset  $S_P$  of  $I_P$  that covers  $[0, 1]_P$  and do the same for  $Q$ , if such subsets exist, that is, if  $d_H(P, Q) \leq \varepsilon$ . This can be done in  $O(n^2 \log n)$  time using a simple greedy algorithm [49]. As output we have two selections of intervals,  $S_P$  and  $S_Q$ . The intervals correspond to components. We build the union of both lists and output the selection of components  $S$  that contributed at least one of the chosen intervals.

As both  $S_P$  and  $S_Q$  are minimal in order to cover both parameter spaces individually, we know that for the minimal solution  $S_{\text{opt}}$  covering both interval spaces at the same time, we have  $|S_P| \leq |S_{\text{opt}}|$  and  $|S_Q| \leq |S_{\text{opt}}|$ . Thus  $|S| \leq 2|S_{\text{opt}}|$ . Schäfer proves that the approximation factor 2 is indeed tight [107] for our algorithm, that is, Schäfer shows that there is a class of instances for which this algorithm does produce a solution with twice the number of components needed.

**Lemma 5.7.** *For given polygonal curves  $P$  and  $Q$ , and  $\varepsilon > 0$ , we can approximate the minimum  $k$  with  $d_{\text{cover}}(k, P, Q) \leq \varepsilon$  up to a factor 2 in  $O(n^2 \log n)$  time, if such a  $k$  exists.*

## 5.5 Computing the Cut Variant

Deciding the cut distance problem is NP-hard and optimizing  $k$  is APX-hard [38]. Here, we give a polynomial-time algorithm for  $k = 2$ .

### 5.5.1 Cut Placements

The first difficulty is that there are infinitely many possibilities of placing a cut. For  $k = 2$ , we can reduce this amount to a finite set of discrete positions. We define *interesting points* to be local extrema of a maximal component's boundary. We call a horizontal or vertical cut line through an interesting point an *interesting line*. Also, for a fixed horizontal (vertical) line  $\ell$ , we define *artificial points* as any point of intersection between  $\ell$  and the boundary of the free space. We want cut lines  $\ell$  and  $h$  such that there are two components  $c_1$  and  $c_2$  contained in two opposing quadrants formed by  $\ell$  and  $h$ , such that  $c_1$  and  $c_2$  together cover both parameter spaces. Such a placement of cut lines is called *valid*.

**Lemma 5.8.** *If there exists a valid placement of cut lines in  $F_\varepsilon(P, Q)$  for  $k = 2$ , there exists another valid placement of cut lines where at least one line is an interesting line.*

*Proof.* For a vertical line  $\ell$  (a point  $a$ ), we define  $\ell_P(a_P)$  as the point on  $[0, 1]_P$  covered by  $\ell$  ( $a$ ). Assume we are given a valid placement of cut lines and assume none of them features an interesting point. The cut lines subdivide the free-space diagram into four quadrants of which two opposing ones are covered by components  $c_1$  and  $c_2$ .

We fix the horizontal cut line  $h$ . It intersects  $c_1$  and  $c_2$  at artificial points  $a_1, \dots, a_k$  sorted from left to right. Note that moving the vertical cut line to the right from  $\ell$  to a line  $\ell'$  can change the coverage in general, see Figure 5.15.

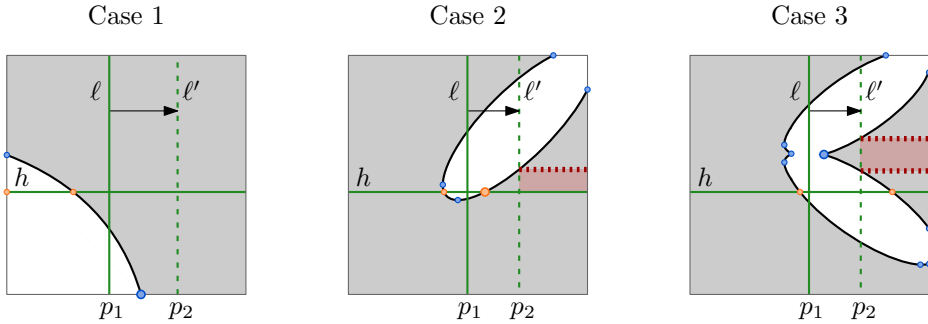


Figure 5.15: Given valid cut lines, moving one cut line beyond interesting or artificial points may alter coverage of a quadrant. Interesting points are blue, artificial points for  $h$  are orange.

1. The left component may not cover a subinterval of  $[\ell_P, \ell'_P]$  on the curve  $P$ .
2. The right component may no longer cover an interval on the curve  $Q$ .
3. The right component may become disconnected.

These cases can only occur when there is an interesting or artificial point between  $\ell$  and  $\ell'$ . Symmetrically, this holds for moving  $\ell$  to the left, or moving the horizontal line  $h$ . Thus, as long as we do not move cut lines past interesting or artificial points, the cut lines stay valid.

Let  $I = [(a_i)_P, (a_{i+1})_P] \subseteq [0, 1]_P$  be the interval induced by adjacent artificial points containing  $\ell_P$ . If there is an interesting point that covers a point in  $I$ , we move  $\ell$  to the closest interesting point in  $I$ . This can be done without crossing an artificial or interesting point. Else we move  $\ell$  to  $a_i$ . The intersection  $z$  of  $\ell$  and  $h$  now lies on the boundary of a maximal component. Next, we can move  $\ell$  and  $h$  simultaneously such that  $z$  moves along the component's boundary until one of the cut lines reaches an interesting point. As both lines move simultaneously they do not cross any artificial points while moving.  $\square$

Note that for  $k > 2$ , this approach does not work. We know that moving one line without crossing interesting or artificial points could necessitate moving another line, but for  $k > 2$  this may cascade further, causing multiple other lines to move as well. As shown in Figure 5.16, this may create loops, meaning one is unable to move any lines, even though no line is interesting.

## 5.5.2 Algorithm

We present an algorithm that decides if  $d_{\text{cut}}(2, P, Q) \leq \varepsilon$  for a given  $\varepsilon > 0$ . We first test if there exists a valid placement of cut lines in  $F_\varepsilon(P, Q)$  for  $k = 2$  where the vertical

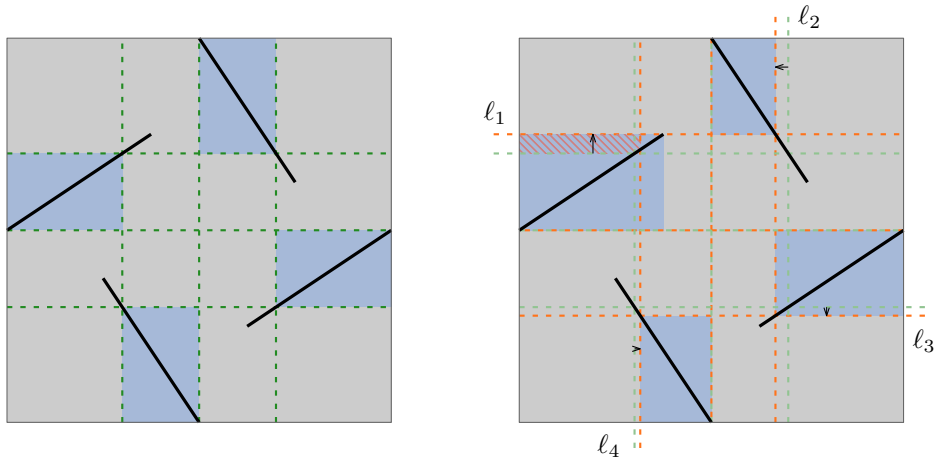


Figure 5.16: (Left) Given these four components, we can apply valid cut lines (green). (Right) We try to move the cut lines to interesting points by moving line  $\ell_1$  up. This forces  $\ell_2$  to move left, which in return moves  $\ell_3$  down, which moves  $\ell_4$  to the right. Line  $\ell_4$  now intersects the left component in such a way that the resulting component no longer covers the red area.

line is an interesting line and where one of the components covering the parameter spaces is contained in the bottom left quadrant and the other one in the top right quadrant. The other cases can be tested in a symmetrical way.

The algorithm scans all interesting lines from left to right. For each interesting line  $\ell_i$ , it determines if there is a component that intersects  $\ell_i$  as well as the bottom and left boundaries of the free-space diagram. Per interesting line  $\ell_i$ , there is at most one such component, called a *candidate component*. Furthermore, we determine for each candidate component  $c_i$  for which horizontal lines  $h$ , there is a subcomponent of  $c_i$  below  $h$  that still covers that bottom left quadrant. We call such horizontal lines *candidate lines*. As components are connected sets, the  $y$ -coordinates of the candidate lines for a candidate component of an interesting vertical line  $\ell_i$  lie in an interval. Thus, we determine the interval  $I_i^{\swarrow} = [y_i^{\min}, y_i^{\max}]$  such that each line  $h \equiv y = y'$ , with  $y' \in I_i^{\swarrow}$ , is a candidate line.

We then repeat this step to determine the candidate components covering the top right quadrant and the respective candidate lines, described by the intervals  $I_i^{\nearrow}$ . Finally, we once again iterate over all vertical lines where there are both a bottom left candidate component and a top right candidate component. We then test if there is a horizontal line that is a candidate line for both candidate components, that is, we test  $I_i^{\swarrow} \cap I_i^{\nearrow} \neq \emptyset$ . Such a line leads to a valid cut.

First we determine the candidate components and the maximum height of their

candidate lines. We use the union-find data structure for this purpose, see Chapter 22 in [49]. By means of this data structure, one can merge two groups of elements and one can determine if two elements are in the same group, both in  $O(\alpha(n))$  time, where  $n$  is the number of elements in this data structure and  $\alpha(\cdot)$  is the inverse Ackermann function. Note that  $\alpha(n) \in o(\log n)$  holds.

**Precomputation.** Recall that interesting lines stem from the local extrema of maximal components. So, we can compute all interesting vertical lines and sort them from left to right in  $O(n^2 \log n)$  time. This gives us the lines  $\ell_0, \ell_1, \dots$ , where  $\ell_i \equiv x = x_i$  and  $x_0 = 0$ . We also determine how the components of adjacent cells are connected. For each pair of adjacent cells  $\{C_1, C_2\}$  containing components, if those components intersect, we determine the  $x$ -coordinate  $x'$  of a leftmost point of intersection. We then also determine  $i$  such that  $x_i \leq x' < x_{i+1}$ . We store this pair of cells  $\{C_1, C_2\}$  in a list  $A_i$ .

**Initialization.** Each element in our union-find data structure represents a component. For each component  $c$ , we store the following properties:

- if  $c$  intersects the left boundary,
- if  $c$  intersects the bottom boundary,
- the  $x$ -coordinate of a rightmost point in  $c$  and the column  $o$  of cells containing this point,
- information to determine the  $y$ -coordinate of a topmost point in  $c$  left of a line  $\ell_i$  in  $O(1)$  time; namely:
  - the  $y$ -coordinate of a topmost point in  $c$  that is not in column  $o$ ,
  - the topmost cell in column  $o$  containing part of  $c$ .

At the start we create an element within our union-find data structure for each component within a cell and we set  $i = 0$ . In each step we aim to find candidate components for the line  $\ell_i$ .

**Step  $i \rightarrow i + 1$ .** See Figure 5.17. We go through all pairs of adjacent cells  $\{C_1, C_2\}$  in the set  $A_i$  and combine the components within those cells, if the corresponding components are not already merged in the union-find data structure. We update the properties of the newly formed component in  $O(1)$  time. If, during a merge, we get a component that intersects  $\ell_{i+1}$  and both bottom and left boundaries, we have identified a candidate component. Recall that for each vertical line, there is at most one candidate component. If during the merge step no candidate component is identified, we check if the candidate component  $c_i$  for  $\ell_i$  is still a candidate component for  $\ell_{i+1}$  by checking if  $\ell_{i+1}$  intersects  $c_i$  using the  $x$ -coordinate of a rightmost point

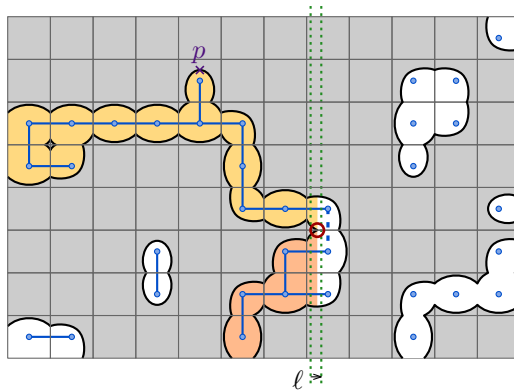


Figure 5.17: A free-space diagram. The starting elements are represented by blue disks. Elements left of the line  $\ell$  are connected, meaning their respective components are merged. When  $\ell$  moves over the point marked in red, the orange and yellow components are merged; this creates a component that touches the left and bottom boundaries and has  $p$  as topmost point.

of  $c_i$ . If a candidate component is identified, the  $y$ -coordinate of a topmost point of  $c_{i+1}$  is recorded as  $y_{i+1}^{\max}$ .

We repeat this step until we have iterated through all interesting vertical lines. It takes  $O(n^2 \alpha(n))$  time as there are  $O(n^2)$  interesting lines and we merge two components  $O(n^2)$  times. This identifies all candidate components for the interesting vertical lines and identifies the maximum height of those components.

**Computing  $y^{\min}$ .** For each maximal component  $c$  that contains candidate components, we walk along its bottom boundary to determine  $y_i^{\min}$  for the corresponding lines  $\ell_i$ , that is, for the  $\ell_i$  for which a subcomponent of  $c$  is a candidate component, see Figure 5.18. Let  $L_c$  be the set of all those lines  $\ell_i$ .

We start at the bottommost point  $p_c$  of  $c$  that is on the left boundary. We walk counterclockwise along the boundary of  $c$  and record the maximal  $y$ -coordinate encountered as  $\hat{y}$ . When we encounter a line  $\ell_i \in L_c$ , we set  $y_i^{\min}$  as the current value of  $\hat{y}$ . A horizontal line  $h$  with height at least  $y_i^{\min}$  and at most  $y_i^{\max}$  is a candidate line for  $\ell_i$ , as the subcomponent of  $c$  containing  $p_c$  left of  $\ell_i$  and below  $h$  intersects  $\ell_i$ ,  $h$ , and the left and bottom boundaries. Any horizontal line  $h$  with height less than  $y_i^{\min}$  intersects the boundary of  $c$  that we walked along and thus, the part of  $c$  below  $h$  does not contain a component that intersects  $\ell_i$  and the left boundary. The computation of the values  $y_i^{\min}$  for all  $i$  takes  $O(n^2)$  time as  $O(n^2)$  is the complexity of the components in the free space and as there are at most  $O(n^2)$  interesting vertical lines.

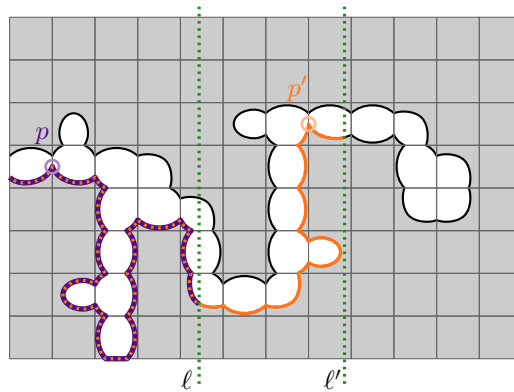


Figure 5.18: A free-space diagram. We walk along the bottom boundary of a component until encountering an interesting vertical line while recording the topmost point. For line  $\ell$ ,  $p$  is the topmost encountered point. For  $\ell'$ ,  $p'$  is the topmost encountered point.

**Putting it all together.** For each interesting vertical line  $\ell_i$ , we have determined if there is a candidate component and if there is, we have identified the interval  $I_i^{\swarrow} = [y_i^{\min}, y_i^{\max}]$ . We now call these components *bottom left candidate components*. We repeat the computation to determine for each interesting vertical line  $\ell_i$ , if there is a *top right candidate component* and if there is, we identify the interval  $I_i^{\nearrow}$ . As a final step, we iterate over all interesting vertical lines  $\ell_i$  and check if there is both a bottom left and a top right candidate component. If that is the case we check if  $I_i^{\swarrow} \cap I_i^{\nearrow} \neq \emptyset$ . In that case any horizontal line  $h \equiv y = y'$  with  $y' \in I_i^{\swarrow} \cap I_i^{\nearrow}$  leads to a valid cut together with  $\ell_i$ . Hence we conclude:

**Theorem 5.9.** For a value  $\varepsilon$  and two polygonal curves  $P$  and  $Q$  of complexity  $n$ , we can decide whether the cut distance  $d_{\text{cut}}(2, P, Q)$  is at most  $\varepsilon$  in  $O(n^2 \log n)$  time.

## 5.6 Conclusion

We presented two novel variants of the Fréchet distance for polygonal chains that allows to compare objects of rearranged pieces. We ask for  $k$  (possibly overlapping) subcurves per input curve that have pairwise small weak Fréchet distance. Thus, the  $k$ -Fréchet distance provides a transition between weak Fréchet and Hausdorff distance.

Computing any variant of the  $k$ -Fréchet distance of two polygonal curves is NP-hard. However, we were able to tackle the computational challenge from different angles: we give an XP-algorithm depending on  $k$  and approximate  $k$  by factor 2



---

for the cover distance. We also present a polynomial time algorithm for  $k = 2$  that decides whether the cut distance between two curves is below a certain value. For general  $k$ , computing the cut distance is NP-hard, and approximating the number of cuts  $k$  is APX-hard. For  $k \geq 3$ , we conjecture that cuts may have to be placed at non-interesting points, which is an indication of hardness, see Figure 5.16.



## Chapter 6

# Diverse Partitions of Colored Points

Imagine that a set of objects is represented by points in space and that different types or classes of objects are represented by colors. We study the algorithmic problem of creating convex or Voronoi partitions of space with maximally diverse cells, using two classic diversity measures: the *richness* (number of different colors) and the *Shannon index*. The diversity of a partition is the sum of the diversity scores of its cells. Hence, we wish to compute either a *diverse convex partition* (DCP) or a *diverse Voronoi partition* (DVP), which maximises the diversity score of the partition. Surprisingly, computing a DVP is NP-hard already in 1D and for only four colors, while DCP can easily be computed with dynamic programming. We show that DVP can be solved in polynomial time in 1D if a discrete set of candidate positions for the Voronoi sites is part of the input. These results apply to both the richness and the Shannon index. For richness, we also present a polynomial-time algorithm to compute a Voronoi partition whose diversity is at least  $1 - \varepsilon$  times the optimal diversity. In 2D, we show that both DCP and DVP are NP-hard, for richness as diversity measure. The reductions use constantly many colors for DVP and polynomially many colors for DCP. This chapter is based on the following publication:

M. van Kreveld, B. Speckmann and J. Urhausen. Diverse Partitions of Colored Points. *17th Algorithms and Data Structures Symposium (WADS)*, pages 641–654, 2021.

### 6.1 Introduction

Imagine that a data set consists of objects that have different types, or classes, like genre of a book or species of a tree. As an abstraction, we represent such different

types by different colors, and we are interested in *diverse* subsets. There are many different ways to partition objects, and also many different ways to define the diversity of a partition. In this chapter we study a fundamental *geometric* variant, namely the diversity of groups of colored points that are induced by general convex partitions of space, or those induced by a Voronoi diagram.

Two common ways to define the diversity of a set are the (*species*) *richness* and the *Shannon index*. The richness is the number of different colors in the set, while the Shannon index is defined as  $-\sum_{i=1}^h \rho_i \log_2 \rho_i$ , where  $h$  is the number of colors and  $\rho_i$  is the proportion of objects of color  $i$  in the set. For example, the Shannon index of the set {red, green, blue} is  $-(\frac{1}{3} \log_2 \frac{1}{3} + \frac{1}{3} \log_2 \frac{1}{3} + \frac{1}{3} \log_2 \frac{1}{3}) = \log_2 3 \approx 1.585$ , whereas the Shannon index of {red, green, blue, blue} is  $-(\frac{1}{4} \log_2 \frac{1}{4} + \frac{1}{4} \log_2 \frac{1}{4} + \frac{1}{2} \log_2 \frac{1}{2}) = 1.5$ . Hence the first set is more diverse. When we have a partition of space and the objects are points, we can view the points in each cell as a set, and hence we can define the *diversity score* of a cell. The diversity score of the partition is the sum of the diversity scores of its cells.

Besides general convex partitions, a meaningful subclass of convex partitions in this context are *Voronoi partitions*, that is, partitions of space which are induced by the Voronoi diagram of a set of *sites*. The Voronoi site can serve as the representative of the points contained in its cell. Conversely, each point is represented by the site closest to it. Using richness, a diverse site represents many colors; using the Shannon index, a diverse site also represents many colors, which are additionally present in roughly equal proportions. Intuitively the Shannon index of a region increases if we add a point with a new color or if we equalize the proportions of the existing colors.

**Formal problem statement.** Our input is a set  $P$  of  $n$  points in  $h$  different colors and a number  $k \in \mathbb{N}$ . For any partition into  $k$  cells, the diversity  $d_i$  in a cell is the *score* of that cell, and  $\sum d_i$  is the *score* of the partition. Our goal is to compute either a *diverse convex partition (DCP)* or a *diverse Voronoi partition (DVP)* with  $k$  cells which maximises the overall diversity score according to the richness measure or the Shannon index. Since  $k$  is given, maximising the total diversity and average diversity is equivalent. In the case of diverse Voronoi partitions, the problem is to find a set  $S = \{s_1, \dots, s_k\}$  of  $k$  sites such that the sum of the diversity scores over all Voronoi cells is maximised. See Figure 6.1 for an example of a convex partition (left) and a Voronoi partition (right, white disks represent Voronoi sites).

**Results and organization.** We study diverse convex partitions (DCP) and diverse Voronoi partitions (DVP) both on the line (1D) and in the plane (2D). We begin by surveying related research on diversity and on partitioning problems for colored points. In Section 6.2 we illustrate how convex and Voronoi partitions differ in 1D and also show how to test if a given convex partition can be realized by a Voronoi partition. It is straightforward to compute a DCP in 1D using dynamic programming. Surprisingly, Section 6.3 shows that computing a DVP is NP-hard already in 1D

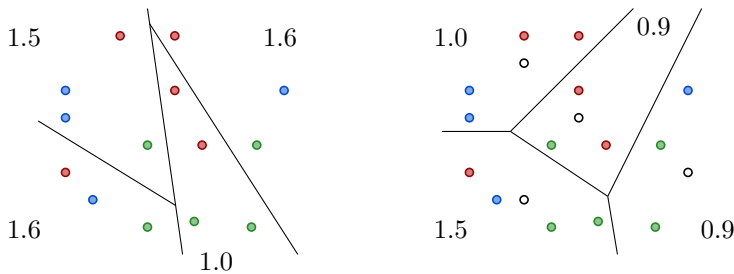


Figure 6.1: A colored point set with a convex partition (left, diversity score 11 for richness, 5.7 for Shannon index) and a Voronoi partition (right, diversity score 9 for richness, 4.3 for Shannon index; white disks denote sites). Each cell is annotated with its (rounded) Shannon index.

and for only four colors. This result holds for both the richness and the Shannon index. For richness, the NP-hardness can be extended to 2D using 12 colors. We also show that a DVP can be computed in polynomial time in 1D if a discrete set of  $m$  candidate positions for the Voronoi sites is part of the input. In Section 6.5 we show how to compute—in polynomial time for any constant  $\varepsilon > 0$ —a 1D Voronoi partition whose richness diversity is at least  $1 - \varepsilon$  times the richness of the DVP. Finally, in Section 6.6 we show that computing a DCP is NP-hard in 2D for richness. We reduce from MAXIMUM INDEPENDENT SET IN ORTHOGONAL LINE SEGMENTS using a colored grid structure which allows us to limit the possible shapes of convex sets.

### 6.1.1 Related Work

Diversity as a scientific concept is used to characterize sets; it is related to entropy, variety and representation [54]. Diversity plays an important role, for example, in ecology ([99, 117]; diversity of species), information retrieval ([25, 118]; diversification of query results), evolutionary algorithms ([91]; diversity in the population), and machine learning ([114]; diversity of classifiers). There are many measures of diversity, including species richness, the Shannon index, the Simpson index, and Rényi entropy. All these measures relate to sets of objects in classes, and have no spatial aspect. In a seminal paper, Whittaker [117] argued the need for differentiation in local (small-scale) diversity and regional (larger-scale) diversity. This requires a partition of a larger region into several smaller regions, and subsequently the analysis of diversity in the larger region and all of the smaller regions. The combined diversity measure is referred to as  $\beta$ -diversity; there are multiple different definitions in use [115]. The partitions we compute maximize the average or sum of local diversities over arguably reasonable geometric partitions.

The measure richness leads to algorithmic problems with a classic combinatorial and geometric structure. The Shannon index, however, gives rise to new challenges,

which can be observed in the NP-hardness proof for DVP in 1D, and the fact that the approximation algorithm does not easily generalize. For richness, a DCP has at most  $n$  different summed diversity scores; for the Shannon index this is exponential. Furthermore, the richness of a cell cannot decrease if we grow the cell and collect more points; the Shannon index can decrease when extra points make the distribution of colors less balanced.

Partitioning problems for sets of points are extensively studied in computational geometry and related areas. There is a variety of specific problem formulations and, correspondingly, a multitude of related work. Here we highlight some results on bi-colored and multi-colored point sets and convex partitions.

Kaneko and Kano [83] present a survey of results for red and blue points in the plane. This includes computing convex partitions of the plane, such that each cell contains  $a$  red and  $b$  blue points, or, alternatively, a specific ratio. Bespamyatnikh et al. [23] study equitable partitions of red-blue point sets using convex sets. This line of inquiry has been further extended by Bereg et al. [15, 17], Chierichetti et al. [43], and Holmsen et al. [78]. Bereg et al. [16] define coarseness of bicolored point sets as a measure of how mixed the two colors are. They give efficient algorithms for 2-partitions in the plane, and for point sets in convex position. Dumitrescu and Pach [55] partition multi-colored point sets into uni-colored subsets, whose convex hulls are disjoint; this minimizes diversity. Majumder et al. [100] consider the same problem, however, they partition the multi-colored input with axis-parallel lines. In  $d$  dimensions, Blagojević et al. [24] show that  $dn$   $d$ -colored points can always be partitioned into  $n$  sets with disjoint convex hulls and evenly distributed colors, thus maximizing diversity.

Diverse Voronoi Partition is in some sense a clustering problem reminiscent of  $k$ -means clustering, which aims to represent multiple points by a single point, following a nearest neighbor rule. While  $k$ -means clustering minimizes the sum of squared distances to nearest representatives, DVP aims to maximize the diversity of colors for each representative. DVP also bears resemblance to multi-criteria facility location, where sites need to be placed to optimize two or more often conflicting criteria (see the extensive survey by Farahani et al. [58]). In the case of DVP these criteria are distance and diversity.

## 6.2 Convex versus Voronoi Partitions in 1D

In this section we explore the difference between convex and Voronoi partitions in 1D. Consider the example in Figure 6.2 of 15 colored points, 5 in each of  $h = 3$  colors. It is easy to see that there is a convex partition with a perfect richness score of 15 using  $k = 5$  cells (intervals). To achieve the same score with a Voronoi partition, we need to place 5 sites such that the induced *boundaries* between Voronoi cells lie between the same input points as the corresponding boundaries of the convex partition. We capture this restriction on the Voronoi partition using so-called *b-intervals*. A *b-interval*

is the open interval between two consecutive input points. A Voronoi partition that realizes a richness score of 15 must place 5 sites in such a way that each boundary between Voronoi cells lies in the corresponding  $b$ -interval. A careful inspection shows that this cannot be done. The middle site  $s_3$  must be sufficiently far to the left to ensure that the second boundary is correctly placed (between the second green and the third red point), and at the same time sufficiently far to the right to ensure that the third boundary is correctly placed (between the third green and the fourth red point). It is impossible to move the other sites  $s_1, s_2, s_4,$  and  $s_5$  to realize this.

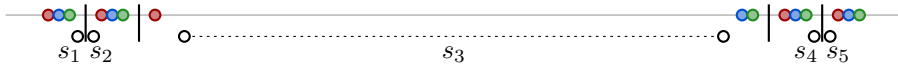


Figure 6.2: Points that admit a perfect convex partition but not a perfect Voronoi partition.

**Testing realizability for Voronoi partitions.** Using  $b$ -intervals it is straightforward to test—using linear programming—if a given convex partition can be realized as a Voronoi partition (see also [74]). The convex partition directly induces the  $b$ -intervals. Let  $b_i$  be the midpoint of the interval between the sites  $s_i$  and  $s_{i+1}$ , for  $1 \leq i < k$ . Then it must hold that  $s_1 \leq s_2 \leq \dots \leq s_k$ , and  $b_i = (s_i + s_{i+1})/2$ . To ensure that the Voronoi cell boundaries  $b_i$  lie inside their respective  $b$ -intervals, we use another  $2k - 2$  linear inequalities. Altogether, we have a system of  $5k - 5$  linear inequalities whose solution—if it exists—gives a Voronoi partition.

**Perfect partitions.** If the input  $S$  consists of exactly  $n/h$  points per color, we can ask if there is a *perfect partition* using  $k = n/h$  sites that together have a richness score of  $n$ . The unique perfect convex partition, if it exists, can be found in  $O(n)$  time if the points are given in sorted order. Constructing the corresponding system of linear inequalities also takes  $O(n)$  time. Solving this linear program, and hence testing for a perfect Voronoi partition, takes polynomial time in  $k = n/h$ .

## 6.3 NP-Hardness Proofs for the Voronoi Partition

### 6.3.1 Richness as the Diversity Measure in 1D

We prove that the decision version of DVP (D-DVP) is NP-complete, even in 1D. D-DVP has an extra parameter  $z$  and asks if a diversity score of at least  $z$  can be realized with a Voronoi partition using  $k$  points. We first argue containment in NP. For a given instance of D-DVP, there are only exponentially many partitions into subsets, each defined by  $k - 1$   $b$ -intervals. We can test for each of these partitions

if they can be realised as a Voronoi partition with  $k$  sites in polynomial time using linear programming (see Section 6.2).

For hardness we reduce from SUBSET SUM: for a set  $A = \{a_1, \dots, a_r\}$  of  $r$  integers and an integer  $b$ , is there a subset  $A' \subseteq A$  such that  $\sum_{a_j \in A'} a_j = b$ ? We first define a few terms. A point  $p \in P$  is *represented* by a site  $s \in S$  if  $s$  is the site closest to  $p$ . For each color  $i \in \{1, \dots, h\}$ ,  $P_i$  is the subset of points of color  $i$ , and a point  $p \in P_i$  is *scored* if each other point  $p' \in P_i$  that is represented by the same site is to the right of  $p$ . That is, for each site only the leftmost point of each color that it represents is scored. A point is *unscored* if it is not scored. Hence, an optimal set  $S$  of sites maximises the number of scored points. Our reduction uses only four colors, so we define point sets  $P_1, P_2, P_3$ , and  $P_4$  from an instance of SUBSET SUM with total size  $n = 8r + 14$ .

**The construction.** Let  $0 < \delta \ll 1$  be a small real and let  $0 < \varepsilon \ll \delta$  be an even smaller real. We can take  $\delta = 1/r^2$  and  $\varepsilon = 1/r^4$ . Later we can multiply the coordinates of the constructed points by  $r^4$  and thus obtain a set of integer positions with polynomial size. Let  $a = \sum_{i=1}^r a_i$  be the total sum of the integers of the SUBSET SUM instance.

We construct the set  $P$  using the values  $a_i$  and  $b$ . The goal is that the new D-DVP instance has a solution if and only if the SUBSET SUM instance has a solution. We describe  $P$  from left to right. First, there is a starting gadget  $H$  of six points. Then we have a gadget  $D^j$  for each  $a_j$ , consisting of eight points (these gadgets can be in any order). Next, we have a subset sum gadget  $E$  of two points to represent  $b$ , and finally we have an ending gadget  $G$  of six points.  $P = H \cup D^1 \cup \dots \cup D^r \cup E \cup G$ . Figure 6.3 shows an example for  $A = \{1, 2\}$  and  $b = 2$ , so  $P = H \cup D^1 \cup D^2 \cup E \cup G$ . Intuitively, each cell in a solution of the D-DVP instance contains exactly one (blue) point from  $P_1$  and one (green) point from  $P_2$ . In each gadget  $D^j$  there is a choice to either separate two (red) points from  $P_3$  or two (yellow) points from  $P_4$ . This choice corresponds to not choosing or choosing  $a_j$  in the subset sum so far.

To start the construction we define a set  $H$  of six points in two colors. We set  $H_1 = \{-2\delta, -\delta, 0\} \subset P_1$  and  $H_2 = \{-2\delta - \varepsilon, -\delta - \varepsilon, -\varepsilon\} \subset P_2$ . The set  $H$  thus forms three groups of two points of different colors. We can only score all points in  $H$  with three sites  $s_{-2}, s_{-1}, s_0$  if we have  $-\delta < s_0 < 2\delta - 2\varepsilon$ . So, in order to score all six points with three sites, the rightmost of those sites,  $s_0$ , needs to be close to zero.

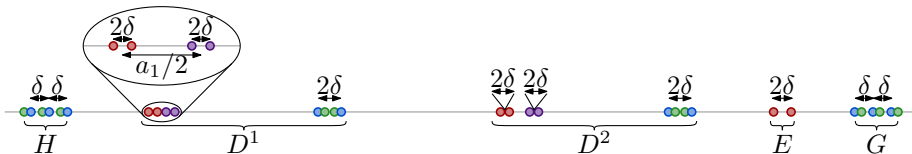


Figure 6.3: Reducing SUBSET SUM to D-DVP for  $a_1 = 1, a_2 = 2$  and  $b = 2$  using  $P_1$  (blue),  $P_2$  (green),  $P_3$  (red), and  $P_4$  (yellow). Touching points are at  $\varepsilon$  distance,  $\delta$  is not drawn to scale.



For each  $a_j \in A$  we create a set  $D^j$  of eight points that encodes whether  $a_j$  is chosen in the subset  $A'$  or not. Let  $D^j = D_1^j \cup D_2^j \cup D_3^j \cup D_4^j$ , with  $D_i^j \subset P_i$  (the points in  $D_i^j$  have color  $i$ ). Let  $D_1^j = \{(4j-1)a - \delta, (4j-1)a + \delta\}$ ,  $D_2^j = \{(4j-1)a - \delta + \varepsilon, (4j-1)a + \delta - \varepsilon\}$ ,  $D_3^j = \{(4j-3)a - \delta, (4j-3)a + \delta\}$  and  $D_4^j = \{(4j-3)a + a_j/2 - \delta, (4j-3)a + a_j/2 + \delta\}$ . The distances between  $D_3^j$  and  $D_4^j$  are roughly  $a_j/2$ .

We define a set  $E \subset P$  that encodes that we want the subset sum to be  $b$ . We define  $E \subset P_3$  with  $E = \{4ra + (a-b)/2 - \delta, 4ra + (a-b)/2 + \delta\}$ .

Finally, we define a set  $G$  of six points, similar to  $H$ . It can only be scored fully by three sites if the leftmost of the sites is close to  $(4r+1)a$ . We set  $G_1 = \{(4r+1)a, (4r+1)a + \delta, (4r+1)a + 2\delta\} \subset P_1$  and  $G_2 = \{(4r+1)a + \varepsilon, (4r+1)a + \delta + \varepsilon, (4r+1)a + 2\delta + \varepsilon\} \subset P_2$ .

The instance of D-DVP has  $n = 8r + 14$  points and asks to place  $k = 2r + 6$  sites to realize a score of  $z = 7r + 14$ , which can be achieved if and only if the corresponding SUBSET SUM instance has a solution.

**Equivalence.** We lead with the intuition of the proof. We want to use the sites to create boundaries that separate the first three pairs of points, the last three pairs of points, either  $D_3^j$  or  $D_4^j$ , and also  $E$ . Separating  $D_3^j$  corresponds to not choosing  $a_j$  in a subset and separating  $D_4^j$  corresponds to choosing  $a_j$ . If we choose the correct  $a_j$ , the boundary between the last site chosen for  $D^r$  and the first site chosen for  $G$  “magically” separates the points in  $E$ . Then, only one point of each  $D^j$  is not scored. See Figure 6.4 for an example. The proof of correctness argues that essentially there are no other options: the SUBSET SUM instance has a solution if and only if the D-DVP instance can score  $7r + 14$ .

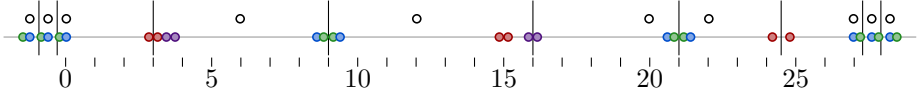


Figure 6.4: D-DVP instance:  $a_1 = 1$ ,  $a_2 = 2$  and  $b = 2$ . Sites and boundaries for a score of  $7r + 14$ .

Let  $A'$  be the subset with  $\sum_{a_j \in A'} a_j = b$ . We define  $b_j = \sum_{a_i \in A', i \leq j} a_i$  as the subset sum up until  $j$ . We set  $S = S_H \cup S_G \cup \bigcup_{j=1}^r S_j$ , where  $S_H = H_1 = \{-2\delta, -\delta, 0\}$ ,  $S_G = G_1 = \{(4r+1)a, (4r+1)a + \delta, (4r+1)a + 2\delta\}$  and for  $j \in \{1, \dots, r\}$ ,  $S_j = \{(4j-2)a + b_j, 4ja - b_j\}$ . Only one point per  $D^j$  is not scored: if  $a_j \in A'$ , the right point in  $D_3^j$  is not scored, else, the right point in  $D_4^j$  is not scored. Both points in  $E$  are scored because the boundary between the last site from  $S_r$  and the first site from  $S_G$  lands precisely between the points of  $E$ , because  $b_r = \sum_{a_i \in A', i \leq r} a_i = b$ . This brings the total score to  $7r + 14$  as shown in Figure 6.4.

Conversely, we want to prove that if there is a set  $S$  with  $|S| = k$  and score  $7r + 14$ , then the original SUBSET SUM instance has a solution. Let us now assume that

we have a set of  $2r + 5$  boundaries  $B$  subdividing  $P$  into a set of cells such that it maximizes the number of points of  $P$  scored. As above  $p \in P$  is considered scored if it is the leftmost point of that color within its cell. Let  $\mathcal{I}$  be the set of b-intervals  $\{(-2\delta, -\delta - \varepsilon), (-\delta, -\varepsilon), ((4r+1)a + \varepsilon, (4r+1)a + \delta), ((4r+1)a + \delta + \varepsilon, (4r+1)a + 2\delta)\} \cup \{(\min D_2^j, \max D_2^j) \mid j \in \{1, \dots, r\}\}$ . For  $j \in \{1, \dots, r\}$ , let further  $J_j$  and  $L_j$  be the b-intervals  $(\min D_3^j, \max D_3^j)$  and  $(\min D_4^j, \max D_4^j)$ . We have:

**Lemma 6.1.** *For boundaries  $B$  with score  $7r + 14$ , there is exactly one boundary in each b-interval of  $\mathcal{I}$  and, for each  $j \in \{1, \dots, r\}$ , there is exactly one boundary in  $J_j$  or  $L_j$ .*

*Proof.* We define intervals relating to the gadgets: let the segment  $I_1$  be  $(\max H, \max D^1]$  and for  $j \in \{2, \dots, r\}$  let the segment  $I_j$  be  $(\max D^{j-1}, \max D^j]$ . Furthermore we define the start segment  $I_s = (\min H, \max H]$  and the end segment  $I_e = (\max D^r, \max G]$ . The segments partition the interval  $(\min P, \max P]$  and each segment ends with a point of  $P_1$ , with the exception of the last segment  $I_e$ .

We count the *unscored* points, that is, points that are not scored. For a total score of  $7r + 14$  only  $r$  points can be unscored.

We have the following cases for the start segment  $I_s$ : without boundaries at most two points are scored (because they are the leftmost of their color), so at least four points are not scored. For one boundary, at least two points are not scored and for two boundaries one can score all points.

The end segment  $I_e$  contains the gadgets  $E$  and  $G$ . Without boundaries within  $I_e$  at most the leftmost point in  $E$  and the leftmost point of color two can be scored, leaving six points. For one boundary at least four points are not scored and for two boundaries at least two points are unscored. Only for three boundaries all can be scored.

For  $j \in \{1, \dots, r\}$ , we have the following case distinction depending on the boundaries within the segment  $I_j$ : without boundaries at least five points are not scored. For one boundary three points remain unscored, with two boundaries at least one remains unscored and only for three boundaries all can be scored.

We know that by placing two boundaries in  $I_s$ , three boundaries in  $I_e$  and two boundaries in each segment  $I_j$ , there are at least  $r$  points that are not scored. Only the segments  $I_j$  can score more. However if we remove one boundary from a segment  $I_s, I_e$  or  $I_i$  ( $i \neq j$ ) to add to  $I_j$ , then at least two more points are unscored for each boundary removed and at most one point more is scored for each boundary added. So moving boundaries to different segments only reduces the overall score. This means that the above assignment of boundaries to segments is optimal.  $\square$

Note that the proof of Lemma 6.1 also implies that  $7r + 14$  is the maximum possible score. So overall if the answer of the D-DVP instance is "Yes", we know that there is set of sites with a score of  $7r + 14$  for this instance. By Lemma 6.1, we know the position of the boundaries. To be precise we know which b-intervals these boundaries lie in. We define the set  $A'$  as the set of all  $a_i$  for which there is a boundary in the

b-interval  $L_i$ . We define  $b_j = \sum_{a_i \in A', i \leq j} a_i$ . For  $x, y \in \mathbb{R}$ , let  $x \pm y$  be the interval of width  $2y$  centered around  $x$ .

We know that the first three sites  $s_{-2}$ ,  $s_{-1}$  and  $s_0$  are roughly at positions  $-2\delta$ ,  $-\delta$  and  $0$ . To be more exact, we have  $-\delta < s_0 < 2\delta - 2\varepsilon$  and thus  $s_0 \in 0 \pm 2\delta$ . Each b-interval  $J_i$ ,  $L_i$  and each b-interval in  $I$  has width at most  $2\delta$ . Thus, knowing the positions of the boundaries, we get  $s_{2i-1} \in (4i-2)a + b_i \pm 4i\delta$  and  $s_{2i} \in 4ia - b_i \pm (4i+2)\delta$ . As  $G$  is symmetric to  $H$ , we have  $s_{2r+1} \in (4r+1)a \pm 2\delta$ . Using that value, we can calculate the position of the boundary  $b_{2r}$ :

$$\begin{aligned} b_{2r} &= (s_{2r} + s_{2r+1})/2 \\ &\in (2ra - b_r/2) + ((2r+1/2)a) \pm (2r+2)\delta = 4ra + (a - b_r)/2 \pm (2r+2)\delta. \end{aligned}$$

As we have  $\delta \ll 1/r$  and  $4ra + (a - b)/2 - \delta = \min E < b_{2r} < \max E = 4ra + (a - b)/2 + \delta$ , we get:  $4ra + (a - b_r)/2 = 4ra + (a - b)/2$  and thus  $b_r = b$ , which shows that the SUBSET SUM instance has a solution.

**Theorem 6.2.** *Deciding if a diverse Voronoi partition of  $n$  colored points in four colors in 1D of richness diversity at least  $z$  exists, using  $k$  cells, is NP-complete.*

### 6.3.2 Richness as the Diversity Measure in 2D

The NP-hardness of DVP in 1D does not immediately extend to 2D by embedding the 1D construction in 2D on a line, as it is known that a line can be subdivided into arbitrary convex cells using a 2D Voronoi diagram. Hence, the construction of points in  $P$  must use 2D as well. The main idea of the proof is to use the 1D construction on three parallel lines, each with a different set of four colors. The construction can be seen in Figure 6.5. Intuitively, we ensure that (nearly) all sites are horizontally (nearly) aligned, so that all relevant bisectors are (nearly) vertical Voronoi edges. However, additional care is needed to ensure that the sites do not stray too far away from the constructed instance.

**Construction.** We reduce again from SUBSET SUM. Let  $A = \{a_1, \dots, a_r\}$ ,  $b$  be a SUBSET SUM instance with positive integers, and let  $P = \bigcup_{i=1}^4 P_i = H \cup E \cup G \cup \bigcup_{i=1}^r D^i$  be the 1D DVP instance constructed in Section 6.3.1. Let  $0 < \varepsilon \ll \delta \ll 1$ , for example  $\delta = 1/(n^2 a^2)$  and  $\varepsilon = \delta^2$  with  $a = \sum a_i$ . For this proof we extend the start gadget with more points to become  $H_1 = \{-4\delta, -3\delta, -2\delta, -\delta, 0\} \subset P_1$  and  $H_2 = \{-4\delta - \varepsilon, -3\delta - \varepsilon, -2\delta - \varepsilon, -\delta - \varepsilon, -\varepsilon\} \subset P_2$ . We make three copies  $P^0$ ,  $P^1$  and  $P^2$  of this new set. We give each set its own four colors, so we use twelve colors overall. We place  $P^0$  on the line  $y = 0$ ,  $P^1$  on the line  $y = 1$  and  $P^2$  on the line  $y = 2$ . As a last step, we move the five leftmost points of  $P^i$  left by  $i$ , see Figure 6.5.



Figure 6.5: The set of points of twelve colors constructed for  $a_1 = 1$ ,  $a_2 = 2$  and  $b = 2$ . The start gadget (on the left) ensures that there is a site at roughly  $(0, 0)$ .

**Equivalence.** We prove that using  $2r + 8$  cells, the maximum score for DVP on the  $n = 24r + 54$  points is  $21r + 54$  if and only if the SUBSET SUM instance has a solution. First, assume that the SUBSET SUM instance has a solution, that is, there exists  $A' \subseteq A$  with  $\sum_{a_i \in A'} a_i = b$ . As before, we define  $b_j = \sum_{a_i \in A', i \leq j} a_i$  as the sum up until  $j$ . We set  $S = S_H \cup S_G \cup \bigcup_{j=1}^r S_j$  as the set of sites, where  $S_H = \{(-3\delta, -\delta), (-5\delta/2, -\delta/2)\} \cup \{(-i\delta, 0) \mid i \in \{0, 1, 2\}\}$ ,  $S_G = \{(4r + 1)a + i\delta, 0) \mid i \in \{0, 1, 2\}\}$ . In  $P^0$ , only one point per  $D^j$  is not scored: if  $a_j \in A'$ , the right point in  $D_3^j$  is not scored, else, the right point in  $D_4^j$  is not scored. Both points in  $E$  are scored because the boundary between the last site from  $S_r$  and the first site from  $S_G$  lands precisely between the points of  $E$ , because  $b_r = \sum_{a_i \in A', i \leq r} a_i = b$ . The same is true for  $P^1$  and  $P^2$ . This brings the score to  $7r + 18$  for each of  $P^0, P^1$  and  $P^2$ , for a total of  $21r + 54$ .

Conversely, assume there is a set of  $2r + 8$  cells  $\mathcal{C}$  with a score of at least  $21r + 54$ . The line  $y = 0$  is subdivided by  $\mathcal{C}$  into at most  $2r + 8$  convex cells. The result of Lemma 6.1 still holds here, as the additional four points at the start of the instance do not influence the proof. As a result, using  $2r + 8$  cells, there are at least  $r$  points of  $P^0$  that are not scored. The same holds for  $P^1$  and  $P^2$ . As a score of  $21r + 54$  means that exactly  $3r$  points are not scored, and as the sets  $P^i$  use different colors,  $\mathcal{C}$  subdivides each of the three sets into  $2r + 8$  convex cells. As a consequence, the intersection of the Voronoi diagram and the strip between the lines  $y = 0$  and  $y = 2$  is a set of  $2r + 7$  segments with one endpoint at  $y = 0$  and one at  $y = 2$ . We call those segments *boundary segments*. Lemma 6.1 further dictates the location of the boundaries. There is a boundary between adjacent pairs of blue and green points and for each gadget  $D^j$  there is a boundary either between the two red or the two yellow points. Concerning the latter, if a boundary segment separates two yellow points of  $P^0$ , that segment must also intersect the yellow points of  $P^1$  and  $P^2$  as otherwise the boundary segment must be curved (for example, it is not possible that a straight boundary separates corresponding pairs yellow points in  $P^0$  and  $P^1$ , and a pair of red points in  $P^2$ ). The analogous statement holds if a boundary segment separates two red points of  $P^0$ . Thus each boundary segment, with the exception of the two leftmost ones, is almost vertical: the difference between the  $x$ -coordinates of their endpoints on the strip boundary is at most  $\delta$ . The two leftmost boundary segments have a slope of close to  $-1$ , that is, the difference between the  $x$ -coordinates of their endpoints is between  $2 - \delta$  and  $2 + \delta$ .

In the following we ignore any terms with  $\varepsilon$  and  $\delta^2$  in larger expressions as they are negligible compared to terms with  $\delta$  and constant terms. From left to right, we name

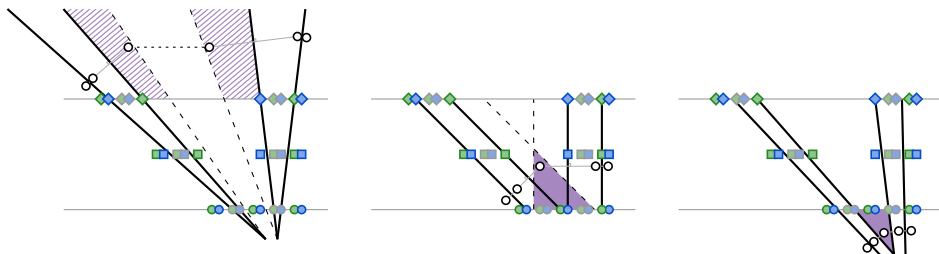


Figure 6.6: We consider the five leftmost sites  $s_{-4}$ ,  $s_{-3}$ ,  $s_{-2}$ ,  $s_{-1}$  and  $s_0$ . Left: the middle site  $s_{-2}$  can not be above  $y = 2$ . Middle: when  $s_{-2}$  is between  $y = 0$  and  $y = 2$ , it is in a small area close to  $(2\delta, 0)$ . Right: When  $s_{-2}$  is below  $y = 0$ , it is in a small triangle close to  $(2\delta, 0)$ .

the sites  $s_{-4}, \dots, s_{2r+3}$ . We consider the position of site  $s_{-2}$ , see Figure 6.6. We first assume for the sake of contradiction that  $s_{-2}$  is above the line  $y = 2$ . Then, as  $s_{-1}$  is located between two close, nearly vertical lines, the site  $s_{-2}$  is right of the line through  $(-\delta, 0)$  with slope  $-2/(3\delta)$ . As  $s_{-3}$  is located between two close almost diagonal lines, the site  $s_{-2}$  is left of the line through  $(-6d/(2-d), 0)$  with slope  $-(2+d)/(2-2d)$ . The intersection of the halfplanes induced by those lines does not contain points above  $y = 2$ ; thus  $s_{-2}$  is below that line. If  $s_{-2}$  is between the lines  $y = 0$  and  $y = 2$ , it must be right of the line  $x = -4\delta + \varepsilon$  and left of the line through  $(-2\varepsilon)$  with slope  $-1$ . Their intersection is the point  $(-4\delta + \varepsilon, 4\delta + 3\varepsilon)$ . If  $s_{-2}$  is below the line  $y = 0$ , it is left of the line through  $(-\delta - \varepsilon, 0)$  with slope  $-2/(\delta - \varepsilon)$  and right of the line through  $(-3\delta, 0)$  with slope  $-2/(2 - \delta + \varepsilon)$ . They intersect in  $(-\delta/(1 - \delta), -2\delta/(1 - \delta))$ . It can thus be concluded that  $s_{-2} \in (-2\delta, 0) \pm 4\delta$ , where  $(x, y) \pm z\delta = \{(x', y') \mid |x - x'| \leq z\delta \wedge |y - y'| \leq z\delta\}$ . This implies  $s_0 \in [-5\delta, +5\delta] \times [-5\delta, +5\delta]$ , that is, close to  $(0, 0)$ .

The almost vertical lines have a slope of at least  $2/\delta$  or at most  $-2/\delta$ , and hence the difference in the  $y$ -coordinate between adjacent sites  $s_i$  and  $s_{i+1}$ , with  $i \geq 0$  is at most  $a\delta$ . We define the set  $A' \subseteq A$  as the set of all  $a_j$  for which there is a boundary segment separating two adjacent yellow vertices in the gadget  $D_j$  and we define  $b_j = \sum_{a_i \in A', i \leq j} a_i$  for  $j \in \{1, \dots, r\}$ . Knowing the position of the boundary segments, we get  $s_{2i-1} \in ((4i-2)a + b_i, 0) \pm (2i-1)a\delta + 5\delta$  and  $s_{2i} \in (4ia - b_i, 0) \pm 2ia\delta + 5\delta$  for  $i \in \{1, \dots, r\}$ . Finally, we have  $s_{2r+1} \in ((4r+1)a + b_r - b, 0) \pm (2r+1)a\delta + 5\delta$ . The site  $s_{2r+1}$  lies in the strip between  $y = -1$  and  $y = 1$ . This implies that  $s_{2r+2}$  and  $s_{2r+3}$  do so too. This in return implies  $s_{2r+1} \in ((4r+1)a, 0) \pm (2r+1)a\delta + 5\delta$ . As  $\delta \ll 1/an$ , we have  $b_r = b$ , which shows that the SUBSET SUM instance has a solution.

**Theorem 6.3.** *Deciding if a diverse Voronoi partition of  $n$  colored points in twelve colors in 2D of richness diversity at least  $z$  exists, using  $k$  cells, is NP-complete.*

### 6.3.3 Shannon Index as the Diversity Measure in 1D

We prove that deciding DVP is NP-complete in 1D, also when using the Shannon index as diversity measure. For ease of argument, we allow points of different colors to share locations. For containment in NP, we note that we need logarithms of  $1..n$  only, and we can still guess the partition and approximate its total Shannon index sufficiently precisely. For NP-hardness, we reduce from SUBSET SUM as before: for a set  $A = \{a_1, \dots, a_r\}$  of integers and an integer  $b$ , is there a subset  $A' \subseteq A$  such that  $\sum_{a_j \in A'} a_j = b$ ? The set of points we construct is similar to the one using richness, but the proof arguments are more complex. We place all yellow, red, and blue points at exactly the same positions as before, and set  $\varepsilon = 0$  so that each green point coincides with the nearest blue point. We use two more red points in the start gadget and two more in the end gadget, to coincide with the first two blue points and the last two blue points, as shown in Figure 6.7.

The decision question corresponding to SUBSET SUM is: using  $k = 2r + 6$  sites, can we get a score of at least  $z = (\ell_3 + \ell_5)r + 6\ell_3$  in their Voronoi cells, where  $\ell_3 = \log_2(3) \approx 1,58$  is the score of a cell with three points of different colors and  $\ell_5 = \log_2(5) - 2/5 \approx 1,92$  is the score of a cell with two points of one color and three other points of different colors?



Figure 6.7: The set of points constructed for  $a_1 = 1$ ,  $a_2 = 2$  and  $b = 2$  to prove NP-completeness when using the Shannon index for diversity, corresponding to Figure 6.4.

**Equivalence.** First, given a solution to the SUBSET SUM instance, we construct a solution to the DVP problem in the same manner as for the richness in Subsection 6.3.1. Second, we assume we have a Voronoi subdivision of 1D with score  $(\ell_3 + \ell_5)r + 6\ell_3$  and prove that the corresponding SUBSET SUM instance has a solution in the following.

Table 6.1 gives the Shannon index for all possible cells in this instance with up to eight points, where  $[x_1, \dots, x_m]$  denotes a cell with  $m$  different colors and  $x_i$  points per color. Note that any cell, even with more points, has a score of at most 2, the maximum with four colors.

Similar to the proof for richness in Subsection 6.3.1, we first prove that with convex cells, the solution follows a certain pattern.

**Lemma 6.4.** *Using  $2r + 6$  convex cells, the maximum possible score for the constructed instance is  $(\ell_3 + \ell_5)r + 6\ell_3$ , and this can only be achieved with  $r + 6$  cells of the form  $[1, 1, 1]$  and  $r$  cells of the form  $[2, 1, 1, 1]$ .*

Table 6.1: The (rounded) Shannon index for distributions of up to eight points that occur in the construction.

cell	score	cell	score	cell	score
$[\emptyset]$	0	$[1, 1, 1]$	1.585	$[2, 1, 1, 1]$	1.922
$[1]$	0	$[2, 1, 1]$	1.5	$[2, 2, 1, 1]$	1.918
$[2]$	0	$[2, 2, 1]$	1.522	$[2, 2, 2, 1]$	1.950
$[1, 1]$	1	$[2, 2, 2]$	1.585	$[2, 2, 2, 2]$	2
$[2, 1]$	0.918	$[3, 3, 2]$	1.561	$[3, 2, 2, 1]$	1.906
$[2, 2]$	1	$[3, 3, 3]$	1.585		

*Proof.* Let  $A$  be the subdivision of the DVP instance into  $2r + 6$  convex cells that yields the maximum total Shannon index score. Let us count the number of cells of this optimal assignment using classes. Class  $C_0$  contains the cells that contain at most 1 color, and  $C_1$  ( $C_{\ell_3}$ ) contain the cells with points of precisely two (respectively, three) colors. The cells with precisely four colors appear in two classes:  $C_{\ell_5}$  and  $C_2$  contain the cells with four colors that contain exactly one, respectively two or more blue point(s). Let  $c_i = |C_i|$ . Note that a cell in a class  $C_i$  has Shannon index at most  $i$ . Each cell with three or more colors contains at least one blue point, since blue and green points coincide. As there are  $2r + 6$  cells and  $2r + 6$  blue points, the following inequalities hold:

$$\begin{aligned}
2r + 6 &= c_0 + c_1 + c_{\ell_3} + c_{\ell_5} + c_2 && \text{(count the cells)} \\
\implies 2r + 6 &\geq c_1 + c_{\ell_3} + c_{\ell_5} + c_2 && (1) \\
2r + 6 &\geq c_1 + c_{\ell_3} + c_{\ell_5} + 2c_2 && \text{(count blue points per cell)} \\
\implies 2r + 6 &\geq c_{\ell_3} + c_{\ell_5} + 2c_2 && (2)
\end{aligned}$$

$$\begin{aligned}
\text{Max Score} &\leq 0 \cdot c_0 + 1 \cdot c_1 + \ell_3 \cdot c_{\ell_3} + \ell_5 \cdot c_{\ell_5} + 2 \cdot c_2 \\
&\leq c_1 + \ell_3 c_{\ell_3} + \ell_5 c_{\ell_5} + 2c_2 \\
&\stackrel{(1)}{\leq} (2r + 6 - c_{\ell_3} - c_{\ell_5} - c_2) + \ell_3 c_{\ell_3} + \ell_5 c_{\ell_5} + 2c_2 \\
&= 2r + 6 + (\ell_3 - 1)c_{\ell_3} + (\ell_5 - 1)c_{\ell_5} + c_2 \\
&\stackrel{(2)}{\leq} 2r + 6 + (\ell_3 - 1)c_{\ell_3} + (\ell_5 - 1)c_{\ell_5} + (2r + 6 - c_{\ell_3} - c_{\ell_5})/2 \\
&= 3r + 9 + (\ell_3 - 3/2)c_{\ell_3} + (\ell_5 - 3/2)c_{\ell_5} && (3)
\end{aligned}$$

Since we have  $\ell_5 > \ell_3 > 3/2$  and  $c_{\ell_3} + c_{\ell_5} \leq 2r + 6$ , in order to upper-bound expression (3) we maximise  $c_{\ell_5}$  first and  $c_{\ell_3}$  second. Each cell in class  $C_{\ell_5}$  has points of four colors and exactly one blue point. No cell in  $C_{\ell_5}$  can contain a yellow point  $p$

and a red point  $q$  with  $p < q$ , otherwise the cell would contain two blue points. The yellow points appear in adjacent pairs and there are  $r$  such pairs. Thus  $c_{\ell_5} \leq r$  holds, and equality is attainable. Given  $c_{\ell_5} = r$ , we get  $c_{\ell_3} \leq r + 6$ , and also here equality is attainable in the construction. We have:

$$\begin{aligned} \text{Max Score} &\leq 3r + 9 + (\ell_3 - 3/2)c_{\ell_3} + (\ell_5 - 3/2)c_{\ell_5} \\ &\leq 3r + 9 + (\ell_3 - 3/2)(r + 6) + (\ell_5 - 3/2)r \\ &= (\ell_3 + \ell_5)r + 6\ell_3 \end{aligned}$$

This concludes the proof.  $\square$

This lemma implies that the Voronoi subdivision for the constructed DVP instance follows the exact same pattern as in Subsection 6.3.1. Using the same arguments we can conclude that then the initial SUBSET SUM instance has a solution. This concludes the proof and shows:

**Theorem 6.5.** *Deciding if a diverse Voronoi partition of  $n$  colored points in four colors in 1D of Shannon index at least  $z$  exists, using  $k$  cells, is NP-complete.*

## 6.4 Polynomial-Time Solution for Discrete Candidate Sites

If we assume that there is a fixed set of candidate positions for the sites, then optimal diverse Voronoi partitions can be computed in 1D by dynamic programming both for richness and for the Shannon index. We now assume that the  $k$  sites can be placed only at a finite set  $M$  of pre-specified positions. We work our way from left to right, using the fact that in a placement of the first  $i$  sites, we use an optimal placement of the first  $i - 1$  sites. However, the Voronoi boundaries between sites are determined by the last two sites, so our recurrence for an optimal solution has parameters for the last two sites. Furthermore, since we do not know the score for the  $i$ -th site (its right boundary is not yet fixed) we define the maximum total score (by richness or Shannon index) for the first  $i - 1$  sites for all possible candidate positions of the  $(i - 1)$ -th and  $i$ -th sites.

Consider  $f : \{2, \dots, k\} \times M \times M \rightarrow \mathbb{N}$ . For  $i \in \{2, \dots, k\}$  and  $u, v \in M$ , with  $u < v$ , let

$$f(i, u, v) = \max_{s_1, \dots, s_{i-2} \in M} \left\{ \sum_{j=1}^{i-1} d_j \mid s_1 < \dots < s_i, s_{i-1} = u, s_i = v \right\}$$

be the best possible score of the first  $i - 1$  cells while fixing  $s_{i-1} = u$  and  $s_i = v$ , where  $d_j$  is the diversity score of the cell represented by site  $s_j$ , by richness or Shannon



index. We further define the function  $g : (M \cup \{-\infty\}) \times M \times (M \cup \{+\infty\}) \rightarrow \mathbb{N}$ , where  $g(a, b, c)$  is the diversity score of the cell of site  $b$ , where  $a$  and  $c$  are its left and right neighboring sites. For  $a, b \in M$ , we have  $f(2, a, b) = g(-\infty, a, b)$ . Additionally we have:

$$\forall i \in \{3, \dots, k\}, \forall b, c \in M, \text{ with } b < c : f(i, b, c) = \max_{a \in M, a < b} \{f(i-1, a, b) + g(a, b, c)\}.$$

To find the best sites overall we determine  $\max_{a, b \in M, a < b} \{f(k, a, b) + g(a, b, +\infty)\}$ , which includes the diversity score of the  $k$ -th cell in the second term. The dynamic program uses a table of size  $O(km^2)$ , where  $m = |M|$ . Filling an entry requires optimizing over  $m$  choices. To evaluate the function  $g(a, b, c)$  one must first determine how many points of each color are present in the cell of site  $b$  and then evaluate either the richness or the Shannon index. Since one can trivially determine the colors in a cell in  $O(n)$  time, we immediately get an  $O(km^3n)$  time algorithm. The positions of the sites can then be obtained by backtracking.

We can avoid spending  $O(n)$  time for each of the  $O(km^3)$  evaluations of  $g$  by preprocessing: First for all  $a, b \in M$ , compute  $(a + b)/2$  and sort these values in  $O(m^2 \log m)$  time. Then we can scan through all those boundaries and all points in  $P$  simultaneously from left to right and record for each boundary how many points of each color are to the left of the boundary in  $O(hm^2 + n)$  time. Lastly for each triplet  $a, b, c \in M$  with  $a < b < c$ , we can determine how many points of each color are between the boundaries  $(a + b)/2$  and  $(b + c)/2$  by just computing the difference between the points left of the boundaries calculated in the previous step. This last step allows us to compute all  $g(a, b, c)$  in  $O(hm^3)$  time which leads to an overall time of  $O(hm^3 + n)$  for the precomputation.

To calculate the richness measure of a cell, we only need to know which colors are present and do not care about the number of points of that color. Therefore, for the richness measure, we can also preprocess as follows. Again we determine and sort all boundaries  $(a + b)/2$  in  $O(m^2 \log m)$  time. We scan this sequence this time from right to left, together with the points in  $P$ . We maintain the leftmost point of each color. When we encounter a potential Voronoi cell boundary  $(a + b)/2$ , we construct the sorted list of at most  $h$  differently colored points that are leftmost (but right of  $(a + b)/2$ ), and store it with this potential cell boundary. This scan takes  $O(m^2 h \log h + n)$  time. Then, for each  $a, b \in M$  with  $a < b$ , we scan the at most  $h$  differently colored points in its list from left to right, and simultaneously scan the potential boundaries  $(b + c)/2$  with  $b < c$ . Keep a count of the number of colored points encountered (with different colors by construction), and fill in the richness count in  $g(a, b, c)$  when  $(b + c)/2$  is encountered in  $O(1)$  time. These  $O(m^2)$  scans take  $O(h + m)$  time each. In total, preprocessing takes  $O(m^3 + m^2 h \log h + n)$  time in this case.

**Theorem 6.6.** *A diverse Voronoi partition of  $n$  points in  $h$  colors in sorted order in 1D into  $k$  cells using  $m$  discrete candidate positions can be computed in  $O(km^3 + hm^3 + n)$  time, for richness or Shannon index, or in  $O(km^3 + m^2h \log h + n)$  time for richness.*

## 6.5 Approximation for Diverse Voronoi Partition in 1D

In this section we show that for any constant  $\varepsilon > 0$ , we can compute a Voronoi partition whose diversity (richness) is at least  $1 - \varepsilon$  times the optimal diversity in polynomial time. We first use more sites than allowed in order to separate subproblems, which we solve optimally using linear programming. We combine the subsolutions using dynamic programming, and then remove sites to the desired number without deteriorating the solution too much.

Let  $P = \{p_1, \dots, p_n\}$ ,  $k$ , and  $0 < \varepsilon < 1$  be given. Let  $e = \lceil 2/\varepsilon \rceil$ , and let  $\delta$  be a small number, for example  $\min_{i=1, \dots, n-1} \{(p_{i+1} - p_i)/4\}$ . For  $i = 1, \dots, n-1$  we define  $q_i = (p_i + p_{i+1})/2$  as the middle between  $p_i$  and its right neighbor  $p_{i+1}$ .

Our goal is to subdivide  $P$  into  $g = \lceil k/e \rceil$  subsets and then place  $e$  sites optimally within each subset. For each of the  $\binom{n+1}{2}$  non-empty convex subsets of points, we calculate a specific score  $s(i, j)$  that is specified for a convex subset  $p_i, \dots, p_j$ , where  $2 \leq i < j \leq n-1$ , as follows: place two sites, one at  $q_{i-1} + \delta$  and one at  $q_j - \delta$ ; these are fixed. Then we place another  $e$  sites in between these two sites in an optimal way, maximizing the score. We do this by placing the  $e+1$  boundaries between the  $e+2$  sites, and then checking whether these boundaries are realizable by a Voronoi partition, using linear programming. There are  $O((j-i)^{e+1}) = O(n^{e+1})$  choices to be checked. We store the maximum in a table for  $s(i, j)$ . For the convex subsets  $p_1, \dots, p_j$  or  $p_i, \dots, p_n$  we compute the score slightly differently, because they do not need the leftmost or rightmost extra site, and because the last convex subset may have fewer than  $e$  sites remaining. For the last convex subset, we have  $k \bmod e = k - e(g-1)$  sites, to be precise, so we compute  $s(i, n)$  for all  $i$  and  $k \bmod e$  sites, plus one extra at  $m_{i-1} + \delta$ .

We then find the optimal subdivision of  $P$  into  $g$  convex subsets, such that the sum of the scores of all the subsets is maximal. We do this using dynamic programming to compute a function  $f(h, j)$ , representing the optimal score for the points  $p_1, \dots, p_j$  by using  $h$  convex subsets that partition  $p_1, \dots, p_j$ , and each convex subset is scored with the  $s(\cdot, \cdot)$  function. Since the application of dynamic programming is standard, we simply state:

$$f(h, j) = \max_{\ell < j} \{f(h-1, \ell) + s(\ell+1, j)\}.$$

The value  $f(g, n)$  then gives the maximum sum of scores when subdividing  $P$  into  $g$  subsets; a set  $S'$  of  $k + 2(g-1)$  sites attains this. We prove in the next lemma that the score of the Voronoi partition  $S'$  obtained by the approximation algorithm is at least the score of an optimal solution  $S^*$ . Since  $S'$  uses more sites than  $S^*$ , we continue to show that we can reduce the number of sites in  $S'$  and not lose much in the score.

**Lemma 6.7.** *The score of the calculated sites  $S'$  is at least the score  $\delta^*$  of an optimal solution  $S^*$  with  $k$  sites.*

*Proof.* Assume that the optimal solution  $S^*$  with  $k$  sites uses  $e$  sites whose cells contain  $p_i, \dots, p_j$ . We denote these sites by  $s_1^*, \dots, s_e^*$ . We can assume that  $s_1^*$  scores at least one point (namely  $p_i$ ), otherwise we just remove  $s_1^*$  and let  $s_2^*$  be the leftmost point. The extra point can be inserted anywhere between  $s_2^*$  and  $s_e^*$ , and this does not reduce the score of the solution. Similarly, we can assume that the cell of  $s_e^*$  contains at least  $p_j$ . Consequently, we can assume that there exists an optimal solution where the  $e - 1$  boundaries between  $s_1^*, \dots, s_e^*$  lie between  $p_i$  and  $p_j$ .

Our approximation algorithm calculated a score  $s(i, j)$  for  $p_i, \dots, p_j$  that uses  $e + 2$  sites, where the leftmost one is fixed at  $q_i + \delta$  and the rightmost one is fixed at  $q_{j-1} - \delta$ . Let us call the sites in our approximation  $a_0, \dots, a_{e+1}$ . We show that the placement of  $a_1, \dots, a_e$  between  $a_0$  and  $a_{e+1}$  allows us to get exactly the same boundaries between Voronoi cells as  $s_1^*, \dots, s_e^*$ , and two more boundaries. Since a refinement of a set of boundaries can never score lower with the richness measure, the result then follows.

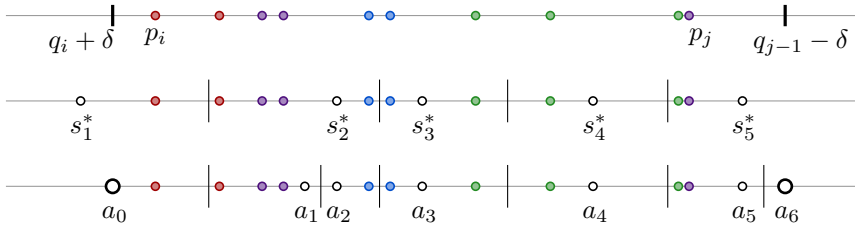


Figure 6.8: Top: a set  $p_i, \dots, p_j$  of points, scored (middle) by sites  $s_1^*$  to  $s_5^*$  in a possible optimal solution. Bottom: Seven sites  $a_0, \dots, a_6$  that score at least as high as the five sites  $s_1^*, \dots, s_5^*$  from  $S^*$ . On the left, the first case, and on the right, the second case of the argument.

To show that  $s(i, j)$  is at least the summed score of  $s_1^*, \dots, s_e^*$ , we observe that if we let  $a_h = s_h^*$  for  $2 \leq h \leq e - 1$ , all boundaries between  $a_2, \dots, a_{e-1}$  coincide with all boundaries between  $s_2^*, \dots, s_{e-1}^*$ . Now there are two cases for the left side of the solutions:  $s_1^*$  is left of  $a_0$  or not, see Figure 6.8. If  $s_1^*$  is to the left of  $a_0$ , then we can place  $a_1$  so that the boundary between  $s_1^*$  and  $s_2^*$  is the same as the boundary between  $a_0$  and  $a_1$ . The cell of  $s_2^*$  is now subdivided and scored by  $a_1$  and  $a_2$ , which cannot be worse. If  $s_1^*$  is to the right of  $a_0$ , then we place  $a_1$  at  $s_1^*$ , and now the cell of  $s_1^*$  is subdivided into two cells for  $a_0$  and  $a_1$ .

On the right side, we can show in the same way that either the cell of  $s_{e-1}^*$  is subdivided in two cells, or the cell of  $s_e^*$  is. Since this part of the approximation algorithm is brute force, this set of boundaries is examined and found.

This argument applies to all groups of  $e$  sites in the optimal solution, with any convex subset  $p_i, \dots, p_j$  in their cells. Since the dynamic program optimizes over all

combinations of groups of  $e$  sites with two extra sites, the score of  $S'$  is at least  $\delta^*$ .  $\square$

The remainder of the algorithm is simple: we determine the score of each site, choose the site with the lowest score, and remove it. After  $2(g-1)$  iterations, we have a set of  $k$  sites, which we show to be a  $(1-\varepsilon)$ -approximation of the best possible with  $k$  sites.

**Lemma 6.8.** *Reducing the set  $S'$  of sites to a set with  $k$  sites costs no more than  $\varepsilon$  times the score of  $S'$*

*Proof.* When removing a site the overall score drops by at most the score of the cell of the site that was removed. So one by one, we choose the site whose associated cell has the lowest score and remove it. Let  $Y_0$  be the score of the set of sites  $S'$ . For  $w = |S'| = k + 2g - 2$ , the score drops by at most  $Y_0/w$  for removing the first cell, thus the score after removing that cell is larger than  $Y_1 = Y_0 \frac{w-1}{w}$ . Iteratively, we define  $Y_i = Y_{i-1} \frac{w-i}{w-(i-1)}$ . Using a telescope sum we get  $Y_i = Y_0 \frac{w-i}{w}$ . Thus the score after removing  $2g-2$  sites is at least  $Y_{2g-2} = Y_0 \frac{w-2g+2}{w}$ . So the score  $Y'$  of the remaining  $k$  sites is at least  $Y_{2g-2} \geq \delta^* \frac{(k+2g-2)-2g+2}{k+2g-2} = \delta^* \frac{k}{k+2\lfloor k/e \rfloor - 2} \geq \delta^* \frac{k}{k+2(k/e+1)-2} = \delta^* \frac{1}{1+2/e}$ . Thus  $Y'(1+2/e) \geq \delta^* \iff Y'(1+\varepsilon) \geq \delta^* \implies Y' \geq \delta^*(1-\varepsilon)$ .  $\square$

Theorem 6.9 directly follows.

**Theorem 6.9.** *Let  $P$  be a set of  $n$  points in 1D, let  $k$  be a positive integer, and let  $\varepsilon > 0$  be a constant. There is a polynomial-time  $(1-\varepsilon)$ -approximation algorithm for computing a diverse Voronoi partition on  $P$  with  $k$  sites based on richness diversity.*

## 6.6 Diverse Convex Partition is NP-Hard in 2D

We show that the following diverse convex partition problem is NP-hard: given a set of colored points in the plane, partition the plane into the minimum number of convex regions so that the total diversity score according to richness is the same as the number of points (*full score*). Note that no subset in the partition has two points of the same color if it has full score. It is a special case of the original diverse convex partition problem, and easier to show NP-hard due to the property just given.

The main part of the proof shows that a slightly more general problem is NP-hard: the convex sets need not partition the plane, but must still be disjoint. These convex sets can now be the convex hulls of the points they contain. At the end of the section we show that our proof also applies to the stated diverse convex partition problem of full score.

### 6.6.1 Construction

Before we start with the proof, we study a special colored grid structure that is essential in the reduction presented later. It provides a scaffold (or padding) to limit the possibilities in the actual reduction.

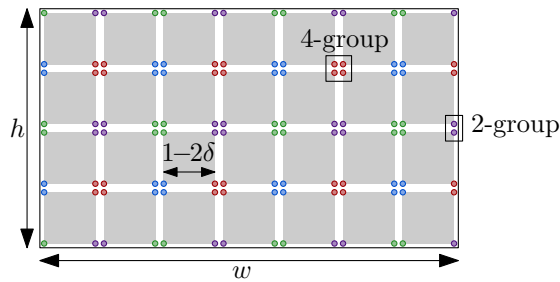


Figure 6.9: The set  $G$  of  $4wh$  colored points, a 4-group, and a 2-group.

**Definitions.** Consider the integer grid; we let the *4-fold colored grid point set* be the point set with all points  $(i \pm \delta, j \pm \delta)$  with colors: green if  $i$  and  $j$  are even; yellow if  $i$  is odd and  $j$  is even; blue if  $i$  is even and  $j$  is odd; red if  $i$  and  $j$  are odd. We choose a rectangular portion  $[0 : w] \times [0 : h]$  of the 4-fold colored grid point set and call it  $G$ ; see Figure 6.9. It has  $4wh$  colored points. We let  $n = \max(w, h)$  and  $\delta = 1/n^2$ .

The four equal-colored points near a grid point are called a *4-group*. They form a tiny square. The choice of  $\delta$  ensures that any line segment that connects a point in a 4-group at  $(i, j)$  to a point in a 4-group at  $(i', j')$  does not touch any (tiny) 4-group square if  $|i' - i|$  and  $|j' - j|$  are relative prime. In an  $n \times n$  unit grid, it is known that the shortest strictly positive distance from a line segment between grid points to another grid point is  $\Omega(1/n)$ , and the 4-group points are just  $\sqrt{2}/n^2$  away from their closest grid point.

A *square-set* is a set of four different-colored points of  $G$  that form a square of side length  $1 - 2\delta$ . These squares are shown in gray in the figures. A *4-set* is a set of four different-colored points of  $G$  whose convex hull does not contain any other points of  $G$ . We observe that  $G$  has a diverse convex partition into  $wh$  convex sets with diversity score  $4wh$ , the best possible, by choosing the  $wh$  square-sets. A *3-set* is a set of three different-colored points of  $G$  whose convex hull does not contain any other points of  $G$ . A *2-set* is a pair of points such that the connecting line segment does not contain other points of  $G$ , a *1-set* is a single point of  $G$ , and a *0-set* is the empty set.

**Reduction.** We are now ready to reduce from MAXIMUM INDEPENDENT SET (MIS) IN ORTHOGONAL LINE SEGMENTS, which is NP-hard, if we allow two horizontal or two

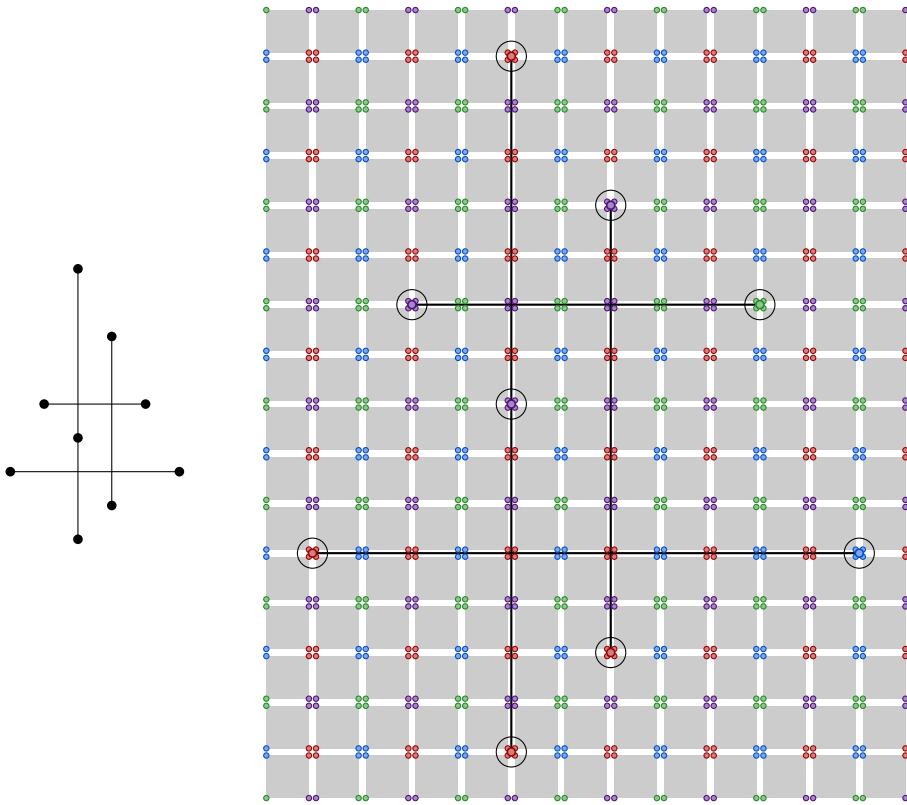


Figure 6.10: Reduction from an instance of MAXIMUM INDEPENDENT SET IN ORTHOGONAL LINE SEGMENTS to an instance of 2D DIVERSE CONVEX PARTITION.

vertical line segments to intersect in a joint endpoint. For completeness, we provide the proof by reduction from PLANAR MAX 2-SAT in Subsection 6.6.3 below. We can assume that all endpoints of the line segments have different  $x$ - and  $y$ -coordinates, with the exception of: (i) the two endpoints of one line segment, and (ii) pairs of parallel line segments that share an endpoint.

Figure 6.10 illustrates the reduction. Assume an instance of MIS IN ORTHOGONAL LINE SEGMENTS is given. We can normalize or scale the input any way we like, since this does not change the intersection graph. We choose a version with the following properties:

- All segment endpoints are integers.
- All endpoints of different line segments have different  $x$ -coordinates ( $y$ -coordinates), with the exception of a pair of vertical (horizontal) line seg-

ments that share an endpoint. Different  $x$ -coordinates ( $y$ -coordinates) are at least 2 apart.

- All line segments have odd length.

We can choose the  $x$ -coordinates of the endpoints easily from left to right. We choose the next  $x$ -coordinate 2 larger than the previous one, unless it is the right endpoint of a horizontal line segment. In that case we choose it 2 or 3 larger, whichever gives the line segment odd length. The  $y$ -coordinates are assigned analogously.

We now choose a 4-fold colored grid point set  $G$  large enough so that every segment endpoint is in the middle of a 4-group. We color the segment endpoint the same as the surrounding 4-group. The extra point is the *fifth point*, and together they are a *5-group*. The total number of fifth points is noted as  $f$ . Since segments have odd length, the fifth points corresponding to the endpoints of one segment have different colors.  $G$  has size  $O(n) \times O(n)$ .

We will now use many more colors to make sure that most of the 2, 3, and 4-sets cannot occur in a maximum score, minimum disjoint convex partition. We call the original four colors the *primary colors* and the extra colors for disqualifying shapes the *secondary colors*. It is important to realize that the “4” in “4-set” refers to the primary colors only. The same holds for 0, 1, 2, and 3-sets. We first define two concepts and then place the points of secondary color. Let the *extended square* of a square-set be the axis-aligned square of side length  $1 - \delta$  that has the same center of gravity as the square-set. For two points  $(x, y)$  and  $(x', y')$ , we say they are *horizontally (vertically) near-aligned* if  $|y - y'| \leq 2\delta$  (respectively  $|x - x'| \leq 2\delta$ ). Two points are *near-aligned* if they are vertically or horizontally near-aligned, otherwise we say they are *unaligned*.

On an  $n \times n$  4-fold colored grid point set, there are polynomially many 2-sets between two points belonging to different square-sets. For each such 2-set  $\{a, b\}$ , let  $S_a$  ( $S_b$ ) be the square-set of  $a$  ( $b$ ). Let  $I_a$  ( $I_b$ ) be the intersection of the line segment  $\overline{ab}$  and the extended square of  $S_a$  ( $S_b$ ). Then, let  $p_a$  ( $p_b$ ) be a point in the intersection  $I_a$  ( $I_b$ ) that is not yet a colored point. We take a new color and place two points of that color on  $p_a$  and  $p_b$ , as seen in Figure 6.11. We wish proceed the same for 2-sets between a fifth point and an unaligned point and therefore need the following lemma:

**Lemma 6.10.** *Let  $\{a, b\}$  be a 2-set where  $a$  and  $b$  are unaligned, and  $a$  is a fifth point. The segment  $\overline{ab}$  intersects two extended squares.*

*Proof.* We prove  $|a_x - b_x| > 1 - \delta$  or  $|a_y - b_y| > 1 - \delta$  holds. Since  $a$  and  $b$  are unaligned, this statement implies the correctness of the lemma.

Assume without loss of generality that  $b$  is below  $a$  and to the right of  $a$ . We assume for the sake of contradiction that  $|a_x - b_x| \leq 1 - \delta$  and  $|a_y - b_y| \leq 1 - \delta$ . Then  $b$  is not a fifth point, since different  $x$ -coordinates ( $y$ -coordinates) of fifth points are at least 2 apart. The point  $b$  is an element of the square-set  $S$  directly to the bottom-right of  $a$ . Only the bottom right point of  $S$  is unaligned with  $a$ , implying  $b$  is that point. Then, the segment  $\overline{ab}$  intersects the top-left point of  $S$ , a contradiction.  $\square$

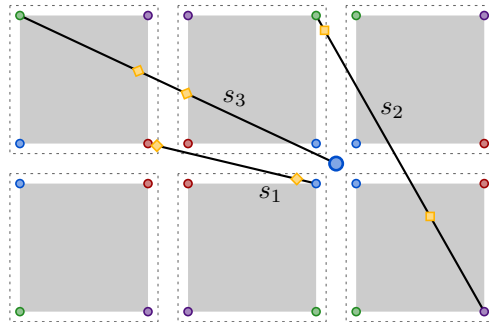


Figure 6.11: The procedure for adding secondary colors using extended squares (dashed). The figure shows three 2-sets: one between two near-aligned points of different square-sets ( $s_1$ ), one between two unaligned points of different square-sets ( $s_2$ ), one between a fifth point and an unaligned point in a square-sets ( $s_3$ ).

There are also polynomially many 2-sets with two unaligned fifth points and polynomially many 2-sets with a fifth point and an unaligned point in a square-set. Let  $\{a, b\}$  be such a 2-set. By Lemma 6.10, the line segment  $\overline{ab}$  intersects at least two extended squares. Let  $S$  and  $S'$  be two of those extended squares and let  $I$  ( $I'$ ) be the intersection of  $S$  ( $S'$ ) and  $\overline{ab}$ . Again, let  $p$  ( $p'$ ) be a point in the intersection  $I$  ( $I'$ ) that is not yet a colored point. We take a new color and place two points of that color on  $p$  and  $p'$ .

Now, the convex hull of any 3-set or 4-set with points from different square-sets contains two points of the same secondary color. Additionally, each 3-set and 4-set that contains a fifth point, also contains a point that is unaligned with that fifth point and thus its convex hull also contains two points of the same secondary color. Overall, we have that each 3-set and 4-set whose convex hull does not contain two points of the same color contains points from just a single square-set.

It remains to be proven that the MIS IN ORTHOGONAL LINE SEGMENTS instance  $I$  has a maximum independent set of size  $z$  if and only if the constructed DVP instance  $D$  can be subdivided into  $wh + f - z$  convex sets of full diversity score.

## 6.6.2 Equivalence

First, assume we are given a maximum independent set of  $I$  of size  $z$ . We construct a set of  $wh + f - z$  convex sets covering  $D$  as follows: each square-set forms its own convex set, together with all points of secondary color in the extended square of that square-set; for each segment in the independent set, the two fifth points corresponding to that segment form a convex set; all remaining fifth points are alone in their convex set. That construction has full score and no two convex sets intersect, since the segments were part of an independent set.



Inversely, assume we are given a set  $Q$  of  $wh + f - z$  convex sets of full diversity score. Recall that due to the added secondary colors, we can state:

**Lemma 6.11.** *Each set in  $Q$  is of one following types, classified using the points of primary color:*

- a 1, 2, 3, or 4-set with only point(s) from a single square-set;
- a 2-set with a fifth point and a near-aligned point of a square-set;
- a 2-set with two near-aligned fifth points;
- a 1-set with only one fifth point;
- a 0-set with no points of primary colors.

In the following arguments, we focus on points with primary colors. We start by removing all points of secondary color from the sets in  $S$ . From Lemma 6.11 we can deduce that if two unaligned points are in one set, they are in the same square-set. We can thus state the following:

**Lemma 6.12.** *Let  $R \in Q$  be a set and let  $S$  be a square-set. If  $R$  contains no points of  $S$ , the convex hull of  $R$  does not intersect the square induced by  $S$ .*

We call a set of  $Q$  a *standard* set if it is one of the following:

- a 4-set that is a square-set,
- a 2-set with two near-aligned fifth points,
- a 1-set with a single fifth point.

All other sets are called *non-standard* sets. We now transform  $Q$  without increasing the number of sets in  $Q$  such that it only contains standard sets and such that it still covers all points of primary color.

Let  $S$  be a square-set that is not a 4-set. Since different  $x$  and  $y$ -coordinates of fifth points differ by at least 2, there is at least one point  $a$  in  $S$  that is not near-aligned with any fifth point. By Lemma 6.11,  $a$  is in a 1, 2, or 3-set  $R$  with only point(s) from  $S$ . We remove all other points in  $S$  from their respective sets in  $Q$  and add them to  $R$ . This makes  $R$  a 4-set. Also, by Lemma 6.12, this does not create any intersections. We repeat this step for all square sets. Last, we remove all 0-sets.

After these transformations, all square-set are 4-sets, and there are no 0-sets in  $Q$ . This implies that all sets in  $Q$  are standart. The set  $Q$  still contains at most  $wh + f - z$  pairwise non-intersecting convex sets covering all points of primary color. We obtain an independent set of size  $z$  for the problem instance  $I$  by taking the segments corresponding to the 2-sets in  $S$ .

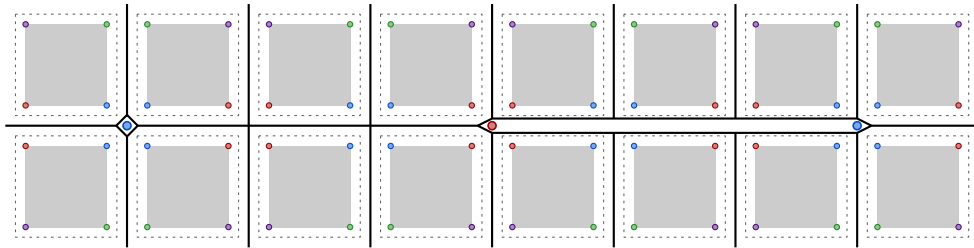


Figure 6.12: Converting a solution to partition by disjoint convex sets to a solution that partitions the plane into convex regions. We embed square-sets with their extended squares, 1-sets of fifth points, and 2-sets of near-aligned fifth points.

**Theorem 6.13.** *Computing a minimum-size convex partition of full richness diversity score is NP-hard.*

*Proof.* By the given reduction from MIS IN ORTHOGONAL LINE SEGMENTS. The reduction is polynomial because the grid  $G$  has  $O(n^2)$  points with primary colors, there are  $O(n)$  fifth points, and there are no more than  $O(n^4)$  points with secondary colors. An instance of MIS IN ORTHOGONAL LINE SEGMENTS can be solved using the reduction, as proven above.

We show that a solution to partition by disjoint convex sets can be transformed to a convex partition of the plane with the same number of regions and the same points in the regions as in the sets. In general this is not true [56], but for the subsets in our partition it is, and we need to show it only for one solution, since convex partition of the plane is more restricted. Figure 6.12 shows the easy conversion for the case where we have square-sets together with the corresponding extended squares, 1-sets with fifth points, and 2-sets with two fifth points only.  $\square$

### 6.6.3 NP-Hardness of Maximum Independent Set in Orthogonal Segments

It was claimed in [3] that the problem of computing a maximum independent set in the intersection graph of a set of orthogonal line segments is NP-hard. Tracing the references given in [3], this is not so obvious, although there exists a standard reduction. Since we need a restricted form of orthogonal line segments, we give the proof here that has this form, for completeness. The reduction is from PLANAR MAX 2-SAT, which is shown NP-hard in [67].

We use variable gadgets that consist of bundles of  $n$  horizontal line segments intersecting bundles of  $n$  vertical line segments, and this in a cycle structure (see Figure 6.13). The variable is set to TRUE if we take all vertical line segments and to FALSE if we take all horizontal ones. Choosing a mix will always cost us at least  $n$  line segments in the independent set.

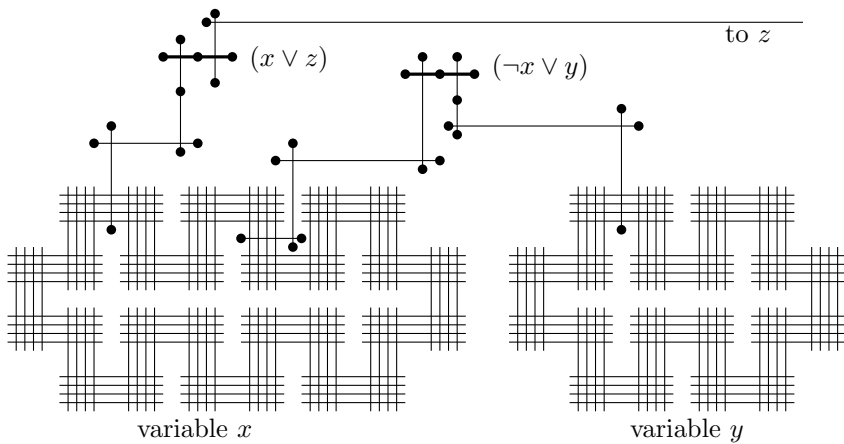


Figure 6.13: Reduction of PLANAR MAX 2-SAT to a specific form of maximum independent set in orthogonal line segments. Clause segments are thick, channel segments are thin and have endpoints shown, and variable segments are thin without endpoint markers.

When a literal  $x$  is used positively in a 2-SAT clause, it uses a vertical line segment that intersects a horizontal bundle, and when it is used in negated form, it uses a horizontal line segment that intersects a vertical bundle. In both cases, an even number of line segments is used to connect the variable with the clause, which is called a channel gadget. We may use two co-linear line segments that have a joint endpoint, to get the odd-even count right and arrive at a clause in the correct orientation.

A clause consists of two horizontal line segments that share one endpoint, and hence they are co-linear. The two channels of literals arrive vertically and each intersect a different line segment of the clause.

When a positive literal of a variable in TRUE setting arrives, it need not take the line segment that intersects the clause gadget, and the same is true for a negative literal of a variable in FALSE setting. This is due to the fact that channels have even length, and we can take half without intersecting the clause segment. Otherwise, we either cannot take half of the line segments in a channel, or we must take the line segment that intersects the clause gadget.

We can never take both line segments in a clause because they intersect, but we can take exactly one if at least one of the literals is satisfied. We can always take half the number of line segments in variables and channels in the independent set, and never more. We can take one more line segment for every clause that is satisfied. Hence, the number of satisfied clauses in a planar 2-SAT formula corresponds exactly to the size of the maximum independent set, minus half the line segments in variables and channels.

## 6.7 Conclusion

We have cast the non-spatial concept of diversity into a geometric setting and introduced the computational problem of computing geometric partitions with high diversity. We used two versions of diversity: richness and the Shannon index. While richness leads to computational problems common in computational geometry, the Shannon index needs new proof ingredients. The main open problems are finding a  $(1 - \varepsilon)$ -approximation scheme for 1D DVP for the Shannon index and an NP-hardness proof for 2D DCP for richness using constantly many colors. NP-hardness of 2D DCP for the Shannon index is also unresolved.

As an alternative to high within-cell diversity, a partition could aim for a high diversity *between* different cells. Then statistics like the Jaccard index could be used to measure differences between cells. We leave this extension to future work.

# Chapter 7

## Conclusion

### 7.1 Summary

In this thesis, we explore geometric measures and how to compute them. First, we use similarity measures to determine the quality of the output of a transformation. Second, we explore new algorithms and data structures that allow us to calculate these distance measures. Third, we present a new distance measure and investigate its computation. Last, we deal with diversity measures by researching how to subdivide a set into diverse subsets.

To be more precise, in Chapters 2 and 3, we transform a vector graphic into a raster image and measure the quality of the output using the Hausdorff distance. In Chapter 2, the vector graphic we want to transform is a set of regions. We investigate restrictions on the regions, like convexity or fatness, and prove worst case lower bounds of the Hausdorff distance between the regions and their corresponding grid polygons under these conditions. We finish by describing algorithms that match these bounds.

In Chapter 3, the regions we want to transform into a raster image are points. Even for this simple shape we were able to prove that determining pixels with the smallest possible Hausdorff distance to the input points is NP-hard. Nonetheless, we are able to find two polynomial-time algorithms that determine a set of pixels corresponding to the input points: first, we show an algorithm where the Hausdorff distance between the points and the pixels is at most a small constant factor larger than the optimal solution. Second, we show an algorithm where the Hausdorff distance between the points and the pixels is at most a small additive constant larger than the optimal solution.

In Chapter 4, we consider the computation of the Hausdorff distance. We present a data structure on a set of segments that allows queries of the following type: for a given segment, we can quickly determine the Hausdorff distance between the

query segment and the set of segments the data structure is built on. We provided a mechanism to balance between space usage and preprocessing time for that data structure.

In Chapter 5, we explore a new similarity measure, the  $k$ -Fréchet distance. We define the two variants: the cover distance and the cut distance. Both bridge between the Hausdorff and the Fréchet distance and measure the similarity between two curves. Even though both the Fréchet distance and the Hausdorff distance are easy to compute, the  $k$ -Fréchet distances are NP-hard to compute. Still, we present efficient algorithms that compute the cover or cut distance with specific restrictions.

In Chapter 6, we focus on diversity measures instead of similarity measures like in the previous chapters. For sets of colored points we use the species richness measure or the Shannon index to determine its diversity. We investigate the problem of how to subdivide a set of colored points into subsets such that the subsets are diverse. We present results for multiple variants: we subdivide either into convex subsets or using a Voronoi diagram, and investigate the problem both in 1D and in 2D.

## 7.2 Open Problems

We assume in Chapter 2 that our regions all had the same shape and size characteristics. In some cases it is interesting to see what happens in combinations. For example, what Hausdorff distance can we achieve, if only one set is of arbitrary shape and all others are point regions? Another avenue is to improve the output our construction algorithms produce. We concentrated on worst-case optimal bounds and as a result our constructive proofs of the upper bounds will often give visually unfortunate output. Additionally, we do not consider improving the Hausdorff distance between the input regions and the constructed grid polygons, even when doing so is possible. So, multiple avenues for new research open up: similar to Chapter 3, investigating approximation algorithms is an interesting open problem. On the other hand, after constructing a grid polygon using one of the algorithms in Chapter 2, one could perform a series of small steps such that each step decreases the Hausdorff distance or the area of symmetric difference between a convex region and the corresponding grid polygon, similar to the heuristic improvements proposed by Bouts et al. [26]. This would increase the similarity between the regions and the grid polygons. It would be interesting to investigate how much such heuristics could improve the output and how far from an optimal solution the final result is, depending on a variety of parameters and measures.

The approximation algorithms presented in Chapter 3 for mapping points to the grid can be extended in multiple ways. Similar to Chapter 2, one can aim to produce approximation algorithms for more complex regions: the concepts presented in Chapter 3 may lead to an approximation algorithm that maps convex regions to the grid. The results can also be extended in a different way: the NP-hardness proof in Section 3.2, implies that it is NP-hard to approximate the optimal solution by factor

$\sqrt{5/2} \approx 1.58$ , or to approximate an optimal solution within an additive constant of  $\sqrt{5} - \sqrt{2} \approx 0.82$ . In Section 3.3, we prove that, in polynomial time, we can achieve a Hausdorff distance between the points and the pixels of at most  $8\delta^*$ , where  $\delta^*$  is the Hausdorff distance of an optimal solution. Alternatively, we can achieve a Hausdorff distance between the points and the pixels of at most  $2\sqrt{2}(\lceil\delta^*\rceil + 1) \approx 2.83(\lceil\delta^*\rceil + 1)$  or at most  $\delta^* + \sqrt{2} \approx \delta^* + 1.41$ . It would be interesting to close those gaps.

In Chapter 4, we show that we can answer queries on the Hausdorff distance between a set of segments and a query segment. The next step here is to work this data structure into a way to determine the Hausdorff distance between two sets of segments and update this distance while the sets change. As a first step, one goal is to make the query data structure dynamic.

In Chapter 5, we define a new distance that bridges between the Hausdorff and weak Fréchet distance. In a similar vein, one can explore a measure bridging between the weak Fréchet distance and the Fréchet distance, that is, limit how much backtracking is allowed. Gheibi et al. [63] already considered the variant where the amount of backtracking is limited by the union of the portions of backward movements. Alternatively, one can limit the number of times a single point can be visited, or limit the number of times a traversal of the curves can switch directions. Coming back to the  $k$ -Fréchet distance, the cut variant has proven to be harder than the cover distance. The NP-hardness proof for the cut distance uses a value for  $k$  that depends on the number of line segments that form the curves [38]. The only known algorithm for the cut distance, presented in Section 5.5, works only for  $k = 2$ . It would be interesting to know the bounds on the computability of this problem. Is there a polynomial time algorithm to decide if two curves are within cut distance  $\varepsilon$  for every fixed value  $k$ ? Is deciding that question even a problem within NP, or does this problem lie within a different complexity class?

Last, in Chapter 6, we introduced the concept of partitioning space into diverse convex sets. We considered many variations, and presented algorithms and NP-hardness proofs. The main open problems are finding a  $(1 - \varepsilon)$ -approximation scheme for 1D DVP for the Shannon index and an NP-hardness proof for 2D DCP for richness using constantly many colors. NP-hardness of 2D DCP for the Shannon index is also unresolved. As an alternative to high within-cell diversity, a partition could aim for a high diversity between different cells. Then statistics like the Jaccard index could be used to measure differences between cells. It would also be highly interesting to see the concept of diverse geometric partition applied to ecological studies and to see if the partitions match the ones present in nature. Furthermore, we are interested in the cells produced by diverse partitions of random sets. For both cases, it would be beneficial to have an approximation or FPT algorithm for diverse partition in 2D.

On that note, a general idea is to apply the concepts developed in the thesis to the field. For example, we are interested in which circumstances the  $k$ -Fréchet distance will find application, even though its computation time is worse than the Hausdorff

distance or (weak) Fréchet distance.

Another concept to approach is the interplay between multiple measures. If two sets are dissimilar, does that give an indication on how diverse the union of those sets is? We investigated subdividing a set into subsets that themselves are diverse. Now, does subdividing a set into subsets that are similar between each other yield comparable results to subdividing that set into diverse subsets? And under which conditions is that not the case?

Finally, measures can be a boon to procedural generation in terms of measuring the quality of the output. If procedural generation is used to generate maps for games for example, measures can be used to determine if the map is balanced for all players. A measure for how balanced a map is could simply be a similarity measure. But, if the game offers asymmetric gameplay, how do does one adapt that measure? Diversity measures on the other hand will help classify how varied the output of the procedural generation can be. New measures are needed, specifically designed for these applications.



# Bibliography

- [1] M. Abrahamsen, J. Erickson, I. Kostitsyna, M. Löffler, T. Miltzow, J. Urhausen, J. Vermeulen and G. Viglietta. Chasing Puppies: Mobile Beacon Routing on Closed Curves. *37th International Symposium on Computational Geometry (SoCG)*, pages 5:1–5:19, 2021.
- [2] P. K. Agarwal. Ray Shooting and Other Applications of Spanning Trees with Low Stabbing Number. *SIAM Journal on Computing*, 21(3):540–570, 1992.
- [3] P. K. Agarwal and N. H. Mustafa. Independent Set of Intersection Graphs of Convex Objects in 2D. *Computational Geometry*, 34(2):83–95, 2006.
- [4] H. A. Akitaya, M. Buchin, L. Ryvkin and C. D. Tóth. Rock Climber Distance: Frogs versus Dogs. *31st Canadian Conference on Computational Geometry (CCCG)*, pages 210–217, 2019.
- [5] H. A. Akitaya, M. Buchin, L. Ryvkin and J. Urhausen. The k-Fréchet Distance Revisited and Extended. *Abstracts of the 35th European Workshop on Computational Geometry (EuroCG)*, 2019.
- [6] H. A. Akitaya, M. Buchin, L. Ryvkin and J. Urhausen. The k-Fréchet Distance: How to Walk Your Dog While Teleporting. *30th International Symposium on Algorithms and Computation (ISAAC)*, pages 50:1–50:15, 2019.
- [7] H. Alt, P. Braß, M. Godau, C. Knauer and C. Wenk. Computing the Hausdorff Distance of Geometric Patterns and Shapes. *Discrete and Computational Geometry*, pages 65–76, 2003.
- [8] H. Alt and M. Godau. Computing the Fréchet Distance between Two Polygonal Curves. *International Journal of Computational Geometry & Applications*, 5:75–91, 1995.

- [9] H. Alt, C. Knauer and C. Wenk. Comparison of Distance Measures for Planar Curves. *Algorithmica*, 38(1):45–58, 2004.
- [10] E. Althaus, F. Eisenbrand, S. Funke and K. Mehlhorn. Point Containment in the Integer Hull of a Polyhedron. *15th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 929–933, 2004.
- [11] B. Aronov, P. Bose, E. D. Demaine, J. Gudmundsson, J. Iacono, S. Langerman and M. Smid. Data Structures for Halfplane Proximity Queries and Incremental Voronoi Diagrams. *Algorithmica*, 80(11):3316–3334, 2018.
- [12] M. J. Atallah. A Linear Time Algorithm for the Hausdorff Distance between Convex Polygons. *Information Processing Letters (IPL)*, 17(4):207–209, 1983.
- [13] L. Babai, B. Just and F. Meyer auf der Heide. On the Limits of Computations with the Floor Function. *Information and Computation*, 78(2):99–108, 1988.
- [14] P. Bachmann. *Analytische Zahlentheorie*. Springer, 1904.
- [15] S. Bereg, P. Bose and D. Kirkpatrick. Equitable Subdivisions within Polygonal Regions. *Computational Geometry*, 34(1):20–27, 2006.
- [16] S. Bereg, J. M. Díaz-Báñez, D. Lara, P. Pérez-Lantero, C. Seara and J. Urrutia. On the Coarseness of Bicolored Point Sets. *Computational Geometry*, 46(1):65–77, 2013.
- [17] S. Bereg, F. Hurtado, M. Kano, M. Korman, D. Lara, C. Seara, R. I. Silveira, J. Urrutia and K. A. B. Verbeek. Balanced Partitions of 3-Colored Geometric Sets in the Plane. *Discrete Applied Mathematics*, 181:21–32, 2015.
- [18] M. de Berg, O. Cheong, M. van Kreveld and M. Overmars. *Computational Geometry: Algorithms and Applications*. 3rd edition, 2008.
- [19] M. de Berg, D. Halperin and M. Overmars. An Intersection-Sensitive Algorithm for Snap Rounding. *Computational Geometry*, 36(3):159–165, 2007.
- [20] M. de Berg and A. Khosravi. Optimal Binary Space Partitions for Segments in the Plane. *International Journal of Computational Geometry & Applications*, 22(03):187–205, 2012.

- 
- [21] M. de Berg, A. D. Mehrabi and T. Ophelders. Data Structures for Fréchet Queries in Trajectory Data. *29th Canadian Conference on Computational Geometry (CCCG)*, pages 214–219, 2017.
- [22] M. de Berg and O. Schwarzkopf. Cuttings and Applications. *International Journal of Computational Geometry & Applications*, 05(04):343–355, 1995.
- [23] S. Bespamyatnikh, D. Kirkpatrick and J. Snoeyink. Generalizing Ham Sandwich Cuts to Equitable Subdivisions. *Discrete & Computational Geometry*, 24(4):605–622, 2000.
- [24] P. V. M. Blagojević, G. Rote, J. K. Steinmeyer and G. M. Ziegler. Convex Equipartitions of Colored Point Sets. *Discrete & Computational Geometry*, 61(2):355–363, 2019.
- [25] A. Borodin, A. Jain, H. C. Lee and Y. Ye. Max-Sum Diversification, Monotone Submodular Functions, and Dynamic Updates. *ACM Transactions on Algorithms (TALG)*, 13(3):1–25, 2017.
- [26] Q. W. Bouts, I. Kostitsyna, M. van Kreveld, W. Meulemans, W. Sonke and K. A. B. Verbeek. Mapping Polygons to the Grid with Small Hausdorff and Fréchet Distance. *24th European Symposium on Algorithms (ESA)*, pages 22:1–22:16, 2016.
- [27] K. Bringmann. Why Walking the Dog Takes Time: Fréchet Distance Has No Strongly Subquadratic Algorithms Unless SETH Fails. *IEEE 55th Symposium on Foundations of Computer Science (FOCS)*, pages 661–670, 2014.
- [28] K. Bringmann and B. R. Chaudhury. Polyline Simplification has Cubic Complexity. *Journal of Computational Geometry*, 11(2):94–130, 2021.
- [29] K. Q. Brown. Geometric Transforms for Fast Geometric Algorithms. Tech. rep. Carnegie Mellon University, 1979.
- [30] K. Buchin, M. Buchin, C. Knauer, G. Rote and C. Wenk. How Difficult Is It to Walk the Dog? *Abstracts of the 23rd European Workshop on Computational Geometry (EuroCG)*, pages 170–173, 2007.
- [31] K. Buchin, M. Buchin, W. Meulemans and W. Mulzer. Four Soviets Walk the Dog: Improved Bounds for Computing the Fréchet Distance. *Discrete and Computational Geometry*, 58:180–216, 2017.

- [32] K. Buchin, M. Buchin and Y. Wang. Exact Algorithms for Partial Curve Matching via the Fréchet Distance. *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 645–654, 2009.
- [33] K. Buchin, M. Löffler, T. Ophelders, A. Popov, J. Urhausen and K. A. B. Verbeek. Computing the Fréchet Distance between Uncertain Curves in One Dimension. *17th Algorithms and Data Structures Symposium (WADS)*, 2021.
- [34] K. Buchin, M. Löffler, T. Ophelders, A. Popov, J. Urhausen and K. A. B. Verbeek. Computing the Fréchet Distance between Uncertain Curves in One Dimension. *Computational Geometry*, 109: 2023.
- [35] M. Buchin. On the Computability of the Fréchet Distance between Triangulated Surfaces. PhD thesis. Free University Berlin, 2007.
- [36] M. Buchin, A. Driemel and B. Speckmann. Computing the Fréchet Distance with Shortcuts Is NP-Hard. *30th International Symposium on Computational Geometry (SoCG)*, pages 367–376, 2014.
- [37] M. Buchin, I. van der Hoog, T. Ophelders, L. Schlipf, R. I. Silveira and F. Staals. Efficient Fréchet Distance Queries for Segments. arXiv. 2022. doi: 10.48550/arXiv.2203.01794.
- [38] M. Buchin and L. Ryvkin. The  $k$ -Fréchet Distance of Polygonal Curves. *Abstracts of the 34th European Workshop on Computational Geometry (EuroCG)*, 2018.
- [39] M. Buchin, L. Ryvkin and J. Urhausen. Computing the Cut Distance of Two Curves. *Abstracts of the 36th European Workshop on Computational Geometry (EuroCG)*, 2020.
- [40] S. Cabello. Approximation Algorithms for Spreading Points. *Journal of Algorithms*, 62(2):49–73, 2007.
- [41] T. M. Chan and K. Tsakalidis. Optimal Deterministic Algorithms for 2-D and 3-D Shallow Cuttings. *Discrete & Computational Geometry*, 56(4):866–881, 2016.
- [42] B. Chazelle. Cutting Hyperplanes for Divide-and-Conquer. *Discrete & Computational Geometry*, 9(2):145–158, 1993.

- 
- [43] F. Chierichetti, R. Kumar, S. Lattanzi and S. Vassilvitskii. Fair Clustering through Fairlets. *Advances in Neural Information Processing Systems*, pages 5029–5037, 2017.
- [44] T. Christ, D. Pálvölgyi and M. Stojaković. Consistent Digital Line Segments. *Discrete & Computational Geometry*, 47(4):691–710, 2012.
- [45] J. Chun, K. Kikuchi and T. Tokuyama. Consistent Digital Curved Rays. *Abstracts of the 34th European Workshop on Computational Geometry (EuroCG)*, 2019.
- [46] J. Chun, M. Korman, M. Nöllenburg and T. Tokuyama. Consistent Digital Rays. *Discrete & Computational Geometry*, 42(3):359–378, 2009.
- [47] C. Colombe and K. Fox. Approximating the (Continuous) Fréchet Distance. *37th International Symposium on Computational Geometry (SoCG)*, pages 26:1–26:14, 2021.
- [48] S. A. Cook. The Complexity of Theorem-Proving Procedures. *3rd ACM Symposium on Theory of Computing*, pages 151–158, 1971.
- [49] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein. *Introduction to Algorithms*. MIT press, 2022.
- [50] C. van Dommelen, M. van Kreveld and J. Urhausen. Spiroplots: A New Discrete-Time Dynamical System to Generate Curve Patterns. *Bridges*, pages 353–360, 2020.
- [51] C. van Dommelen, M. van Kreveld and J. Urhausen. The Spiroplot App. *Media Exposition at the 36th International Symposium on Computational Geometry (SoCG)*, pages 71:1–71:5, 2020.
- [52] A. Driemel and S. Har-Peled. Jaywalking Your Dog - Computing the Fréchet Distance with Shortcuts. arXiv. 2011. doi: 10.48550/arXiv.1107.1720.
- [53] A. Driemel, S. Har-Peled and C. Wenk. Approximating the Fréchet Distance for Realistic Curves in Near Linear Time. *Discrete and Computational Geometry*, 48:94–127, 2012.
- [54] M. Drosou, H. V. Jagadish, E. Pitoura and J. Stoyanovich. Diversity in Big Data: A Review. *Big Data*, 5(2):73–84, 2017.

- [55] A. Dumitrescu and J. Pach. Partitioning Colored Point Sets into Monochromatic Parts. *International Journal of Computational Geometry & Applications*, 12(05):401–412, 2002.
- [56] H. Edelsbrunner, A. D. Robison and X.-J. Shen. Covering Convex Sets with Non-overlapping Polygons. *Discrete Mathematics*, 81(2):153–164, 1990.
- [57] J. Erickson, I. van der Hoog and T. Miltzow. Smoothing the Gap between NP and ER. *SIAM Journal on Computing*, :FOCS20–102, 2022.
- [58] R. Z. Farahani, M. SteadieSeifi and N. Asgari. Multiple Criteria Facility Location Problems: A Survey. *Applied Mathematical Modelling*, 34(7):1689–1709, 2010.
- [59] T. Feng, L. Ryvkin, J. Urhausen and G. Viglietta. Complexity of Critter Crunch. *IEICE Transactions on Information and Systems*, 105(3):517–531, 2022.
- [60] J. Fiala, J. Kratochvil and A. Proskurowski. Systems of Distant Representatives. *Discrete Applied Mathematics*, 145:306–316, 2005.
- [61] M. Fréchet. Sur Quelques Points du Calcul Fonctionnel. *Rendiconti del Circolo Mathematico Di Palermo*, : 1906.
- [62] M. R. Garey and D. S. Johnson. *Computers and Intractability*. W. H. Freeman and Company, 1979.
- [63] A. Gheibi, A. Maheshwari and J.-R. Sack. Weighted Minimum Backward Fréchet Distance. *Theoretical Computer Science*, 783:9–21, 2019.
- [64] A. van Goethem, I. Kostitsyna, M. van Kreveld, W. Meulemans, M. Sondag and J. Wulms. The Painter’s Problem: Covering a Grid with Colored Connected Polygons. *25th International Symposium on Graph Drawing and Network Visualization (GD)*, pages 492–505, 2018.
- [65] J. E. Goodman, J. Pach and C.-K. Yap. Mountain Climbing, Ladder Moving, and the Ring-Width of a Polygon. *The American Mathematical Monthly*, 96:494–510, 1989.
- [66] M. T. Goodrich, L. J. Guibas, J. Hershberger and P. J. Tanenbaum. Snap Rounding Line Segments Efficiently in Two and Three Dimensions. *13th International Symposium on Computational Geometry (SoCG)*, pages 284–293, 1997.

- 
- [67] L. J. Guibas, J. E. Hershberger, J. S. B. Mitchell and J. S. Snoeyink. Approximating Polygons and Subdivisions with Minimum-Link Paths. *International Journal of Computational Geometry & Applications*, 3(04):383–415, 1993.
- [68] L. J. Guibas and F. F. Yao. On Translating a Set of Rectangles. *12th ACM Symposium on Theory of Computing*, pages 154–160, 1980.
- [69] P. Haghhighatkhah, W. Meulemans, B. Speckmann, J. Urhausen and K. A. B. Verbeek. Obstructing Classification via Projection. *46th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, 2021.
- [70] P. Haghhighatkhah, W. Meulemans, B. Speckmann, J. Urhausen and K. A. B. Verbeek. Obstructing Classification via Projection. *Abstracts of the 37th European Workshop on Computational Geometry (EuroCG)*, 2021.
- [71] P. Haghhighatkhah, W. Meulemans, B. Speckmann, J. Urhausen and K. A. B. Verbeek. Obstructing Classification via Projection. *Computing in Geometry and Topology*, 1(1):2:1–2:21, 2022.
- [72] S. Har-Peled. *Geometric Approximation Algorithms*. American Mathematical Society, 2011.
- [73] T. E. Harris and F. S. Ross. Fundamentals of a Method for Evaluating Rail Net Capacities. Tech. rep. RAND Corporation, 1955.
- [74] D. Hartvigsen. Recognizing Voronoi Diagrams with Linear Programming. *ORSA Journal on Computing*, 4(4):369–374, 1992.
- [75] W. Harvey. Computing Two-Dimensional Integer Hulls. *SIAM Journal on Computing*, 28(6):2285–2299, 1999.
- [76] F. Hausdorff. *Grundzüge der Mengenlehre*. Veit & Company, 1914.
- [77] J. Hershberger. Stable Snap Rounding. *Computational Geometry*, 46(4):403–416, 2013.
- [78] A. F. Holmsen, J. Kynčl and C. Valculescu. Near Equipartitions of Colored Point Sets. *Computational Geometry*, 65:35–42, 2017.
- [79] I. van der Hoog, V. Keikha, M. Löffler, A. Mohades and J. Urhausen. Maximum-Area Triangle in a Convex Polygon, Revisited. *Information Processing Letters (IPL)*, 161:105943, 2020.

- [80] I. van der Hoog, M. van de Kerkhof, M. van Kreveld, M. Löffler, F. Staals, J. Urhausen and J. L. Vermeulen. Mapping Multiple Regions to the Grid with Bounded Hausdorff Distance. *17th Algorithms and Data Structures Symposium (WADS)*, pages 627–640, 2021.
- [81] I. van der Hoog, M. van de Kerkhof, M. van Kreveld, M. Löffler, F. Staals, J. Urhausen and J. L. Vermeulen. Mapping Multiple Regions to the Grid with Bounded Hausdorff Distance. *Abstracts of the 37th European Workshop on Computational Geometry (EuroCG)*, 2021.
- [82] P. Jungeblut, L. Kleist and T. Miltzow. The Complexity of the Hausdorff Distance. *38th International Symposium on Computational Geometry (SoCG)*, pages 48:1–48:17, 2022.
- [83] A. Kaneko and M. Kano. Discrete Geometry on Red and Blue Points in the Plane, a Survey. *Discrete and Computational Geometry, The Goodman-Pollack Festschrift*, pages 551–570, 2003.
- [84] V. Keikha, M. van der Kerkhof, M. van Kreveld, I. Kostitsyna, M. Löffler, F. Staals, J. Urhausen, J. L. Vermeulen and L. Wiratma. Convex Partial Transversals of Planar Regions. *Computational Geometry: Young Researchers Forum (YRF)*, 2018.
- [85] V. Keikha, M. van der Kerkhof, M. van Kreveld, I. Kostitsyna, M. Löffler, F. Staals, J. Urhausen, J. L. Vermeulen and L. Wiratma. Convex Partial Transversals of Planar Regions. *29th International Symposium on Algorithms and Computation (ISAAC)*, pages 52:1–52:12, 2018.
- [86] R. Klette and A. Rosenfeld. *Digital Geometry: Geometric Methods for Digital Picture Analysis*. Morgan Kaufmann, 2004.
- [87] R. Klette and A. Rosenfeld. Digital Straightness - a Review. *Discrete Applied Mathematics*, 139(1-3):197–230, 2004.
- [88] M. van Kreveld, M. Löffler and L. Wiratma. On Optimal Polyline Simplification Using the Hausdorff and Fréchet Distance. *Journal of Computational Geometry*, 11(1):1–25, 2020.
- [89] M. van Kreveld, B. Speckmann and J. Urhausen. Diverse Partitions of Colored Points. *17th Algorithms and Data Structures Symposium (WADS)*, pages 641–654, 2021.



- 
- [90] M. van Kreveld, B. Speckmann and J. Urhausen. Diverse Voronoi Partitions of 1D Colored Points. *Abstracts of the 36th European Workshop on Computational Geometry (EuroCG)*, 2020.
- [91] B. Lacevic and E. Amaldi. On Population Diversity Measures in Euclidean Space. *IEEE Congress on Evolutionary Computation*, pages 1–8, 2010.
- [92] R. Laxhammar and G. Falkman. Sequential Conformal Anomaly Detection in Trajectories Based on Hausdorff Distance. *14th International Conference on Information Fusion*, pages 1–8, 2011.
- [93] C. Liu. Nearly Optimal Planar  $k$  Nearest Neighbors Queries under General Distance Functions. *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2842–2859, 2020.
- [94] M. Löffler and M. van Kreveld. Largest Bounding Box, Smallest Diameter, and Related Problems on Imprecise Points. *Computational Geometry: Theory and Applications*, 43(4):419–433, 2010.
- [95] M. Löffler and W. Meulemans. Discretized Approaches to Schematization. *29th Canadian Conference on Computational Geometry (CCCG)*, 2017.
- [96] M. Löffler, J. A. Simons and D. Strash. Dynamic Planar Point Location with Sub-Logarithmic Local Updates. *13th International Symposium on Algorithms and Data Structures (WADS)*, pages 499–511, 2013.
- [97] M. Löffler, F. Staals and J. Urhausen. New Results on Trajectory Grouping under Geodesic Distance. *Abstracts of the 32nd European Workshop on Computational Geometry (EuroCG)*, 2016.
- [98] M. Löffler and J. Urhausen. Mapping Points to the Grid with Bounded Hausdorff Distance. *33rd Canadian Conference on Computational Geometry (CCCG)*, pages 47–55, 2021.
- [99] A. E. Magurran. *Ecological Diversity and Its Measurement*. Princeton University Press, 1988.
- [100] S. Majumder, S. C. Nandy and B. B. Bhattacharya. Separating Multi-Color Points on a Plane with Fewest Axis-Parallel Lines. *Fundamenta Informaticae*, 99(3):315–324, 2010.

- [101] T. Mchedlidze and J. Urhausen.  $\beta$ -Stars or On Extending a Drawing of a Connected Subgraph. *25th International Symposium on Graph Drawing and Network Visualization (GD)*, pages 416–429, 2018.
- [102] A. Nath and E. Taylor. k-Median Clustering under Discrete Fréchet and Hausdorff Distances. *36th International Symposium on Computational Geometry (SoCG)*, pages 58:1–58:15, 2020.
- [103] J. B. Orlin. Max Flows in  $O(nm)$  Time, or Better. *45th Symposium on Theory of Computing (STOC)*, pages 765–774, 2013.
- [104] J. Richter-Gebert. Conics and Their Duals. In: *Perspectives on Projective Geometry: A Guided Tour through Real and Complex Geometry*. 2011. Chap. 9, 145–166.
- [105] W. Rucklidge. *Efficient Visual Recognition Using the Hausdorff Distance*. Springer, 1996.
- [106] L. Ryvkin. On Distance Measures for Polygonal Curves Bridging between Hausdorff and Fréchet Distance. PhD thesis. Ruhr-Universität Bochum, 2021.
- [107] P. Schäfer. Untersuchungen zu Varianten des Fréchet-Abstands. MA thesis. Fernuniversität Hagen, 2019.
- [108] C. Scheffer. More Flexible Curve Matching via the Partial Fréchet Similarity. *International Journal of Computational Geometry and Applications*, 26:33–52, 2016.
- [109] F. Schuhmann, L. Ryvkin, J. D. McLaren, L. Gerhards and I. A. Solov'yov. Across Atoms to Crossing Continents: Application of Similarity Measures to Biological Location Data. *bioRxiv*, : 2022.
- [110] J. M. Scott, B. Csuti, J. D. Jacobi and J. E. Estes. Species Richness. *BioScience*, 37(11):782–788, 1987.
- [111] C. E. Shannon. A Mathematical Theory of Communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.
- [112] M. Sharir. Almost Tight Upper Bounds for Lower Envelopes in Higher Dimensions. *Discrete & Computational Geometry*, 12:327–345, 1994.

- 
- [113] F. Staals, J. L. Vermeulen and J. Urhausen. Querying the Hausdorff Distance of a Line Segment. *Abstracts of the 38th European Workshop on Computational Geometry (EuroCG)*, pages 60:1–60:8, 2022.
- [114] E. K. Tang, P. N. Suganthan and X. Yao. An Analysis of Diversity Measures. *Machine Learning*, 65(1):247–271, 2006.
- [115] H. Tuomisto. A Diversity of Beta Diversities: Straightening up a Concept Gone Awry. *Ecography*, 33(1):2–22, 2010.
- [116] G. Voronoi. Nouvelles Applications des Paramètres Continus à la Théorie des Formes Quadratiques. *Journal für die Reine und Angewandte Mathematik*, (134):97–102, 198–287, 1908.
- [117] R. H. Whittaker. Vegetation of the Siskiyou Mountains, Oregon and California. *Ecological Monographs*, 30(3):279–338, 1960.
- [118] K. Zheng, H. Wang, Z. Qi, J. Li and H. Gao. A Survey of Query Result Diversification. *Knowledge and Information Systems*, 51(1):1–36, 2017.



# Nederlandse Samenvatting

In dit proefschrift onderzoeken we geometrische maten en hoe we deze kunnen berekenen. Ten eerste gebruiken we gelijkheidsmaten om de kwaliteit van de uitvoer van een transformatie te bepalen. Ten tweede ontwikkelen we nieuwe algoritmes en datastructuren waarmee we deze afstandsmaten kunnen berekenen. Ten derde presenteren we een nieuwe gelijkheidsmaten en onderzoeken we de berekening ervan. Ten slotte behandelen we diversiteitsmaten door te onderzoeken hoe een verzameling in diverse deelverzamelingen onderverdeeld kan worden.

Om preciezer te zijn, transformeren we in Hoofdstukken 2 en 3 een vectorafbeelding naar een rasterafbeelding en meten we de kwaliteit van de uitvoer met behulp van de Hausdorff-afstand. In Hoofdstuk 2 is de vectorafbeelding die we willen transformeren een verzameling gebieden. We onderzoeken beperkingen op de gebieden, zoals convexiteit of dikheid, en bewijzen onder deze voorwaarden ondergrenzen voor de Hausdorff-afstand tussen de gebieden en hun overeenkomstige rasterafbeelding. We eindigen met het beschrijven van algoritmes die aan deze grenzen voldoen.

In Hoofdstuk 3 zijn de gebieden die we naar een rasterafbeelding willen transformeren punten. Zelfs voor deze eenvoudige vorm kunnen we bewijzen dat het bepalen van pixels met de kleinst mogelijke Hausdorff-afstand tot de invoerpunten NP-moeilijk is. Desondanks zijn we in staat om twee algoritmes met polynomiale tijdscomplexiteit te vinden die een verzameling pixels bepalen die overeenkomen met de invoerpunten: ten eerste presenteren we een algoritme waarbij de Hausdorff-afstand tussen de punten en de pixels hoogstens een kleine constante factor groter is dan de optimale oplossing. Ten tweede presenteren we een algoritme waarbij de Hausdorff-afstand tussen de punten en de pixels hoogstens een kleine additieve constante groter is dan de optimale oplossing.

In Hoofdstuk 4 beschouwen we de berekening van de Hausdorff-afstand. We presenteren een data structuur die een verzameling segmenten op slaat waarmee we snel kunnen bepalen wat de Hausdorff-afstand is tussen een query segment en de opgeslagen segmenten. We ontwikkelen een mechanisme om een balans te vinden tussen ruimtegebruik en voorbewerkingstijd voor die datastructuur.

In Hoofdstuk 5 introduceren we een nieuwe gelijkheidsmaten, de  $k$ -Fréchet-afstand. We definiëren twee varianten: de dekkingsafstand en de snijafstand. Beide overbruggen de kloof tussen de Hausdorff-afstand en de Fréchet-afstand en meten de gelijkheid tussen twee krommen. Hoewel zowel de Fréchet-afstand als de Hausdorff-afstand eenvoudig te berekenen zijn, zijn de  $k$ -Fréchet-afstanden NP-moeilijk om te berekenen. Toch presenteren we efficiënte algoritmes die de dekkings- of snijafstand berekenen onder specifieke voorwaarden.

In Hoofdstuk 6 richten we ons op diversiteitsmaten in plaats van gelijkheidsmaten zoals in de voorgaande hoofdstukken. Om de diversiteit van een verzameling gekleurde punten te bepalen gebruiken we de Shannon-index of de biodiversiteit gemeten in het aantal verschillende kleuren. Elk punt hierin representeert een individu, waarvan de kleur indicatief is voor de soort. We onderzoeken het probleem van hoe een verzameling gekleurde punten onderverdeeld kan worden in deelverzamelingen zodat de deelverzamelingen divers zijn. We presenteren resultaten voor meerdere varianten: we verdelen de punten in convexe deelverzamelingen, of we gebruiken een Voronoi-diagram, en we onderzoeken het probleem zowel in één als twee dimensie.

Met deze resultaten hebben we bijgedragen aan het verder begrip van de algoritmische complexiteit van verschillende geometrische maten.

# Curriculum Vitae

Jérôme Urhausen was born on the 11th of September in 1991 in Eupen, Belgium. He finished school in 2011 at the Athénée de Luxembourg (LU) with majors mathematics and computer science. He pursued a bachelor in computer science at Karlsruhe Institute of Technology (DE) and a bachelor in mathematics, also at KIT. His combined bachelor's thesis with the title "New Results on Trajectory Grouping under Geodesic Distance", written in cooperation with Utrecht University under the supervision of Maarten Löffler, Frank Staals, and Benjamin Niedermann, was defended in 2015. In 2017, he defended his master's thesis with the title "Extending Drawing with Fixed Inner Faces with and without Bends", supervised by Tamara Mchedlidze at KIT. Thus he achieved his master's degree in computer science with majors in algorithm engineering and parallel computing, and a minor in mathematics. He started his doctorate at the end of 2017 at University Utrecht under the supervision of Marc van Kreveld, Maarten Löffler, and Frank Staals. Since 2022, he is working as a consultant for technical software at ALTEN Netherlands.

During his studies he co-authored multiple papers, listed below in reverse chronological order. Formally reviewed publications are printed in black, presentations at conferences and conference proceedings that are not formally reviewed are printed in gray. The papers that this thesis is based on are highlighted with a star.

- K. Buchin, M. Löffler, T. Ophelders, A. Popov, J. Urhausen and K. A. B. Verbeek. Computing the Fréchet Distance between Uncertain Curves in One Dimension. *Computational Geometry*, 109: 2023.
- P. Haghightakhah, W. Meulemans, B. Speckmann, J. Urhausen and K. A. B. Verbeek. Obstructing Classification via Projection. *Computing in Geometry and Topology*, 1(1):2:1–2:21, 2022.
- ★ F. Staals, J. L. Vermeulen and J. Urhausen. Querying the Hausdorff Distance of a Line Segment. *Abstracts of the 38th European Workshop on Computational Geometry (EuroCG)*, pages 60:1–60:8, 2022.
- T. Feng, L. Ryvkin, J. Urhausen and G. Viglietta. Complexity of Critter Crunch. *IEICE Transactions on Information and Systems*, 105(3):517–531, 2022.

- P. Haghighatkhah, W. Meulemans, B. Speckmann, J. Urhausen and K. A. B. Verbeek. Obstructing Classification via Projection. *46th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, 2021.
- M. Löffler and J. Urhausen. Mapping Points to the Grid with Bounded Hausdorff Distance. *33rd Canadian Conference on Computational Geometry (CCCG)*, pages 47–55, 2021.
- K. Buchin, M. Löffler, T. Ophelders, A. Popov, J. Urhausen and K. A. B. Verbeek. Computing the Fréchet Distance between Uncertain Curves in One Dimension. *17th Algorithms and Data Structures Symposium (WADS)*, 2021.
- M. van Kreveld, B. Speckmann and J. Urhausen. Diverse Partitions of Colored Points. *17th Algorithms and Data Structures Symposium (WADS)*, pages 641–654, 2021.
- Jérôme Urhausen was selected for the Best Student Presentation Award.
- I. van der Hoog, M. van de Kerkhof, M. van Kreveld, M. Löffler, F. Staals, J. Urhausen and J. L. Vermeulen. Mapping Multiple Regions to the Grid with Bounded Hausdorff Distance. *17th Algorithms and Data Structures Symposium (WADS)*, pages 627–640, 2021.
- M. Abrahamsen, J. Erickson, I. Kostitsyna, M. Löffler, T. Miltzow, J. Urhausen, J. Vermeulen and G. Viglietta. Chasing Puppies: Mobile Beacon Routing on Closed Curves. *37th International Symposium on Computational Geometry (SoCG)*, pages 5:1–5:19, 2021.
- P. Haghighatkhah, W. Meulemans, B. Speckmann, J. Urhausen and K. A. B. Verbeek. Obstructing Classification via Projection. *Abstracts of the 37th European Workshop on Computational Geometry (EuroCG)*, 2021.
- I. van der Hoog, M. van de Kerkhof, M. van Kreveld, M. Löffler, F. Staals, J. Urhausen and J. L. Vermeulen. Mapping Multiple Regions to the Grid with Bounded Hausdorff Distance. *Abstracts of the 37th European Workshop on Computational Geometry (EuroCG)*, 2021.
- I. van der Hoog, V. Keikha, M. Löffler, A. Mohades and J. Urhausen. Maximum-Area Triangle in a Convex Polygon, Revisited. *Information Processing Letters (IPL)*, 161:105943, 2020.
- C. van Dommelen, M. van Kreveld and J. Urhausen. Spiroplots: A New Discrete-Time Dynamical System to Generate Curve Patterns. *Bridges*, pages 353–360, 2020.
- C. van Dommelen, M. van Kreveld and J. Urhausen. The Spiroplot App. *Media Exposition at the 36th International Symposium on Computational Geometry (SoCG)*, pages 71:1–71:5, 2020.



- 
- M. van Kreveld, B. Speckmann and J. Urhausen. Diverse Voronoi Partitions of 1D Colored Points. *Abstracts of the 36th European Workshop on Computational Geometry (EuroCG)*, 2020.
  - M. Buchin, L. Ryvkin and J. Urhausen. Computing the Cut Distance of Two Curves. *Abstracts of the 36th European Workshop on Computational Geometry (EuroCG)*, 2020.
  - ★ H. A. Akitaya, M. Buchin, L. Ryvkin and J. Urhausen. The k-Fréchet Distance: How to Walk Your Dog While Teleporting. *30th International Symposium on Algorithms and Computation (ISAAC)*, pages 50:1–50:15, 2019.
  - H. A. Akitaya, M. Buchin, L. Ryvkin and J. Urhausen. The k-Fréchet Distance Revisited and Extended. *Abstracts of the 35th European Workshop on Computational Geometry (EuroCG)*, 2019.
  - V. Keikha, M. van der Kerkhof, M. van Kreveld, I. Kostitsyna, M. Löffler, F. Staals, J. Urhausen, J. L. Vermeulen and L. Wiratma. Convex Partial Transversals of Planar Regions. *29th International Symposium on Algorithms and Computation (ISAAC)*, pages 52:1–52:12, 2018.
  - T. Mchedlidze and J. Urhausen.  $\beta$ -Stars or On Extending a Drawing of a Connected Subgraph. *25th International Symposium on Graph Drawing and Network Visualization (GD)*, pages 416–429, 2018.
  - V. Keikha, M. van der Kerkhof, M. van Kreveld, I. Kostitsyna, M. Löffler, F. Staals, J. Urhausen, J. L. Vermeulen and L. Wiratma. Convex Partial Transversals of Planar Regions. *Computational Geometry: Young Researchers Forum (YRF)*, 2018.
  - M. Löffler, F. Staals and J. Urhausen. New Results on Trajectory Grouping under Geodesic Distance. *Abstracts of the 32nd European Workshop on Computational Geometry (EuroCG)*, 2016.