Institut für Geodäsie und Geoinformation
Rheinische Friedrich-Wilhelms-Universität Bonn

Von der Agrar-, Ernährungs- und Ingenieurwissenschaftliche Fakultät genehmigte

DISSERTATION

zur Erlangung der Doktorwürde der Ingenieurswissenschaften (Dr.-Ing.)

Annika Bonerath
geboren in Bonn

Geometric Algorithms for the Visual Exploration of Spatiotemporal Data

Bonn, 2025

Angefertigt mit Genehmigung der Agrar-, Ernährungs- und Ingenieurwissenschaftlichen Fakultät der Universität Bonn.

Betreuer:	Jan-Henrik Haunert (IGG, Universität Bonn)
Gutachter:	Stephen G. Kobourov (TU München)
Gutachter:	Martin Nöllenburg (TU Wien)
Fachnahes Mitglied:	Jens Behley (IGG, Universität Bonn)
Prüfungsvorsitz:	Juliane Fluck (IGG, Universität Bonn und ZB MED)
Prüfungsdatum:	21. Juni 2024

Acknowledgements

First and foremost, I would like to thank my supervisor, Jan-Henrik Haunert. I am especially grateful for his support of my research: for his ideas, for our discussions, and for his feedback. Besides research, I am also grateful for the motivating and encouraging words, for the support in stressful times, and for all the opportunities he gives to his Ph.D. students. Thanks to Jan, I was able to attend conferences and do research stays at places like Chicago, Vancouver, and Cape Town which I will never forget. Lastly, since my Ph.D. time coincided with the COVID-19 pandemic, I am grateful that Jan tackled this challenge with special regard to creating a good working environment for his Ph.D. students.

Special thanks also go to Benjamin Niedermann, who supported me from the very first day of my Master's thesis with advice and discussions. We spend a lot of time at the whiteboard, discussing new research ideas, writing papers together, and spending late nights before deadlines at the office. It was a great support that Benjamin joined me for my first conference trips to Utrecht and Chicago.

I would like to thank Stephen Kobourov and Martin Nöllenburg who supported my research stays in Vancouver and Vienna. Both stays were great; I enjoyed the discussions and I am happy with the resulting projects.

Although I do not list all my coauthors by name, I thank all of them for their work and the time that they spent.

Many thanks to all my great colleagues who accompanied me during my Ph.D. Thank you Johannes for making the start of the Ph.D. time so easy. Thank you, Axel and Sven, that you went through the whole Ph.D. phase together with me. I am still surprised that Sven and I have been on the journey in academia together since the first day of our Bachelor's studies. I am especially thankful for everyone (Youness, Sven, Benjamin, Axel, Farzane, and Shiyaza) who shared an office with me and who had to deal with me before the first coffee of the day. Thanks to everyone for the meetings at the coffee machine, carnival, Christmas market visits, barbeque, bouldering game nights, and hikes. And especially during the COVID-19 times, the 9:30 a.m. Zoom meetings were an anchor to get social contacts during the day.

Finally, I need to thank my partner, my family, and my friends - without them, it would have been so much harder to complete this thesis. I am grateful for every type of support: their proofreading, their feedback on conference presentations, the fun moments, and their uplifting words when I faced a challenging situation.

Thanks to all of you!

Abstract

The visualization of spatial data is an important research field in geoinformation science. Especially nowadays, where positioning sensors are widely spread, many (large) data sets have spatial information. As an example, take a set of bird observations where a data point corresponds to the location of a bird sighting and possibly additional information (timestamp, photo, species specification, etc.). Often, such data sets are large and complex. Visualizations from cartography allow users to explore and analyze the data. This ranges from visualizations that give an overview of the data, to tools that enable a detailed analysis of data patterns. In this thesis, we develop methods that support such visualizations. To be more precise, we apply methods from theoretical computer science to existing visualizations from cartography to improve them, e.g., to decrease the time needed to produce a visualization.

This thesis has two parts. In the first part, we consider the spatiotemporal case where each data point is an event consisting of a point in space and time. For visualizing spatiotemporal data, it is common to use an interactive visualization. Here, we focus on filtering the data for time windows. Then, the data that temporally lies in the time window is visualized on a map. As visualizations, we consider three standard techniques for visualizing point sets.

- A standard visualization is the representation of the events with one or multiple polygons, i.e., we aggregate the points that lie in the time window into one or multiple polygons. In particular, we build on the existing representation technique α-shapes. It is parameterized by a value α. Depending on α the representation ranges from the convex hull of the point set, over multiple, detailed polygons, to no polygons at all. Typically, one chooses an α value that still reflects the point set distribution while at the same time simplifying it. We also discuss a modification of the standard α-shapes which produces schematized α-shapes.
- Another visualization of a point set is the spatial density map. Here, we overlay the map with a grid, and all grid cells that contain more data points than a given threshold are colored. Such a visualization can also color-encode the number of points in the grid cells. As output, we receive a simplified and schematized aggregation of the points.
- We also look at the labeling of point sets. Labeling is a standard cartography technique to display additional data information. Therefore, we place an icon (e.g., a symbol) on the map over the data point. To achieve good legibility, only a selection of labels is displayed.

To guarantee a pleasant interaction for the user, the visualization must be displayed in real-time. Especially for large data sets this is a challenge. In this work, we develop data structures that guarantee a fast response time. Such data structures are called *time-windowed data structures*. As a general idea, we break down the visualizations into their atomic geometric elements. Then, we pre-process the set of all time-window queries for which each atomic geometric element is displayed. Furthermore, we also look at consistency criteria between two successive time-window queries. Especially, when the interaction is implemented in the user interface with a slider (time-window slider), an interaction with the time-window slider should not lead to flickering effects in the visualization.

In the second part of the thesis, we consider a static case where we have a complex spatial geometry without any temporal information as input. While displaying this complex geometry with all its details can be needed for a thorough analysis, it can be unclear and overwhelming for a user who wants a high-level overview. For example, think of a very detailed border of a country. Often, such a border is used as an underlying base map to give the user a spatial orientation. To not distract the reader from the data that lies over the base map, the base map should not be too detailed. Hence, simplifying the border can be necessary for clear and readable visualizations. We simplify polygons by hulls, i.e., a polygon that contains the input polygon.

Zusammenfassung

Die Visualisierung räumlicher Daten ist ein wichtiges Forschungsgebiet der Geoinformatik. Eine Vielzahl der heutzutage erzeugten Daten haben einen räumlichen Bezug, da Sensoren zur Positionsbestimmung (z.B. mittels GPS) weit verbreitet sind. Ein Beispiel für räumliche Daten sind Vogelbeobachtungen: Ein Datenpunkt entspricht der Position und dem Zeitpunkt des beobachteten Vogels und ggf. zusätzlichen Informationen wie Fotos oder Angaben zur Spezies. Räumliche Datensätze sind oftmals groß und komplex. Die Visualisierung der Daten ermöglicht es, die Daten zu untersuchen und zu analysieren. In dieser Arbeit werden Methoden entwickelt, die solche Visualisierungen unterstützen. Dafür werden Visualisierungstechniken aus der Kartographie mit Methoden aus der theoretischen Informatik kombiniert, um z.B. die Berechnungszeit einer Visualisierung zu verringern.

Diese Arbeit besteht aus zwei Teilen. Im ersten Teil werden raum-zeitliche Daten betrachtet. Hier entspricht jeder Datenpunkt einem Punkt in Raum und Zeit. Für solche raum-zeitliche Daten werden oftmals interaktive Visualisierungen verwendet, wo Nutzer die Daten mittels Zeitfenstern filtern können. Die Daten, welche in dem angefragten Zeitfenster liegen, werden auf einer Karte visualisiert. Da solch eine Visualisierung unübersichtlich werden kann, werden in der Kartographie häufig Techniken verwendet, welche die Übersichtlichkeit erhalten. In dieser Arbeit werden folgende Techniken diskutiert:

- In der Kartographie wird eine Punktmenge häufig durch ein oder mehrere Polygone repräsentiert. Dies kann es erleichtern räumliche Muster zu erkennen und die visuelle Darstellung zu vereinfachen. Ein verbreiteter Ansatz für die Repräsentation mit Polygonen sind α-Shapes. Abhängig von der Wahl des Parameters α entspricht das α-Shape der konvexen Hülle der Punktmenge, oder mehreren detaillierten Polygonen, oder der Eingabepunktmenge. Üblicherweise wählt man einen α-Wert, der die Verteilung der Punktmenge widerspiegelt und sie gleichzeitig vereinfacht. In dieser Arbeit wird eine Modifikation der α-Shapes eingeführt, welche schematisierte α-Shapes erzeugt.
- Eine weitere Visualisierung einer Punktmenge ist die Spatial Density Map. Dazu wird die Karte mit einem Gitter überlagert, und alle Gitterzellen, die mehr Datenpunkte enthalten als ein festgelegter Schwellenwert, werden eingefärbt. Eine solche Visualisierung kann auch die Anzahl der Punkte in den Gitterzellen farblich kodieren. Eine Spatial Density Map ist eine vereinfachte und schematisierte Aggregation der Punkte.
- Eine weitere Standardtechnik der Kartographie für die Darstellung von zusätzlichen Dateninformationen basiert darauf, dass Symbole oder Beschriftungen in der Karte über eine Auswahl der Punkte aus der Punktmenge platziert werden. Die Symbole können beispielsweise Icons sein, welche den Typ des Datenpunktes beschreiben, oder die Beschriftung kann dem Namen des Datenpunktes entsprechen. Um eine gute Lesbarkeit zu garantieren und überlappende Symbole oder Beschriftungen zu vermeiden, wird nur eine Auswahl der Symbole oder Beschriftungen angezeigt.

Um eine angenehme Interaktion für den Nutzer zu gewährleisten, sollte die Visualisierung in Echtzeit dargestellt werden. Besonders bei großen Datenmengen ist dies eine Herausforderung und On-Demand-Berechnungen reichen oftmals nicht aus. In dieser Arbeit werden Datenstrukturen entwickelt, die eine schnelle Visualisierung ermöglichen. Solche Datenstrukturen für Zeitfensterfilterung werden in der Literatur als *time-windowed data structures* bezeichnet. Für diese Datenstrukturen werden die Visualisierungen in ihre atomaren geometrischen Elemente zerlegt. Für diese atomaren geometrischen Elementen werden alle Zeitfenster bestimmt, für die das Element dargestellt wird. Darüber hinaus betrachten wir auch Konsistenzkriterien zwischen aufeinanderfolgenden Zeitfensterfilterungen. Insbesondere wenn die Interaktion in der Benutzeroberfläche mit einem Schieberegler (time-window slider) implementiert ist, sollte eine Interaktion nicht zu Flackereffekten in der Visualisierung führen.

Im zweiten Teil der Arbeit wird das Szenario betrachtet, bei dem komplexe räumliche Daten ohne zeitliche Information vorliegen, welche visualisiert werden sollen. Während die Darstellung dieser komplexen Geometrie mit all ihren Details für eine gründliche Analyse erforderlich sein kann, kann sie für Nutzer, welche sich einen Überblick verschaffen möchten, unübersichtlich und überwältigend sein. Ein Beispiel ist eine sehr detaillierte Landesgrenze, welche in Basiskarten verwendet wird, um Nutzern eine räumliche Orientierung zu geben. Um die Nutzer nicht von den dargestellten Daten abzulenken, sollte die Basiskarte nicht zu kleinteilig sein. Daher kann eine Vereinfachung der Grenze notwendig sein, um eine klare und lesbare Visualisierung zu erreichen. In diesem Teil der Arbeit werden polygonale Hüllen diskutiert, welche komplexe Eingabepolygone vereinfachen.

Contents

1 Introduction

In today's world, we have to deal with a huge amount of data. Due to inexpensive and widespread positioning sensors, a large part of the data has spatial information. For example, each smartphone, camera, navigation device, or car generates such spatial data. Also in research, spatial data is essential, e.g., agricultural-related observations, weather measurements, and sensor observations in robotics. With a large amount of data, it is essential for users to get a good overview of the data and data patterns. Hence, it is essential to provide tools for data exploration.

In the following, we describe two scenarios where data exploration tools are needed. First, we look at the social media platform Flickr. This platform allows users to share their images and make them publicly available. Hence, this generates a large database of images where a lot of images are annotated with their location. Users can also look at, or download images that were generated by others. To allow users to find images that are relevant to them, a data exploration tool is key. Further, one can also gain knowledge from data patterns, e.g., city districts with a large number of images might be touristic districts. Hence, data exploration tools improve the usability of and knowledge about the data. As a second scenario, take a research project with many stakeholders in the field of agriculture. Typically, such research projects generate large amounts of spatial data at one or multiple sites. Often, data sets generated by one researcher are of interest to other researchers as well. Particularly, data sets that are located in the same field are interesting. To support researchers in using the data from others, it is necessary to provide appropriate data exploration tools that also include spatial closeness.

From a theoretical point of view, the concept of data exploration is a part of data analysis. Important aims of data exploration are providing data overviews, improving data comprehension, and illustrating data characteristics. For spatial data, it is key to provide information on the spatial data extent. As already stated by Henrik Ibsen *"A picture is worth a thousand words"*, an important branch of data exploration is data visualization. For spatial data, illustrating the data on a map is an obvious and effective approach. Here, the research fields of data exploration, data visualization, and cartography overlap.

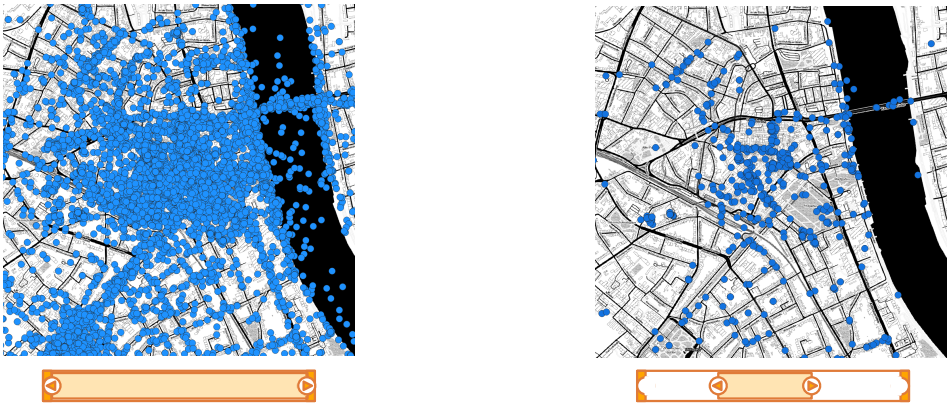
The research field of cartography widely explored the visualization of spatial data. Amongst others, design goals for good visualizations have been developed. A typical cartographic design goal is to balance the information amount while maintaining a clear and comprehensible map. This design goal is especially important for visualization in the context of data exploration. Here, we deal with a large amount of data that can easily lead to overwhelming visualizations. In cartography, researchers developed approaches that aggregate, simplify, schematize, and/or select data for visualization.

In this work, we use such existing visualization techniques and design goals from cartography and enhance them with knowledge from computer science. We partition the thesis into two parts. At first, we consider the visualization of *spatiotemporal events*, i.e., spatial data that is annotated with temporal information. Secondly, we discuss visualizations for the simplification and schematization of complex spatial data that has no temporal information. In the following, we give a more detailed discussion of these two types of data and introduce our approaches to enhance the visualizations.

Exploration of Spatiotemporal Events

A spatiotemporal event is a pair of a spatial geometric object and a timestamp that can be additionally annotated with other information. In this thesis, we focus on events where the geometric object is a point in space. An example of such event sets is the image databases from the social media platform Flickr where images are annotated with a location and a timestamp. Additionally, the images can have titles, descriptions, and user ratings.

For the visual exploration of spatiotemporal events, it is important to consider both, the spatial and the temporal dimensions. Interactive user interfaces are especially well-suited for visual exploration. Figure 1.1 illustrates our interactive user interface. It consists of two components: a map at the top and an interaction panel at the bottom. As an interaction, we allow the user to filter the data for time windows. The interaction panel consists of a timeline and a time window on top. We use a dynamic range slider for the interaction, i.e., the user can slide the endpoints or the whole time window along the timeline. We call the interaction method *time slider*. For a queried time window, the data that lies in this time window is displayed on the map. Note that by continuously sliding the time window a user can also reveal spatiotemporal data patterns. It is key for a pleasant user interaction that the visualization is displayed in real-time. Further, the visualization should be



(a) all events

(b) events with timestamps in year 2017

Figure 1.1: Each point in the maps corresponds to an image taken in the city of Bonn, Germany. Underneath the map, we display the time-slider interface (orange). *Data retrieved from Flickr. Map tiles by Stamen Design, under CC BY 3.0. Map data by OpenStreetMap, under ODbL.*

consistent, i.e., flickering should be avoided. Next, we summarize the challenges that arise with a time-slider interface for visual event exploration.

Challenge (Exploration of Events)**.** *The main challenges for the visual exploration of spatiotemporal events with a time-slider interface are:*

C1 The spatial visualization should be precise and accurate.

C2 The spatial visualization needs to be clear and comprehensible.

C3 If the events have additional information, we need tools to display them.

C4 The response time needs to be small to handle real-time interactions.

C5 When moving the time-slider the changes in the spatial visualization should not be distracting.

C6 The system needs to be capable of handling large data sets.

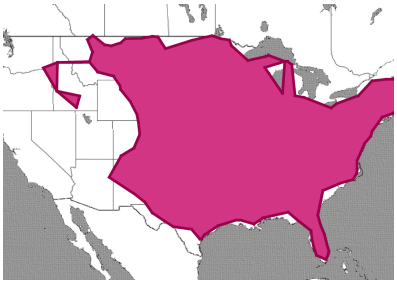
We want to note that the challenges are not sorted by priority or difficulty.

From cartography, there exist several visualization techniques that face challenges C1-C3. These techniques aggregate, simplify, schematize, and/or select parts of the data for the visualization. In the following, we give an overview of visualizations that are used in this thesis.

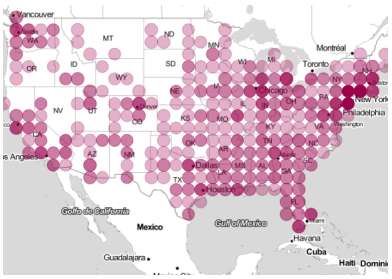
- A common technique in cartography for visualizing a set of points is to aggregate them into one or multiple polygons; see Figure 1.2a. The aim is to use fewer vertices for the polygonal representation such that the visualization simplifies the geometry while still reflecting the point set shape. Hence, a balance between challenges C1 and C2 is achieved. A variety of simple to more refined visualization techniques exist, such as the convex hull, the α-shape [Edelsbrunner et al., 1982, Edelsbrunner, 2010], and the characteristic shape [Duckham et al., 2008]. In this work, we use α-shapes. They are parameterized by a value α. Depending on the value of α, the result can range from convex hulls to polygons that tightly fit the data. To make the visualization even clearer, we also provide a schematized version of α-shapes where the orientation of each line segment of a polygon stems from a pre-defined set of orientations (e.g., horizontal, vertical, and diagonal direction).

- Another common approach for the visualization of point sets is a grid-based spatial density map; see Figure 1.2b. Here, a grid is placed over the map and a grid cell is colored according to the number of points that are contained in this grid cell. One can either use a binary color encoding with one threshold or a color scale with multiple thresholds. Besides this classical version of spatial density maps, also more elaborate information can be depicted. For example, when each point has a weight, we want to color a grid cell according to the sum of the weights of the points that lie in the grid cell.

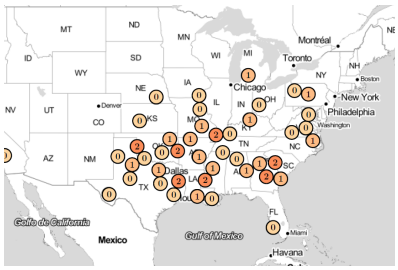
- When we want to visualize additional information as described in challenge C3, a typical approach in cartography is to place symbols (icons, diagrams, etc.) or text labels at the spatial location of the point on



(a) α-shape for storm events in year 1991



(b) density map for Covid19 spread in April 2020



(c) labeling for tornado occurrences between 20th of March 2017 and 20th of June 2017.

Figure 1.2: Different kinds of data visualizations. *Data retrieved from Data.gov. Map tiles by Stamen Design, under CC BY 3.0. Map data by OpenStreetMap, under ODbL.*

the map; see Figure 1.2c. A common problem of such a visualization is the limited amount of space on a map and the symbols or labels overlap. Therefore, only a selection of the symbols or labels is displayed. Typically, the aim is to obtain an overlapping free placement of the symbols or labels [Yoeli, 1972]. In literature, this problem is called point selection or map labeling problem, respectively.

All of these visualizations are well-studied for the static case. In Chapter 2, we discuss existing work from cartography on static visualization techniques in more detail. Hence, challenges C1-C3 were already broadly discussed in cartography.

Nevertheless, an interactive application scenario opens up new challenges (C4-C6). In particular, for large data sets, a real-time query response is needed for a smooth and enjoyable data exploration. Further, it is desirable to have stable or in other terms consistent visualizations for small changes in the queried time window. In this thesis, we tackle these challenges by introducing data structures that enable real-time interactions. Bannister et al. [2013] introduced such data structures for problems from computational geometry and relational event graphs and called them *time-windowed data structures*. In Chapter 3, we will give a more refined discussion on existing work from computer science on algorithms and data structures for time-windowed data structures. The general idea in this thesis is that we break down the visualizations into their atomic geometric objects. The α-shape consists of a set of edges, the spatial density map of a set of grid cells, and the labeling of a set of labels. The data structures enrich each atomic geometric object with the sets of time-window queries for which the geometry is contained in the visualization. Hence, whenever a user queries for a time window, we can directly check the atomic geometric objects instead of going through the whole event data set. For the α-shape visualization, we call our data structure α-structure (see Chapter 5). For spatial-density maps, we introduce the θ-structure (see Chapter 6). In Chapter 7, we present the λ-structure for real-time and consistent map labeling.

Exploration of Spatial Data

The second part of this thesis focuses on the visual exploration of complex spatial data. Here, we consider a polygon with a complex boundary as input. The challenge in visualizing such a complex object is to balance the clearness and the precision. One concept to achieve such a clear visualization is simplification. Typically, a simplified polygon has fewer corner vertices than the original polygon. A simple approach that simplifies a polygon is the convex hull of the polygon. Another concept from cartography to achieve a clear visualization is schematization. The edge directions of a schematized polygon are restricted to a pre-defined set of directions, e.g., for rectilinear polygons, the direction of each edge is either horizontal or vertical. A simple example of a schematization of a polygon is the bounding box. Since the convex hulls and bounding box do not mimic the shape of a polygon well, more sophisticated methods are needed.

In this work, we contribute a model of optimal polygonal hulls that simplify the input polygon and polynomial-time algorithms; see Chapter 11. We call these hulls *shortcut hulls*. We require that each line segment of the hull starts and ends at a vertex of the input polygon. In order to find a good balance between simplification and precision, we introduce a cost function that balances the covered area and the perimeter of the hull. We prove that an optimal shortcut hull with respect to the cost function can be computed in polynomial time. If we forbid holes in the shortcut hull, the problem admits a straightforward solution via shortest-path computation. For the more challenging case in which the shortcut hull may contain holes, we present a polynomial-time algorithm that is based on computing a constrained, weighted triangulation of the input polygon's exterior.

Contribution

We see the main contribution of this thesis in advancing the following open problems motivated by cartography with techniques from computer science.

- Real-time visual exploration of event data with time-window data structures.
- Simplification of polygons with hulls in polynomial time.

Motivated by the real-world problem of big data exploration, both research areas are of high practical relevance. For example, the research on time-windowed data structures allows users to explore such data smoothly and in real-time. But also from a theoretical point of view, the underlying geometric problems are highly interesting.

Outline

This thesis is structured in two parts. In the first part, we look at time-windowed data structures for visual data exploration. This part starts with a discussion of related work from the field of cartography; see Chapter 2. This comprises an overview of relevant research fields and specific visualization techniques. After that, we provide an overview of relevant methodologies from computer science; see Chapter 3. Then, we formalize the problem of time-windowed data structures and introduce concepts that are relevant for our three time-windowed data structures; see Chapter 4. Following these two sections, we discuss our time-windowed data structures: the α-structure (Chapter 5), the θ-structure (Chapter 6), and the λ-structure (Chapter 7). We conclude our first part and discuss open questions in Chapter 8.

In the second part of this thesis, we look at hulls of polygons for visual data exploration. We start this part with a discussion of related work on simplification and schematization of geometries from computational geometry and geoinformation; see Chapter 9. Then, we formalize the problem and introduce relevant concepts and notations; see Chapter 10. In Chapter 11, we introduce shortcut hulls that simplify the input polygon. At last, in Chapter 12, we conclude the second part by giving a short insight into ongoing research and an outlook on remaining open and interesting research questions.

Part I

Time-Windowed Data Structures

2 Part I: Related Work from Cartography

Cartography is the art and science of maps. In this work, we look at the generation of maps from a computer science point of view. In particular, the automatic generation of spatial data visualizations is essential with today's data volumes. Interactive visualizations also require real-time computation. In the following, we provide an overview on broader research branches from cartography that are related to the challenge of real-time visualization of spatial data. Afterward, we discuss more specific techniques and methods in the sections visualizing point sets, visualizing information of point sets, visualizing spatiotemporal data sets, and dynamic query interfaces.

Geovisualization Information visualization is a well-established research branch that deals with visualization approaches, design goals, and also their evaluation. Such information can be anything; research findings, socioeconomic statistics, and teaching material. When having data with spatial information, the subfield of research is called *geovisualization*. It comprises work on visualization approaches, discussions of the design space, and algorithms for creating visualizations for spatial data. Mostly such visualizations offer a high level of interaction [Williams et al., 2013]. Often, geovisualization approaches are well-suited for data exploration.

Geovisual Analytics Cook and Thomas [2005] introduced the research field visual analytics that focuses on how to support data processing by users with digital visualizations. One of its main areas is interactive visual interfaces for supporting users in the analytical process [Andrienko et al., 2010]. The subfield of visual analytics that focuses on spatial data is called geovisual analytics [Andrienko et al., 2007].

Interactive Cartography With digitization, also for cartography, a wide field of possibilities opens up. Interactive cartography focuses on enabling users to interactively manipulate maps. We want to point out the difference between *interactive* and *dynamic* maps. While for dynamic maps the change is given by the system, e.g., animations, a user can actually interact and change an interactive map. The research field *interactive cartography* focuses on the design and model of such user interactions. As claimed by the visual information-seeking mantra by Shneiderman [2003] such interactions should be designed to give the user first an overview, then allow zooming and filtering, and at last, give further details-on-demand. An everyday example of a spatial visualization system that offers user interaction is Google Maps. First, the users get a basic map as an overview, then, they can search, pan, zoom, and get further details on demand. For a detailed survey on interactive cartography, we refer to Roth [2013] and Crampton [2002]. While traditionally interactive cartography focuses on the scenario of a map on a screen, recent research also takes the opportunities of augmented and virtual reality into account. Here, we want to refer to the new field *immersive analytics* [Marriott et al., 2018, Fonnet and Prié, 2021].

Visualizing Point Sets

Visualizing a set of points in a clear and clutter-free way is a widely explored field of research. A common approach is to use a (visual) representation instead of the actual point set. A *representation* can be any set of spatial objects that reflects the characteristic of the input point set, e.g., a polygonal hull of the point set. We focus on work that aims at representing the spatial extent or the spatial distribution of the point set. For these aims, two major research directions exist. One way is to represent the point set by one or more polygonal shapes which (approximately) include the point set. Simple examples of polygonal representations are the bounding box or convex hull of the point set; see Figure 2.1a and Figure 2.1b, respectively. A problem with these simple approaches is that large areas of the map might be covered although no data points are contained which might lead to a bad assessment of the point set extent. Hence, a series of more sophisticated methods were developed. As a second visualization approach, besides the polygonal representations, there exist also grid-based visualizations. Here, the map is discretized with a grid and the shape of the point set is reflected by the coloring of the grid cells. Existing approaches explore different types of grids, various colorings, and different approaches for the coloring constraints [Silverman, 1986, Liu et al., 2013, Battersby et al., 2017].

Often it is desirable to have a schematized representation, i.e., the orientations of shape borders stem from a pre-defined set of directions, e.g., horizontal, vertical, or diagonal directions. While the shape of grid-based

methods naturally is a schematic representation, most of the polygonal representations are not schematic but sometimes can be tweaked to a schematic version.

In the following, we first describe the Delaunay triangulation which is a basic concept in graph theory and geometry and which is used in many approaches of point set representation. After that, we describe a selection of well-established point set representations. We illustrate these in Figure 2.1 and summarize their properties in Table 2.1. We do not want to claim that any of these approaches outperforms the other but are rather suited for different application scenarios. Subsequently, let P be the set of input points and let n be the size of P.

Delaunay Triangulation A triangulation of a set of points P is a set of vertices, edges, and triangles, s.t., the vertices correspond to P and the number of edges is maximal and crossing-free. The Delaunay triangulation is a triangulation where the circumcircle of every triangle is empty with respect to points from P. The Delaunay triangulation of a set of points can be computed in $O(n \log n)$ time.

Convex Hull The convex hull of a set of points is the smallest convex polygon that contains all points. It is a widely explored geometric concept and Chan [1996] provides an algorithm for computing the convex hull in $(n \log h)$ running time where n is the size of the input point set and h is the number of vertices of the convex hull. The convex hull is closely related to the Delaunay triangulation, i.e., it is the outer boundary of the area covered by the Delaunay triangulation. The convex hull is a simple approach for representing a point set but often it is not appropriate since it can cover large areas where no points of the input point set lie.

α-Shape The α-shape [Edelsbrunner, 2010, Edelsbrunner et al., 1983] is a representation of a set of points in the plane by one or more polygons; see Figure 2.1c. It is a generalization of the convex hull and it is strongly related to the Delaunay triangulation of a point set. Let $\alpha > 0$. The *α-shape* consists of all polygons that are formed by edges starting and ending at points of the input point set, where the edges are not longer than α, and where an open disc with radius $\alpha/2$ whose boundary intersects the start and end point of the edge and has its center on the right side of the edge does not contain any other point of the input point set. On the one hand, when choosing the parameter α large enough, the α-shape is equal to the convex hull. On the other hand, when choosing α small enough the α-shape is equal to the input point set. Using the Delaunay triangulation, an α-shape of a set of n points can be computed in $O(n \log n)$ time.

In this thesis, we use α-shapes for our data structure α-structure; see Chapter 5. A benefit of the α-parametrization is that we can offer a high variety of output visualizations. Also, we introduce a technique for a schematized version of α-shapes that restricts the edges of the resulting polygons to predefined orientations. Hence, we can provide also schematized polygonal point set representations. From a technical point of view, the α-shapes are well suited as a basis for time-windowed data structure, as they are defined very locally, i.e., an edge between two points of the input point set is part of the α-shape if there is an empty disc with radius $\alpha/2$ that intersects the edges' endpoints.

r-Shape The r-shape is an approach closely related to α-shapes. The idea is to center a disc of radius r on each point of the input set; see Figure 2.1d. Then an edge is introduced between each pair of points where the corresponding discs intersect and parts of the discs' boundaries are part of the boundary of the union of all discs. In the end, the boundary of the r-shape is obtained by combining all edges [Chaudhuri et al., 1997, Attali, 1998]. The r-shape can be computed in $O(n)$ time.

Characteristic Shape Another method based on the Delaunay triangulation is the *characteristic shape* [Duckham et al., 2008]; see Figure 2.1e. It is defined by a procedure that starts with the edges of the Delaunay triangulation of the input point set. We iteratively check whether after removing the longest edge that is part of the boundary is still a simple polygon. If this is true, we remove the edge, otherwise, we skip this edge. We repeat this procedure until the longest boundary edge is longer than a chosen threshold. This can be done in $O(n \log n)$ time. The resulting boundary is a simple polygon that contains all points of the input set.

Vernacular Regions Based on Shortest-Path Graphs de Berg et al. [2011] introduce *vernacular regions based on shortest-path graphs*; see Figure 2.1f. They represent the input set of points with a single polygon. More in detail, the polygon is defined as the outline of the shortest-path graph, which is again a subgraph of the Delaunay triangulation. The construction takes $O(n \log n)$ time in case the holes are ignored.

O-hull In the field of schematized shapes, we first mention the O-hull, which is closely related to the concept of convex hulls, but the edge orientations are restricted to a certain set of directions [Fink and Wood, 2004, Alegría et al., 2014]; see Figure 2.1g. Previous research discussed the variant where the basic orientation of the map is given, which can be computed in $O(n \log n)$ time where n is the number of points in the input data set. Another variant also optimizes the orientation of the basis map with respect to the area coverage of the O-hull. This second version can be constructed in $O(kn \log n)$ time where k is the number of permitted edge orientations. However, the O-hull is not suitable for our purposes since it can cover large areas that contain no data points (similar to the convex hull). Hence, the O-hull does not guarantee an appropriate representation of the shape characteristics of the input data.

s-Shape Given a set of points and a regular grid consisting of rectangular cells with width and height s, the s-shape is defined as the outer boundary of all grid cells that contain a point. The s-shape is a schematic polygon, consisting only of rectilinear edges, i.e., edges with either horizontal or vertical orientation [Chaudhuri et al., 1997]; see Figure 2.1h. An s-shape can be seen as a hybrid between a grid-based and polygonal representation. The s-shape of a point set can be computed in $O(n)$ time.

Grid-Based Spatial Density Maps A very common approach for visualizing spatial information is a spatial density map. Given a set of points and a grid, the *grid-based spatial density map* consists of all grid cells that contain at least a certain number of data points. Besides classic regular and rectilinear grids; see Figure 2.1i, for example, Carr et al. [1992] also use a regular hexagonal grid.

The very basic version of a spatial density map simply colors all grid cells that contain more points than a pre-defined threshold. More refined are versions that use multiple thresholds and hence, multiple colors, such that the visualization contains more information on the point set distribution. Often, spatial density maps are also used to illustrate additional information about the input point set. For example, let each point correspond to the location of an image on a social media platform (e.g., Flickr). Then, there often also exists a rating for the images. For the spatial density map, we could think of coloring all cells according to the mean rating over all images. For such an application scenario, also a continuous color palette can be applied.

In between different communities, the term spatial density map is not clearly defined, e.g., Chukwuma et al. [2019] call a choropleth map a spatial density map, Kiyosugi et al. [2010] call an isochrone-like visualization a spatial density map and on the other hand, Rafael et al. [2021] call a spatial density map a clustered heatmap. Hence to be precise, in this work, we call the concept a grid-based spatial density map.

In this thesis, we enhance the spatial density map visualization with our data structure θ-structure; see Chapter 6. In order to complement the polygonal representation given by α-shapes and our α-structure, spatial density maps are a natural choice for a grid-based method. In particular, it is a widely used, easily understandable, and versatile visualization.

Continuous and Binned Heatmap Heatmaps of a set of points encode the point density by color [Battersby et al., 2017]. The density is often derived from a *kernel density estimation* [Silverman, 1986] where the idea is to model the density such that every data point has an influence on its surrounding region. We call the visualization where we color every point in the plane according to a continuous density distribution with a continuous color scheme, a *continuous heatmap*; see Figure 2.1j. However, in the real world, more often the plane is discretized by a grid and each grid cell is colored according to the density distribution. Such a visualization is called a *binned heatmap*; see Figure 2.1k. More in detail, we partition the map with a regular grid into cells that aggregate the contained points. In the literature, this technique is called spatial binning and each grid cell is called a bin. For a detailed discussion, we refer to the work by Battersby et al. [2017], who particularly argue that spatial binning has perceptual advantages for our purposes of counting points and summing up their weights. Liu et al. [2013] provide work on binned heatmaps for big data by introducing data cubes.

Table 2.1: Techniques for the visualization of point sets from cartography. We classify the representation type (P = polygonal, G = grid-based, C = continuous); the number of components in the output (1 = a single component, * = arbitrary many components), whether the resulting visualization can have holes (Y = yes, N = no), and whether the representation is a schematization (Y = yes, N = no), i.e., the edge directions are limited to a predefined set of directions. We give the running times for computing the hulls with respect to the number of input points n and for the convex hull, with respect to the number h of vertices of the convex hull.

technique	repr.	comp.	holes	schematic	asymptotic computation time	examples from literature
convex hull	P	1	N	N	$O(n \log h)$	Chan [1996]
α-shape	P	*	Y	N	$O(n \log n)$	Edelsbrunner [2010], Edelsbrunner et al. [1983]
r-shape	P	*	Y	N	$O(n)$	Chaudhuri et al. [1997], Attali [1998]
characteristic-shape	P	1	N	N	$O(n \log n)$	Duckham et al. [2008]
vernacular regions	P	1	Y	N	$O(n \log n)$	de Berg et al. [2011]
O-hull	P	1	Y	Y	$O(n \log n)$	Fink and Wood [2004], Alegría et al. [2014]
s-shape	P/G	*	Y	Y	$O(n)$	Chaudhuri et al. [1997]
grid-based spatial density maps	G	*	Y	Y	$O(n)$ for rectangular grids Eppstein et al. [2015]	Eppstein et al. [2015]
continuous heatmap	C	*	Y	N	depends on density-distribution	Battersby et al. [2017]
binned heatmap	G	*	Y	Y	depends on density-distribution	Liu et al. [2013]

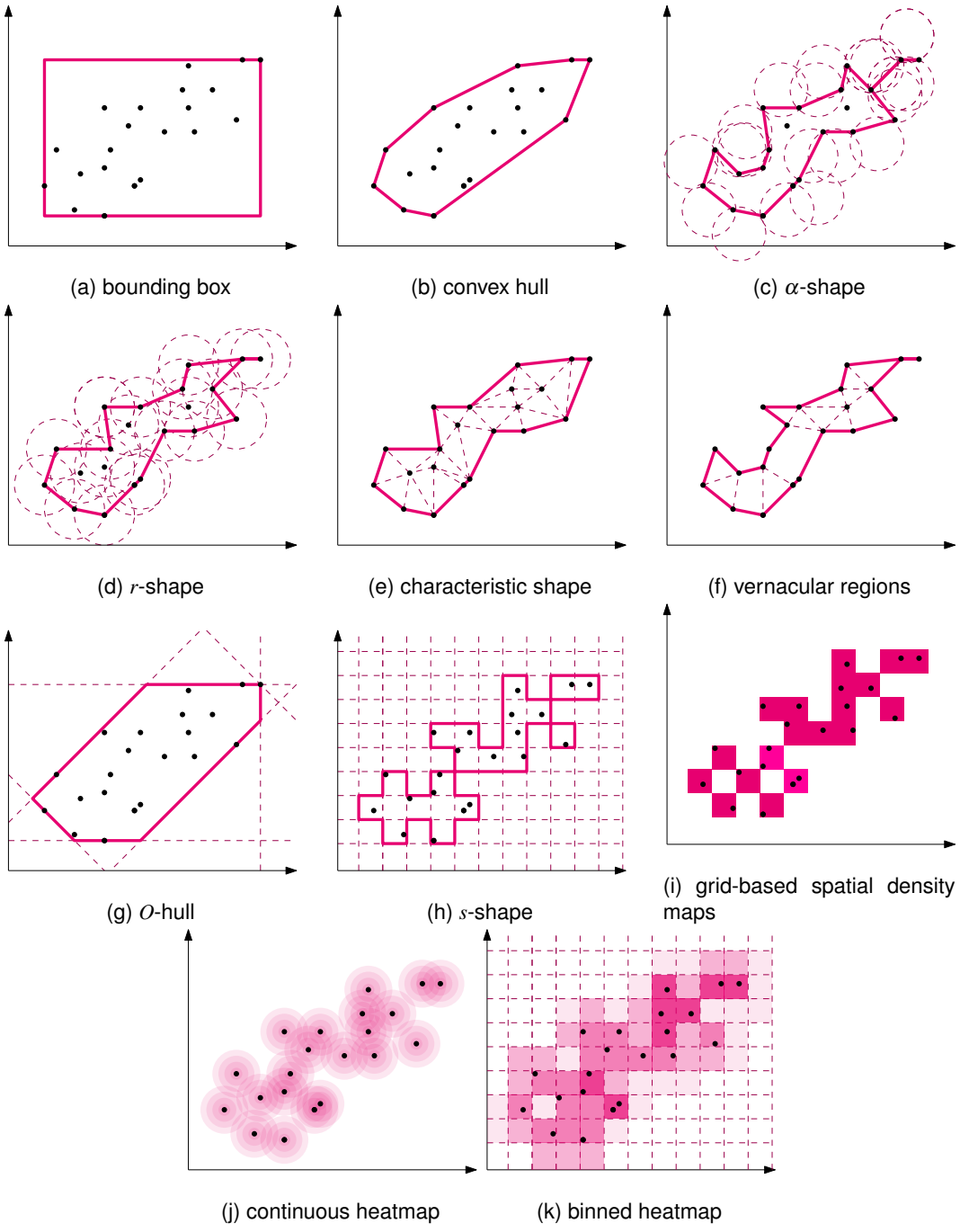


Figure 2.1: Representing a point set. The dashed lines are not part of the visualization, but they are part of the model of the representation. E.g, the α-shapes are defined by the shown set of empty discs.

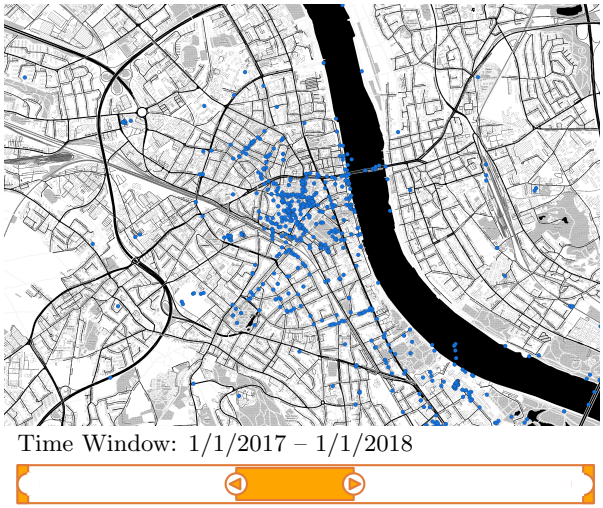


Figure 2.2: Time-slider interface for a data set where each point in the maps corresponds to an image taken in the city of Bonn, Germany in the time range 1st of January 2017 until 1st of January 2018. *Data retrieved from Flickr. Map tiles by Stamen Design, under CC BY 3.0. Map data by OpenStreetMap, under ODbL.*

Visualizing Information of Point Sets

The previously discussed representation of point sets focuses on visualizing the extent and geometric characteristics of a point set. However, for some application scenarios, it can be important to visualize additional information that is associated with the point data. For example, given a set of social media images at a different location and time, see Figure 2.2, it can be interesting to visualize a subset of the images at their location on the map. Another example are animal observations where we want to place an icon that encodes the species of the animal in the map. Also, text can be placed on a map, e.g., the names of restaurants and sights. Often, such additional information (photos, icons, text, etc.) associated with a point is visualized by placing it over the point or close to the point on the map. For real-world data, it is mostly not possible to display all icons or texts in a clear and comprehensive way on the map, e.g., due to overlaps. A standard procedure to solve this problem is to visualize only a selection of the additional information. In cartography, the problem of selecting icons in the map is called a *point selection problem*. It is part of the generalization operations defined by Hake et al. [2002]. On the other hand, placing text on the map is called *map labeling*. Throughout this thesis, we will refer to the presented problem as a map labeling problem but we want to highlight that it is equivalent to the point selection problem when treating each point as a label.

Map labeling is widely investigated in theoretical computer science and there exist plenty of algorithms with respect to optimizing the visualization. In this thesis, we assume that we are given the labels' placement and geometry while we optimize the selection of displayed labels. For the static (non-animated and non-interactive) case, a common goal is to maximize the number of displayed labels while avoiding overlapping labels [Agarwal et al., 1998, Yoeli, 1972, Haunert and Wolff, 2017]. It has been shown that these problems are often NP-hard since finding a maximal independent set in the conflict graph of the labels corresponds to selecting a maximal non-intersecting subset of the labels.

In this thesis, we discuss map labeling for time-slider interaction. While we also want to optimize the sum of displayed labels the visualization should be stable during the time-slider interaction. In Chapter 6, we introduce the λ-structure that offers a solution to this problem.

Visualizing Spatiotemporal Data Sets

Previously, we presented visualizations from cartography that focus on spatial data without temporal information. In the following, we discuss work on visualizing spatiotemporal data from the research branches geovisual analytics [Keim et al., 2008, Andrienko et al., 2010, Sun et al., 2013], geovisualization [Kraak, 2003], and interactive cartography [Williams et al., 2013]. For a detailed survey on the exploration of spatiotemporal data, we refer to Andrienko et al. [2003].

It is noticeable that a special focus in the area of spatiotemporal data visualization lies in the visualization of movement data. In this context, among others, flow maps [Phan et al., 2005, Buchin et al., 2011], density

maps for trajectories [Scheepens et al., 2011], as well as space-time cubes [Andrienko et al., 2013, 2014] have been considered. Applications include the analysis of spatial mobility derived from social media data [Wu et al., 2016] as well as the investigation of animal behavior [Slingsby and van Loon, 2016, Spretke et al., 2011]. Particular visualization systems were developed by Ferreira et al. [2011] for the analysis of bird populations and by Konzack et al. [2019] for the exploration of migration patterns in gull data.

In this thesis, we focus on the visualization of spatiotemporal events that are points in space and time. Although movement data also consists of sequences of points in space and time, our visualization faces different challenges.

Dynamic Query Interface

At last, we want to discuss work from cartography and information visualization with respect to our interaction techniques. In this thesis, we use the established *dynamic query interface* which was introduced by Kapler and Wright [2005]. A dynamic query interface enables the user to graphically define the query and receive a real-time response and continuous animations. It can be used for spatiotemporal data and also for other higher dimensional data sets. Especially the implementation of a dynamic query interface with a time slider is a standard tool for information visualization systems [Kapler and Wright, 2005, Ahlberg and Shneiderman, 2003, Hochheiser and Shneiderman, 2004, Robinson et al., 2017, Andrienko and Andrienko, 1999, Burigat and Chittaro, 2005]. User studies showed that dynamic query interfaces allow significantly smaller task completion time in comparison to paper-printout, text search, and form-fill-in interfaces [Ahlberg et al., 1992]. Due to the required real-time response, the problem of efficiency [Tanin et al., 1996] arises. Additionally, due to the continuous sliding of the query, the challenge of consistent visualization arises. In particular, there is the problem of objects that unnecessarily disappear and appear again during an interaction (flickering). Often the problem of flickering is also faced in animated and interactive map labeling.

In this thesis, we use a dynamic query interface for the visual exploration of the event data with α-shapes, spatial density maps, and map labeling. With our data structures, we tackle the challenge of real-time response time and flickering.

3 Part I: Methodological Background

In this chapter, we introduce data structures from computer science that are used throughout the first part of this thesis. In computer science, a data structure is a particular way of storing and organizing data. For example, for a data set of numbers, we can store them in arrays, lists, heaps, or trees. Depending on the application scenario, some data structures perform better than others. For a scenario where elements are added or removed regularly from the data set, dynamic data structures like lists, trees, or heaps outperform static data structures like arrays. Another typical scenario is searching in the data set. In the context of databases, a data structure is often called an index. Choosing appropriate data structures can enable us to solve experiments with large data sets that otherwise lead to unmanageable, large running times. In this thesis, we develop data structures that support the search for time windows. In the literature, these data structures are called time-windowed data structures [Bannister et al., 2013]. When developing a data structure, one needs to carefully balance the gain in query time while preserving a small storage consumption.

Tree Structures

In this section, we introduce tree structures and different variants. We do not aim at giving a complete survey but rather limit it to structures that are relevant to this thesis. For a fundamental overview, we refer to Cormen et al. [2009].

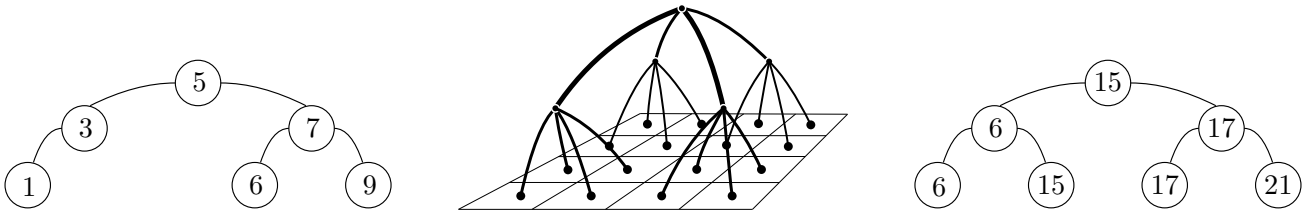
We introduce a tree as a special type of graph. More in detail, let $G = (V,E)$ be a graph where V is the set of vertices and E is the set of edges. The graph G is a *tree* if G is connected, G is acyclic, and G is undirected. Typically, we deal with *rooted* trees, i.e., one vertex of V is the designated root and the others can be ordered hierarchically according to their path length to the root. For ordered rooted trees, every pair of adjacent vertices is in a parent-child relation where the vertex that is contained in the path to the root of the other vertex is the parent of the other vertex. Note that each vertex of a tree can be considered as a root of its own subtree. Vertices that have no children are called leaves. The *height* of a rooted tree is the longest path length between the root and a leaf. We denote the height by h in this section.

Often data structures for indexing data sets are based on tree structures. Such tree structures support searching in the data set. We list typical kinds of search scenarios for using tree structures.

- The simplest kind of query is to report an element if it exists in the given data set. We call such queries *element existence queries*.
- Another application scenario are *range queries*. Here, we are given a data set of k-dimensional points and we query for all elements that are contained in a k-dimensional range. In this thesis, we have such queries, e.g., for spatial queries for the θ-structure; see Chapter 6.
- Similarly, when we are given a data set of polygons a tree structure can support queries for all polygons that contain a query point. Such queries are called *point-in-polygon query*. We use such queries in this thesis, e.g., for the α-structure, the θ-structure, and the λ-structure when searching for all activity regions that contain the time-window query point; see Chapter 5, Chapter 6, and Chapter 7.

In the following, we introduce several variants of tree structures. At first, we introduce a fundamental version of a tree structure: the balanced binary search tree. Afterward, we discuss the quadtree that can be used for the θ-structure. At last, we introduce range trees that we use for the α-structure, θ-structure, and λ-structure; see Chapter 5, Chapter 6, and Chapter 7, respectively.

Binary Tree Probably the most important and widely used type of tree structure is the binary tree, i.e. a tree is a binary tree if every vertex has at most two children. These are called left and right child, respectively. As for each tree, each non-null internal and leaf node also points to its parent node. A widespread type of binary tree is a binary search tree (sorted binary tree/ordered binary tree). A binary search tree allows efficient searching for elements, inserting elements, and deleting elements. Let v be a node in a binary search tree. A binary tree is called a binary search tree if it holds that the key of each node in the subtree rooted at the left child of v is smaller than the key of v. Analogously it holds that the key of each node u in the subtree rooted at the right child of v is larger than the key of v. For a binary search tree, the running time for search, insert, and delete is $O(h)$ where h is the height of the tree. A special form of a binary search tree is a self-balancing binary search tree, e.g., AVL trees [Adelson-Velskii and Landis, 1962]; see Figure 3.1a. For such an AVL tree, it holds that $h \in O(\log n)$ where n is the size of the data set. Then, the search, insertion, and deletion run in $O(\log n)$ time.



(a) AVL tree (self-balancing search tree)

(b) quadtree

(c) 1D range tree

Figure 3.1: Tree data structures.

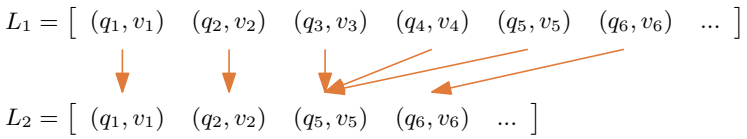


Figure 3.2: Fractional cascading for two arrays L_1 and L_2 of key-value pairs (q_i, v_i) where $L_2 \subset L_1$. The pointers from elements of L_1 to elements of L_2 are illustrated in orange.

Quadtree A quadtree is a tree structure where every internal node has exactly four child nodes. Quadtrees are often used when working with two-dimensional spatial data. In that case, each node corresponds to a square or rectangular region of the map; see Figure 3.1b. We call such a region of a node a cell. The children of a node partition its cell again into four cells. Typically a quadtree is refined until either a certain spatial resolution (minimal cell size) is reached, or the number of spatial elements in the cells is below a predefined certain threshold. Querying a quadtree for all elements that are contained in a two-dimensional range is a typical application scenario for quadtrees. For example, take a digital map where a user can pan the map, zoom in, and zoom out. In order to find all spatial data in the current map extent, a quadtree can support such 2-dimensional range queries. Typically a quadtree has a storage size of $O(nh)$ where n is the size of the data set that needs to be stored and h is the height of the quadtree. A range query can be answered in $O(dh)$ time where d is the number of reported nodes. In this work, we use quadtrees for indexing the grid cells of the density maps for the θ-structure; see Chapter 6.

Range Trees A range tree is another form of a tree structure. They are designed for range queries on points in the k-dimensional space. In fact, in the 1-dimensional case a range tree is a version of a balanced binary search tree; see Figure 3.1c. Range trees were introduced by Bentley [1978] and later Chazelle [1990] improved the memory consumption to $O(n(\frac{\log n}{\log\log n})^{k-1})$ and query time $O(\log^{k-1} n + d)$ where n is the number of elements, k is the dimension, and d is the number of reported elements. In this work, we use range trees for indexing the activity region of the θ-structure; see Chapter 6.

Fractional Cascading

Given a data set and a sequence of subsets of the data set where we perform binary searches for the same element, we can speed up such repeated searches with fractional cascading [Chazelle and Guibas, 1986a,b, de Berg et al., 2008]. In this work, we use fractional cascading to speed up the query time of the θ-structure. In the following, we give a more detailed description. Let $L_1, \ldots, L_J$ be arrays with $L_i \subseteq L_j$ for $1 \leq i \leq j \leq J$ and let q be a key. Then, fractional cascading supports queries for the key q in each array $L_1, \ldots, L_J$. Fractional cascading adds to each element r in L_i a pointer to the first element of L_{i+1} that is equal or larger than r. When searching for q, we perform a binary search in L_1 and therewith we also receive the pointer to the query result in L_1 (i.e., the associated structure of the root) and iteratively for all L_i; see Figure 3.2. An approach without fractional cascading independently searches in each array, which takes $O(d \log n)$ time where n is the number of elements in L_1. With fractional cascading the search takes $O(d + \log n)$ time increasing the space consumption by a constant factor.

Time-Windowed Data Structures

In the following, we introduce time-windowed data structures and showcase the two most important application fields. The concept of *time-windowed data structures* was introduced in 2013 by Bannister et al. [2013]. Our

developed data structures α-structure, θ-structure, and λ-structure all can be classified as time-windowed data structures.

Let P be a data set of elements where each element p is associated with timestamps t. We call a range $[t_S, t_E]$ that starts at timestamp t_S and ends at timestamp t_E a *time-window*. We call a query for a property of all elements with $t \in [t_S, t_E]$ a *time-window query*. A data structure that aims at efficiently answering such time-window queries is called a *time-windowed data structure*. The general idea of such data structures is to pre-process the possibly large data set such that it can be queried with time windows efficiently. Mostly, for visualization purposes, a real-time requirement is enforced.

As a first attempt for a time-windowed data structure, one might want to store the query result for each possible query. Assuming the data set contains n elements, there exist $O(n^2)$ distinct results for time windows. Hence, the resulting storage consumption can be large when storing the results of all time-window queries. To avoid such shortfalls, more elaborate data structures are designed.

The first application field was presented by Bannister et al. [2013] for *relational event graphs*, i.e., a graph where each edge is annotated with a timestamp. Their data structure supports basic counting problems, e.g., on the number of connected components, or the number of isolated vertices. Later, Chanchary and Maheshwari [2019] presented a time-windowed data structure for monotone decision problems such as the bipartiteness and the disconnectedness of relational event graphs. Chanchary et al. [2019] use color range queries on such graphs to answer time-window queries for different graph parameters (e.g., density).

The second application field where time-windowed data structures have already been considered is computational geometry. Bannister et al. [2014] presented data structures for querying the solutions of basic problems such as computing the convex hull of a spatiotemporal point set. Bokal et al. [2015] considered basic monotone decision problems on spatiotemporal point sets, e.g., they can report point sets whose diameter is at most 1 or whose convex hull has an area at most 1. Chan and Pratt [2016] gave improvements for these problems and extended them to similar decision problems. Chan and Pratt [2015] used a quadtree-based approach to report the closest pair for a given time window. Moreover, Chanchary et al. [2018] presented time-windowed data structures for deciding whether there are intersections between geometric objects.

In this thesis, we discuss the visualization of event data sets as a third application field for time-windowed data structures.

Stability Enhancing Data Structures for Visualization

In the context of visualizing spatial data sets for a dynamic or interactive scenario, lately, a lot of effort has been undertaken to provide stable visualizations. This includes stability with regards to map interactions such as zooming, panning, and rotation, but also, when looking at time-series data, stability with regards to time [Nickel et al., 2022].

An extensively studied scenario is stable map labeling. For example, there exists work on stable labelings for animation, where the stability is obtained by additional stability constraints [Barth et al., 2016, Gemsa et al., 2020, Bobák et al., 2020]. For the interactive case, Been et al. [2006] introduced the concept of active ranges for labels, considering zooming, panning, and rotations of the map. This concept was investigated intensively from an algorithmic point of view. Been et al. [2010] proved that active range maximization is NP-hard for zooming and they give constant-factor approximations. Active range maximization for zooming has been similarly considered for further variants [Liao et al., 2016, Schwartges et al., 2013, Zhang et al., 2020, Nöllenburg et al., 2010]. For active range maximization in rotating maps, Gemsa et al. [2016b] presented an NP-hardness proof and proposed approximation algorithms. For the same scenario, Gemsa et al. [2016a] experimentally evaluated approximation algorithms and greedy heuristics with respect to exact solutions obtained by integer linear programming (ILP). Similarly to active range maximization, Funke et al. [2016] and Bahrdt et al. [2017] consider circular and prioritized labels whose radius grows with the scale of the map.

With a similar goal, algorithms for map generalization have been developed that ensure consistency during interactions. This includes methods for the consistent selection of roads from a detailed road data set during continuous zooming [Chimani et al., 2014] or continuous movement of a focus area [van Dijk et al., 2013] as well as methods for the consistent aggregation of areas during continuous zooming [Peng et al., 2020]. A common approach is to pre-compute a data structure from which a map corresponding to a scale or preferences specified by the user can be rapidly retrieved [Meijers et al., 2020].

Another research area with high attention is the stable visualization of time series data. In this scenario, often, the visualization consists of one visualization per time step and it can be presented as an animation or with an interactive user interface. Stability is understood here as a change in the visualization between two consecutive time steps. Lately, such stability was considered for Demers cartograms [Nickel et al., 2022]. Again, the authors pre-process the data set into a data structure that encodes the placement of the symbols per time

step. Closely related is also the field of dynamic graph drawing for event graphs. For example, DynNoSlice is a force-directed algorithm that pre-processes the data set into a data structure that is called a space-time cube [Simonetto et al., 2020]. It can be understood as having a three-dimensional space with the spatial extent in the x-, and y-direction and the time in the z-direction. When slicing the cube for a point in time one receives the graph drawing.

Other Work on Real-Time Interaction with Big Data Visualizations

Besides the previously presented time-windowed data structures, the challenge of real-time interaction with big-data visualizations has been stated in a variety of research branches, and several other problem variants were tackled. Besides the development of data structures for time-window queries, as done in this thesis, there exists a wide range of other approaches, e.g., introducing data structures for other or more general queries (data cubes), improving the server-client infrastructure, and including the prediction of user interaction to prefetch data. Note that in some of these research branches, literature often refers to the query time as latency. In the following, we discuss examples of these approaches.

A data cube is a multidimensional array of values, and the term is typically used in the context of big data where the memory consumption is by far larger than a computer's main memory. In contrast to our work, data cubes aim at providing data aggregation along any set of dimensions. Lins et al. [2013] introduce nanocubes which are an improved version of data cubes with respect to memory consumption. They claim to fit billions of multidimensional data entries into modern laptops' main memory. Nanocubes support the user interaction of aggregating the data set across every possible set of dimensions. Nanocubes can also be understood as a database index structure. Liu et al. [2013] introduce a system imMens that provides the visualizations for queries on multidimensional big data sets in real-time. Their system precomputes multivariate data tiles [Gupta and Mumick, 1999] which are a data structure derived from decomposing data cubes. When users interact via the browser-based frontend, the system visualizes the data set by combining techniques from parallel processing, binned aggregation, and data representation. Similar to nanocubes and imMens are Hashedcubes which were introduced by de Lara Pahins et al. [2017]. Their data structure improves memory with respect to nanocubes and imMens while experiments show a slightly worse query time.

Chan et al. [2008b] focus on optimizing a system infrastructure consisting of database servers, a query distribution server, and the frontend. Their system ATLAS tackles the visualization of massive time series data sets for ad hoc queries. They particularly also consider consistency ("smoothness") for interactions.

ForeCache is a system proposed by Battle et al. [2016] for integrating predictions about the next user interaction into the system. With this prediction, the system prefetches data. Battle et al. show an improvement in query time with respect to non-prefetching and other prefetching systems.

4 Part I: Formalization

In the following, we formalize the input to our data structures, i.e., the set of events. Then, we formalize the visualization by introducing concepts of the time-slider user interface and the interactions. At last, we introduce concepts that are used for the α-structure, θ-structure, and λ-structure.

Input: In Chapter 5, Chapter 6, and Chapter 7, we consider a set of events $E = \{e_1, \ldots, e_n\}$ as input to our problem where each *event* e_i is a pair of a point p_i in the plane and a timestamp t_i when the event occurred. For the θ- and λ-structure, we look at events with a certain weight $w_i \in \mathbb{R}^+$ reflecting the importance of the event. For the λ-structure, additionally, we consider the case that there is some additional information associated with each event that should be visualized, e.g., a description. We assume that the events $e_1, \ldots, e_n$ are ordered by their timestamps, i.e., $t_i \leq t_j$ for $1 \leq i < j \leq n$; see Figure 4.1. An example of such an event set is an image database, where the location and timestamp give information about where and when, respectively, an image has been taken. The number of comments might reflect a weight of an event and the image itself should/can be visualized in the map.

Visualization: Figure 4.2 displays the user interface considered in this thesis. It consists of a map and a time-slider interface as introduced by Andrienko and Andrienko [1999]. In the map, we display the visualization of all events that are queried with the time slider. We call a query $Q = [t', t'']$ triggered by a time-slider interaction a *time-window query*; see Figure 4.3a. We call t' of Q the *query start* and t'' the *query end*. The queries triggered by a time-slider interface are limited by the *left boundary of the time-slider* $t_{\min}$ and the *right boundary of the time-slider* $t_{\max}$. For our time-slider interface, we implement four *basic interactions*; see Figure 4.2:

1. *Panning:* continuous translation of the time window.
2. *Left-Sided Scaling:* continuous change of the left boundary of the time window.
3. *Right-Sided Scaling:* continuous change of the right boundary of the time window.
4. *Uniform Scaling:* continuous change of both boundaries of the time window in opposite directions, such that the center of the time window remains the same.

Problem Formalization: For all of our time-windowed data structures, the priority is to achieve query times that allow real-time interaction with the user interface. From a practical point of view, we define real-time response times based on movie frame rates (24 images per second $\approx$ 40 milliseconds per query). For big data, this can be very challenging. Hence, we relaxed our objective to 200 milliseconds (five frames per second). From a theoretical point of view, we aim for asymptotical running times that are sublinear with respect to the number of events n and low polynomial with respect to the size of the output, i.e., the visual representation for a particular query.

For the λ-structure, we have the additional objective to have small changes when moving the time-slider. We formalize this as follows, when a user performs one of the four basic interactions a symbol should not appear, disappear, and appear again. Such a phenomenon is also called flickering.

Concepts used for our Time-Windowed Data Structures: Now, we introduce concepts that are used in all of the time-windowed data structures of this thesis. At the core of our data structures, each time-window query $Q = [t', t'']$ is represented as a point (t', t'') in the plane; see Figure 4.3a. We call the space where the first axis corresponds to the query start time and the second axis corresponds to the query end time the *activity space*. For brevity depending on the context, we interpret Q either as interval $[t', t'']$ or point (t', t'') in the activity space. Each point representing a time-window query lies in the triangle defined through the left and right boundary of the time-slider $(t_{\min}, t_{\min})$, $(t_{\min}, t_{\max})$, and $(t_{\max}, t_{\max})$, which we call the *query region* of the activity space. We call the line through $(t_{\min}, t_{\min})$ and $(t_{\max}, t_{\max})$ the *main diagonal* of the activity space.

For our data structures, we break down the visualization techniques into their atomic geometric elements, i.e., the edges of the α-shapes, the grid cells of the spatial density maps, and the labels of the labeling; see Figure 4.3b. We pre-compute for each atomic element the set of all time-window queries for which the atomic element is part of the visualization. We call this set of time-window queries *activity region* τ. For a particular time-window query Q specified by a user, we report all atomic elements whose activity regions contain Q.

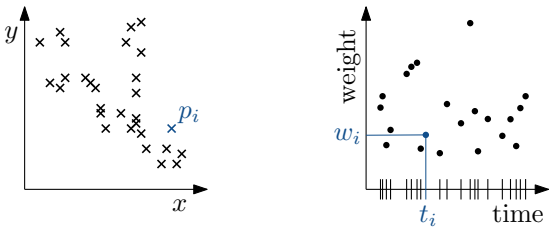


Figure 4.1: A set of events, the event e_i at location p_i, with timestamp t_i, and weight w_i is highlighted in blue.

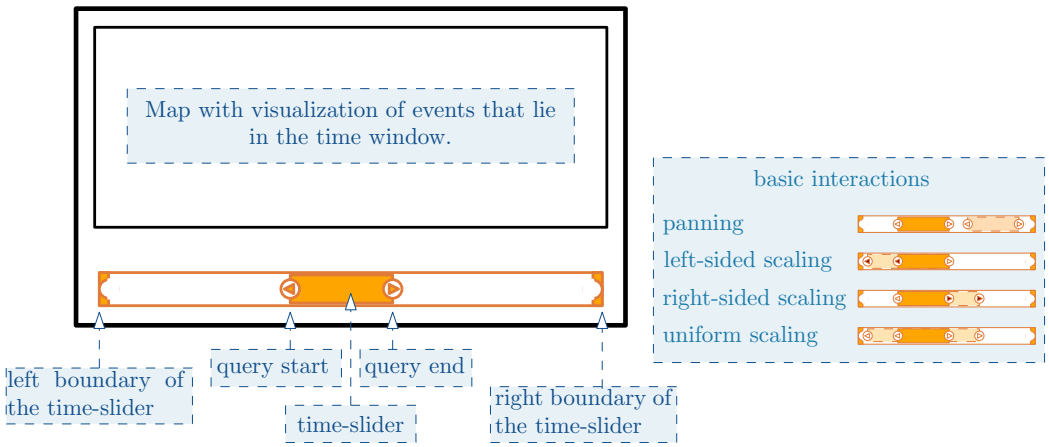
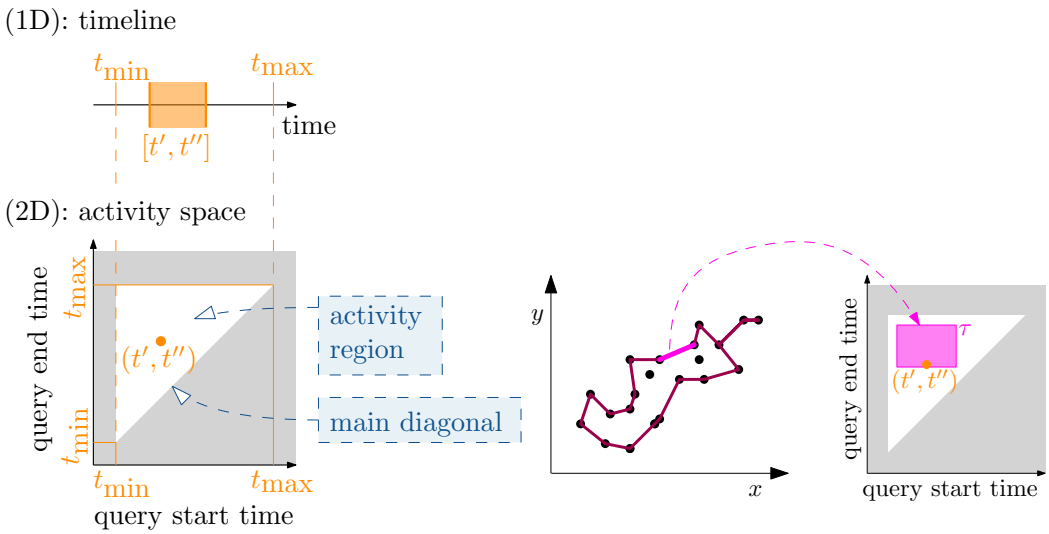


Figure 4.2: Time-slider interface and basic interactions.



(a) time-window query (b) activity region

Figure 4.3: Basic Concepts. (a) Time-window query $Q = [t', t'']$ in the one-dimensional view and the activity space. (b) Activity region τ of an edge (highlighted) of the α-shape for time-window query (t', t'').

5 α-Structure for Polygonal Event Representation

The following chapter is mainly taken from a joint work with Benjamin Niedermann and Jan-Henrik Haunert published at ACM SIGSPATIAL'19 [Bonerath et al., 2019]. It is an extension of the M.Sc. thesis [Bonerath, 2018] that won the second prize of the Karl-Kraus award of the German Society for Photogrammetry, Remote Sensing and Geoinformation (DGPF), the Austrian Society for Surveying and Geoinformation (OVG) and the Swiss Society for Photogrammetry and Remote Sensing (SGPF) [Bonerath et al., 2020b]. A preliminary version was published at the workshop EuroCG'19 [Bonerath et al., 2019].

Abstract

The interactive exploration of data requires data structures that can be repeatedly queried to obtain simple visualizations of parts of the data. We consider the scenario that the data is a set of events such that each event is a point in space and time. The result of each query is visualized by an α-shape, which generalizes the concept of convex hulls. Instead of computing each shape independently, we suggest and analyze a simple data structure that aggregates the α-shapes of all possible queries. Once the data structure is built, it particularly allows us to query single α-shapes without retrieving the actual (possibly large) point set and thus to rapidly produce small previews of the queried data. We discuss the data structure for the original α-shapes as well as for a schematized version of α-shapes, which further simplifies the visualization. We evaluate the data structure on real-world data. The experiments indicate linear memory consumption with respect to the number of events, which makes the data structure applicable in practice, although the size is quadratic for a theoretic worst-case example.

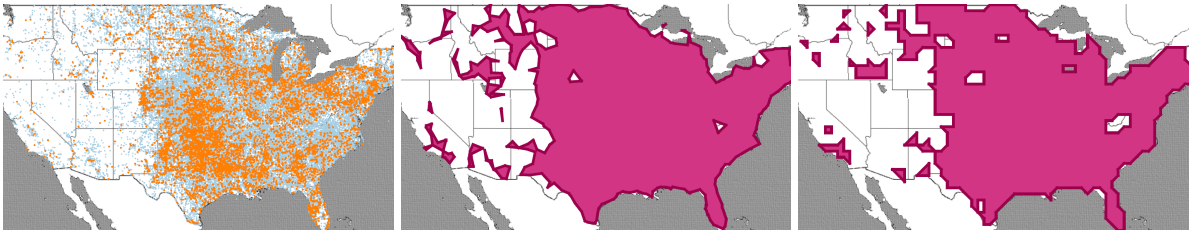
5.1 Introduction

In scientific projects that deal with large amounts of spatiotemporal data, data management is essential. As an example, take a project dealing with a database of storm events in the United States; see Figure 5.1. Each storm event is an event with a geo-location and a timestamp. More generally, an event may be associated with multiple timestamps. Assuming a collection of storm events over several decades, the amount of data becomes enormous. On the other hand, for certain scientific questions, the user may not be interested in all data but only in a subset in a pre-defined time window. Hence, before downloading the actual data for a thorough analysis, the user may be interested in exploring the data by querying simplified visualizations of the data within time windows. Typically such a query interface is implemented with a time-slider.

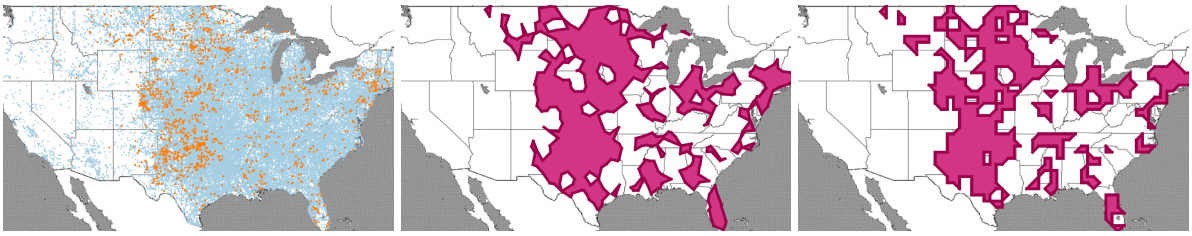
One approach to creating a simplified visualization is to sketch the outline of the queried data set providing the user with the possibility of roughly assessing the spatial distribution of the data. For example, the convex hull is a simple polygonal representation for that purpose. However, for most data sets this representation is not adequate, because the convex hull may easily cover large areas that do not contain any points of the data set.

In this chapter, we use α-shapes [Edelsbrunner, 2010, Edelsbrunner et al., 1983] for representing point sets, which are a generalization of convex hulls and strongly related to Delaunay triangulations. An α-shape of a set $P \subset \mathbb{R}^2$ of n points in the plane is defined as follows. Let $\alpha > 0$. The *spatial domain* of a directed edge $pq \in P \times P$ with $|q-p| \leq \alpha$ is the open disk D_{pq} with radius $\frac{\alpha}{2}$ whose center lies to the right of pq and whose boundary contains the points p and q. The set $S_\alpha(P) \subseteq P \times P$ of all edges that are shorter than α and do not contain any point of P in their spatial domain is called *α-shape*; see Figure 5.2. It can be computed via the Delaunay triangulation in $O(n \log n)$ time [Edelsbrunner et al., 1983].

In our use case, each point $p \in P$ additionally is associated with a *timestamp* $t_p \in \mathbb{R}$, and the pair (p,t) is called an event. Let E be the event set induced by P. To simplify the discussion, we assume that all points in P have pairwise distinct spatial and temporal coordinates; in Section 5.5 we explain how to generalize our approach to events with multiple timestamps. As described in the running example of Figure 5.1, the event set E is queried frequently, e.g., with a time-slider interface. Such a *query* Q is a time window $[t,t']$ and its result is the subset $P_Q = \{p \in P \,|\, t_p \in [t,t']\}$; see Figure 5.1a for the time-window query $Q_1 = [$1991-01-01,1992-01-01$]$ and Figure 5.1d for the time-window query $Q_2 = [$1991-06-01,1991-07-01$]$. We are then interested in visualizing P_Q



(a) storm events (orange) in the year 1991 (b) α-shape for year 1991 (c) octilinear α-shape for year 1991



(d) storm events (orange) in June 1991 (e) α-shape for June 1991 (f) octilinear α-shape for June 1991

Figure 5.1: Storm events (light blue) in the United States in the years 1991–2001. Each row highlights the storm events in a time window (orange) together with two polygonal representations (lilac). The polygons were generated with our approach. *Data retrieved from Data.gov.Map tiles by Stamen Design, under CC BY 3.0. Map data by OpenStreetMap, under ODbL.*

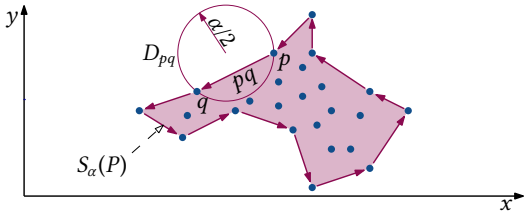


Figure 5.2: α-shape (lilac) for a point set (blue).

by its α-shape; see Figure 5.1b and Figure 5.1e for $\alpha_{200} := 200000$ meter. A straightforward approach for a query Q first queries the set P obtaining P_Q and then computes the α-shape $S_\alpha(P_Q)$. Utilizing a balanced binary search-tree, finding P_Q takes $O(\log n + |P_Q|)$ time. Additionally computing the α-shape, we obtain$O(\log n + |P_Q| \log |P_Q|)$ running time in total. For our use case of frequently providing α-shapes for visualizing the query results, we aim at a better running time per query. In particular, for creating previews of the data, we only want to retrieve the α-shape of P_Q but not the entire set P_Q. On the other hand, storing all possible α-shapes separately is also not a practical way as this results in cubic storage consumption. In a pre-processing phase, we, therefore, compute a data structure that aggregates the α-shapes of all possible queries; we call it the *α-structure* of P; see also Figure 5.3 for illustration. We use this data structure in the query phase to obtain the α-shapes of the incoming queries without receiving the actual point set. We consider the parameter α to be fixed before computing the data structure but provide an extension with which a variable α can be handled.

Besides the standard α-shapes, we also consider a schematized and especially an octilinear schematized variant of α-shapes, which simplifies the visualization; see Figure 5.1c and Figure 5.1f. Additionally, the experiments show that the data structure for schematized α-shapes has a substantially decreased memory consumption.

5.1.1 Our Contribution

In Section 5.2, we discuss related work with a focus on α-shapes. In Section 5.3, we formally introduce the problem setting and discuss properties of the α-structure. In Section 5.4, we present an algorithm that computes an α-structure in $O(n(\log n + m_R \log m_R))$ time utilizing linear and rotational sweeps, where m_R denotes the maximum number of points $p \in P$ in a square of width 2α. For the query phase, we suggest using an existing data structure for filtering search [Chazelle, 1986] to answer a query in $O(\log n + k)$ time, where in our case k

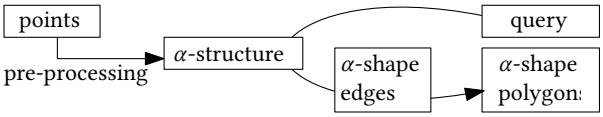


Figure 5.3: Pre-processing and using the α-structure.

is the number of edges of the returned α-shape. In Section 5.5, we generalize the α-structure to data sets for which each event may have one or more timestamps. With this adaption, the α-structure also supports spatial events that occur repetitively. Further, we use this to create α-shapes of points that are aggregated on a given grid. Choosing the size of the grid cells appropriately, we obtain octilinear α-shapes, i.e., the direction of each edge complies with one of eight pre-defined directions; we call this *octilinear α-shapes*. This schematization simplifies the visualization. In Section 5.6, we tweak the α-structure by further extensions. In Section 5.7, we present a detailed evaluation of our approach with real-world data. We show that on the considered data set the storage consumption grows linearly with the number of events. Further, we show that our data structure leads to improved query times. The schematized α-shapes perform well with respect to the similarity to the normal α-shape using the Jaccard index and the memory consumption.

5.2 Related Work

For an overview of other point set representations and time-windowed data structures, we refer to Chapter 2 and Chapter 3. In the following, we repeat the definition and properties of α-shapes [Edelsbrunner, 2010, Edelsbrunner et al., 1983] that we also use as a basis for the α-structure. α-shapes are a generalization of convex hulls and are strongly related to the Delaunay triangulation of a point set. The α-shape of a point set consists of one or more polygons, thereby the points are aggregated into several clusters, which is beneficial for our application scenario. This is beneficial for our application scenario because we do not need any additional clustering techniques. Another advantage of α-shapes for the temporal dynamic data structure, which we aspire to, is that we can decide whether an edge is contained in the α-shape of a point set based on local properties. More precisely, we only need to consider the points in the spatial domain of an edge. Among others, this technique finds its application in digital shape sampling and processing [Bernardini and Bajaj, 1997], in pattern recognition [Vauhkonen et al., 2010] and molecular biology [Edelsbrunner, 1992, Liang et al., 1998].

5.3 α-Structure

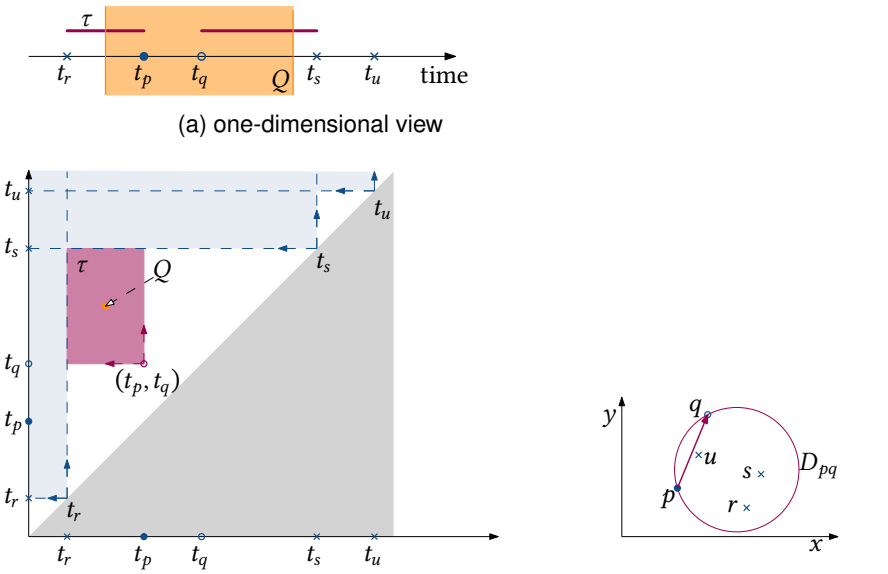
In this section, we introduce a data structure that provides the user with the possibility of receiving the α-shape of any possible temporal query $Q = [t,t']$. That is, for any query Q the data structure yields the set $S_\alpha(P_Q)$ containing the edges that describe the α-shape of the queried point set P_Q without computing P_Q. In the remainder of this chapter, we assume that $[t,t'] \subseteq [t_{\min},t_{\max}]$ for two pre-defined times $t_{\min}$ and $t_{\max}$, with $t_{\min} \leq t_p$ and $t_p \leq t_{\max}$ for all points $p \in P$. For two points p and q with timestamps t_p and t_q, we say that pq is *active* for an interval $[t,t']$ if pq belongs to the α-shape of the point set of the query $[t,t']$, i.e.,

1. p and q have distance at most α,
2. $t_p,t_q \in [t,t']$, and
3. there is no point r in the spatial domain of pq with timestamp t_r and $t \leq t_r \leq t'$.

Further, let L_{pq} be the set of all queries for which the edge pq is active; L_{pq} is the *activity region* of pq. We build the proposed data structure based on L_{pq}. To that end, we first characterize for which edges L_{pq} are not empty.

Lemma 1. *Let p and q be two points in P with timestamps t_p,t_q and $t_p \leq t_q$. The activity region L_{pq} is not empty if and only if pq is active for query $[t_p,t_q]$.*

Proof. First, assume that $L_{pq} \neq \emptyset$ and let $Q \in L_{pq}$. Hence, pq is active for Q. Therefore, we have that $t_p,t_q \in Q$, that the distance between p and q is at most α and that the spatial domain of pq is empty for Q. Therewith, the spatial domain is empty for $[t_p,t_q]$. Conversely, if p and q have distance at most α and there is no point r in the spatial domain of pq with timestamp t_r and $t_p \leq t_r \leq t_q$, then pq is active for query $Q' = [t_p,t_q]$ and therefore $Q' \in L_{pq}$. □



(a) one-dimensional view

(b) activity space

(c) spatial setting

Figure 5.4: Query $Q = [t,t']$ that yields an edge pq.

In order to describe the activity region L_{pq}, we interpret each time-window query $[t,t']$ as the two-dimensional point (t,t') in $\mathbb{R}^2$ in the following. Hence, L_{pq} becomes a region in $\mathbb{R}^2$; as $t \leq t'$, every query Q lies above the diagonal through $(0,0)$; see Figure 5.4; we call the plot in Figure 5.4b the *activity space*. We show that L_{pq} is an axis-aligned rectangle, which implies that computing $S_\alpha(P_Q)$ for a query Q corresponds to finding all edges $pq \in P \times P$ for which there is a rectangle τ_{pq} containing Q. Based on this observation, we define the *α-structure* of P to be the set $\mathscr{S}_\alpha = \{(\tau_{pq}, pq) \mid pq \in P \times P \text{ and } \tau_{pq} \neq \emptyset\}$.

In the following, we show that the activity region L_{pq} of an edge pq is an axis-aligned rectangle defined by the timestamps of the points that are contained in the spatial domain of pq. To that end, let t_p and t_q with $t_p < t_q$ be the timestamps of p and q, and let T_R be the set of all timestamps of points in the spatial domain of pq; see Figure 5.4a. Further, let $t_r \in T_R$ be the largest timestamp that is smaller than t_p; if t_r does not exist, we set $t_r = t_{\min}$. Similarly, let $t_s \in T_R$ be the smallest timestamp that is greater than t_q; if s does not exist, we set $t_s = t_{\max}$. Let τ be the axis-aligned rectangle that is spanned by the lower right point (t_p,t_q) and the upper left point (t_r,t_s); see Figure 5.4b. We call $\tau = (t_r,t_p,t_q,t_s)$ the *activity box* of p and q (with respect to t_p and t_q).

Lemma 2. *If for an edge pq its activity region L_{pq} is not empty, then the activity region is the activity box τ of p and q.*

Proof. Assume without loss of generality that $t_p \leq t_q$. Consider the activity box $\tau = (t_r,t_p,t_q,t_s)$ of pq. As L_{pq} is not empty, there is no point v in the spatial domain D_{pq} of pq with timestamp t_v and $t_p < t_v < t_q$. Hence, that edge pq is active for the query $Q = [t_p,t_q]$, which has a minimum length among all queries that are active for pq. Further, pq is active for all queries $Q = [t,t']$ with $t_r < t \leq t_p$ and $t_q < t' \leq t_s$ as for all these intervals D_{pq} is empty. On the other hand, pq is not active for queries $Q = [t,t']$ with $t \leq t_r$ or $t_s \leq t'$ as D_{pq} contains the point with timestamp t or t', respectively. Consequently, L_{pq} contains exactly all queries that lie in the range $[t_r,t_p] \times [t_q,t_s]$, which corresponds to τ. □

For our use-case of a database, the memory consumption of our approach is decisive for being deployed in practice. We first observe that $O(n^2)$ is an upper bound for the size of the α-structure. Theorem 1 shows that it is also a lower bound in the worst case.

Theorem 1. *For a set P of n points, the α-structure has size $\Omega(n^2)$ in the worst case.*

Proof. Let $P = \{p_1, p_2, \ldots, p_n\}$ be a point set with the time stamps $t_1 < t_2 < \ldots < t_n$ such that the points lie on a circle C of radius $r < \frac{1}{2}\alpha$ ordered clockwise; see Figure 5.5. Let $p_i, p_j \in P$ be two points with $t_i < t_j$. We show that p_ip_j is contained in the α-structure $\mathscr{S}_\alpha(P)$ by proving that p_ip_j is active for $[t_i,t_j]$ (Lemma 1). Due to $r < \frac{1}{2}\alpha$ the points p_i, p_j have distance smaller than α. Hence, it remains to be shown that there is no point $r \in P$ in the spatial domain D_{ij} of p_ip_j with time stamp t_r in $[t_i,t_j]$. We observe that D_{ij} and C intersect in p_i and p_j. Since the radius of C is smaller than the radius of D_{ij}, the boundary of D_{ij} partitions C into two parts. One part is

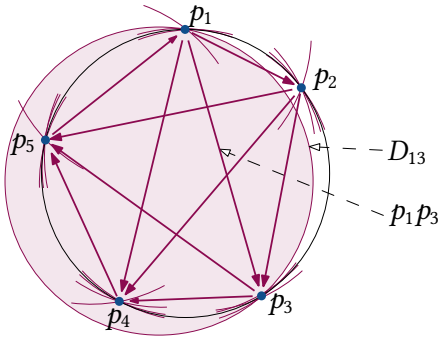


Figure 5.5: Worst-case example for the size of the α-structure.

contained in D_{ij} and the other lies outside of D_{ij}. As the points $p_1, p_2, \ldots, p_n$ appear in clockwise order on C, and since the center of D_{ij} lies to the right of p_ip_j by definition, we obtain that the points $p_1, \ldots, p_{i-1}, p_{j+1}, \ldots, p_n$ lie in D_{ij}, while the others of P do not. As $p_1, \ldots, p_{i-1}$ have smaller timestamps than p_i and $p_{j+1}, \ldots, p_n$ have larger timestamps than p_j, there is no point in the spatial domain of p_ip_j with timestamp in $[t_i, t_j]$. Hence, p_ip_j is active for $[t_i, t_j]$ and thus contained in the α-structure. □

Hence, the database may exceed a size that is applicable in practice. However, this example is rather unlikely to occur in real-world applications. Generally, the size of the data structure is bounded by $O(nm)$, where m is the largest number of points in a distance α to a point in P. Assuming that the point density is bounded by a constant and α is fixed, m is also constant. If, on the other hand, the density increases, it becomes more likely that an edge gets destroyed. Thus, in our real-world data experiments, in which we choose α small in comparison to the data extent, we observe a linear relation between the number of points and the size of the α-structure; see Section 5.7.

5.4 Construction and Query

We provide an algorithm that computes an α-structure in $O(n(\log n + m_{\mathrm{R}} \log m_{\mathrm{R}}))$ time and we describe how to query this data structure. The construction algorithm consists of three components utilizing two sweep-line approaches; see Algorithm 1. The first component, which we call *CPN-Search*, computes all points $CPN(p) \subseteq P$ that satisfy Condition (1) of active edges, i.e., all points that lie in a circle with center at p and radius α. We call this circle the *circle of potential neighbors (CPN)* of p. We use the sweep line approach by Peng and Wolff [2014] to find $CPN(p)$ in $O(\log n + m_{\mathrm{R}})$ time. More precisely, we use a vertical sweep line l that moves from left to right. The *status* $\mathscr{T}$ of the sweep line is the subset of points in P that have a horizontal distance of at most α to l. We store the points in the status in a binary tree in vertical order. We update the status whenever a point enters or leaves the vertical strip with width 2α and axis l. A *sweep event* p occurs when l is at p. We can identify all points $\mathscr{T}$ that lie in a square with center p and width 2α in $O(\log n + m_{\mathrm{R}})$ running time. Afterward, we filter these points obtaining $CPN(p)$.

The second component, which we call *Spatial Domain-Search*, computes the set of points R_{pq} that lie in the spatial domain of an edge pq. To implement this efficiently, we use a rotational sweep. More precisely, we use a circle C of radius $\frac{\alpha}{2}$ which sweeps counterclockwise around p such that the center of C moves along the

Algorithm 1: Computation of the α-structure

Input: Point set P, parameter α
Output: α-structure $\mathscr{S}_\alpha(P)$
Linear sweep on P from left to right:
foreach sweep event $p \in P$ **do**
  CPN-Search: Compute the $CPN(p)$
  Rotational sweep on $CPN(p)$ in counterclockwise order:
  foreach sweep event $q \in CPN(q)$ **do**
    Spatial Domain-Search: Compute set R_{pq}
    Activity Region-Search: Compute activity box τ_{pq} for R_{pq}. If $\tau_{pq} \neq \emptyset$, then add (τ_{pq}, pq) to $\mathscr{S}_\alpha(P)$.

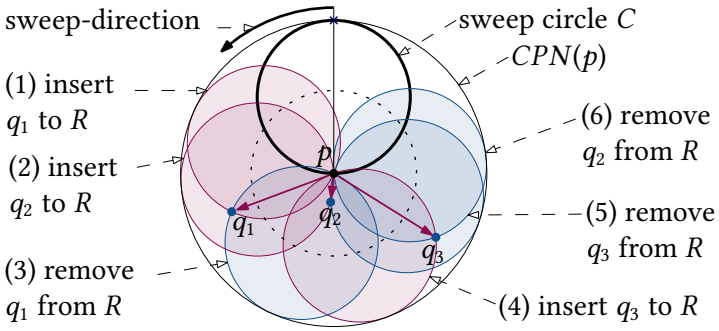


Figure 5.6: Rotational sweep at p with $CPN(p) = \{q_1, q_2, q_3\}$.

circle with center p and radius $\frac{\alpha}{2}$; see Figure 5.6. We call C the *sweep circle* of p. The *status* $\mathscr{R}$ of C contains the points of P that are currently contained in C. The sweep circle C handles an event whenever its boundary intersects with a point $q \in CPN(p)$. Two kinds of events are possible; either q enters C, or it leaves C. In the first case, q is added to $\mathscr{R}$ and in the latter, it is removed from C. Further, whenever a point q enters C, we return $\mathscr{R}$ as the spatial domain R_{pq} of the edge pq. This rotational sweep takes $O(m_R \log m_R)$ running time.

The third component, which we call *Activity Region-Check*, computes the activity region L_{pq} of pq based on R_{pq}. For this purpose, we store all points contained in C in a balanced binary search tree $\mathscr{B}$ ordered by their timestamps. The activity space or more precisely the activity box can be computed by querying $\mathscr{B}$ for the largest timestamp $t \in \mathscr{B}$ before $\min(t_p, t_q)$ and the smallest timestamp $t' \in \mathscr{B}$ after $\min(t_p, t_q)$. If t does not exist we set $t = t_{\min}$, analogously if t' does not exist we set $t' = t_{\max}$. If $t' \leq \max(t_p, t_q)$ the activity region L_{pq} is empty, otherwise it is equal to the activity box $\tau_{pq} = (t, \min(t_p, t_q), \max(t_p, t_q), t')$. We add (τ_{pq}, pq) to $\mathscr{S}_\alpha(P)$. Overall Algorithm 1 has running time $O(n(\log n + m_R \log m_R))$.

Theorem 2. *For a set P of n points the α-structure can be computed in $O(n(\log n + m_R \log m_R))$ time, where m_R is the maximum number of points in a square of width 2α.*

In the query phase, we need to find all two-dimensional activity boxes that contain the query point $Q = (t, t')$. We suggest using a data structure for filtering search [Chazelle, 1986] to answer a temporal range query in $O(\log n + k)$ time, where k is the number of edges of the returned α-shape. This approach has storage consumption $O(|\mathscr{S}_\alpha|)$.

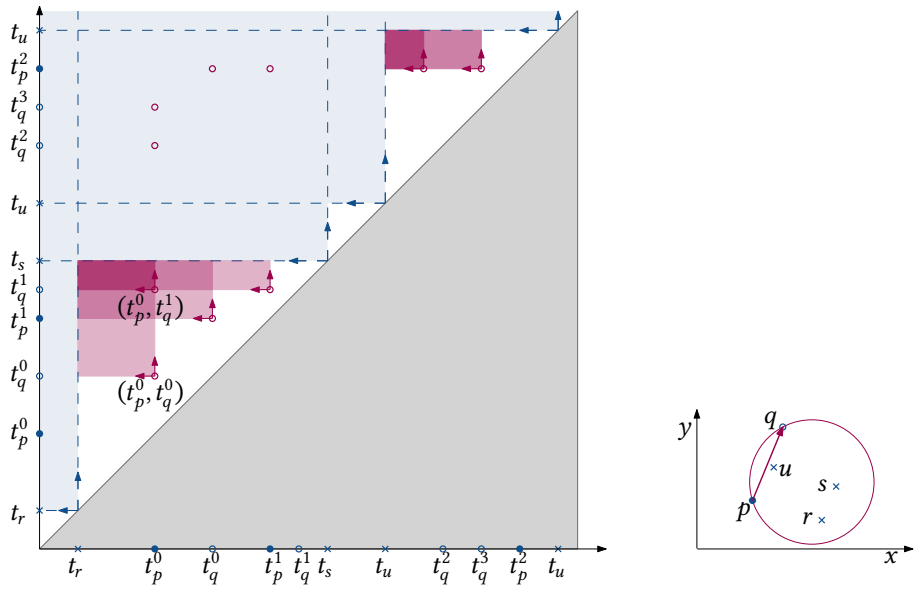
For applications where not the entire map is displayed, the query additionally consists of a spatial two-dimensional range $[x, x'] \times [y, y']$. We can describe each edge of the α-structure by a four-dimensional box and the spatiotemporal query corresponds to a 4-dimensional box, defined by $[x, x'] \times [y, y'] \times [t, t] \times [t', t']$. The edges that fulfill the spatiotemporal query are all those whose boxes intersect with the query box. Answering this query can be done in $O(\log^7 n + k)$ time and $O(|\mathscr{S}_\alpha| \log^3 n)$ storage using the orthogonal intersection search data structure by Edelsbrunner [1983].

5.5 α-Structure for Events with Multiple Timestamps

We extend our approach to the setting that each point $p \in P$ has a sequence $\mathbf{t_p}$ of timestamps so that the point set that matches the query $Q = [t, t']$ is the set $P_Q = \{p \in P \mid \mathbf{t}_p \cap [t, t'] \neq \emptyset\}$. As before, we aim at visualizing P_Q by presenting its α-shape. This generalization finds its application when visualizing events that occur repetitively. We first present the adaptions of our approach and then discuss one application of this extension that we use to substantially reduce the storage consumption of the α-structure.

5.5.1 Generalization of the α-Structure

We first observe that an edge $pq \in P \times P$ might have more than one activity box because each combination of timestamps of p and q might yield one activity box. As before, we define L_{pq} as the activity region of pq, i.e., the set of all queries for which the edge pq is active. We observe that this time L_{pq} might consist of multiple unconnected sub-regions. Further, it is easy to see that L_{pq} can be described as the union of all those activity boxes, since a query Q is contained in L_{pq} if and only if there is an activity box of pq that contains Q. It is therefore tempting to consider copies of the same point each having exactly one timestamp. However, this might lead to an unnecessarily large storage consumption and duplicates in the query results, as the activity boxes of the same edge, but different timestamps might intersect; see Figure 5.7. On the other hand, partitioning L_{pq} into a



(a) activity space (b) spatial setting

Figure 5.7: Activity region of pq for multiple timestamps.

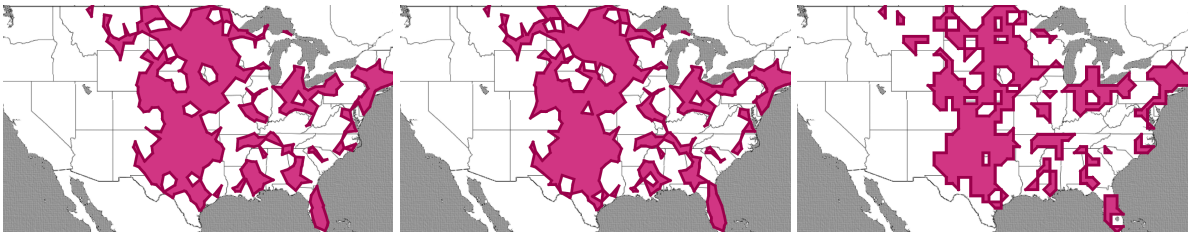


Figure 5.8: Original α-shape (left), α-shape on a grid with $\frac{\alpha}{w} \approx 10.1$ (middle), and octilinear α-shape (right) for $Q = Q_2$ and $\alpha = \alpha_{200}$. *Data retrieved from Data.gov.Map tiles by Stamen Design, under CC BY 3.0. Map data by OpenStreetMap, under ODbL.*

set τ_{pq} of axis-aligned rectangles could lead to a size of τ_{pq} that is quadratic in the number of timestamps of p and q. In the following, we show that for L_{pq} we can find a partitioning τ_{pq} into axis-aligned rectangles whose size is linear in the number of timestamps of pq.

To that end, we first show that each connected region of the activity region L_{pq} is a union of rectangular activity boxes of pq whose top left corners coincide; see Figure 5.7. We observe that the polygon derived by the union of the activity boxes is limited at the top by a horizontal segment h and on the left by a vertical segment v; it is completed by an xy-monotone rectilinear curve forming a staircase. We call such a polygon a *staircase polygon*.

Lemma 3. *If for an edge pq its activity region L_{pq} is not empty, then the activity region is a set of disjoint staircase polygons.*

Proof. Consider two activity boxes $\tau = (t_1, t_2, t_3, t_4)$ and $\tau' = (t_1', t_2', t_3', t_4')$ of pq that intersect; see Figure 5.7. By Lemma 1 the edge pq is active for $I_1 = [t_2, t_3]$ and $I_2 = [t_2', t_3']$. Assume without loss of generality that $t_2 \leq t_2'$; the other case is symmetric. By the definition of τ and τ', the intervals $[t_1, t_4]$ and $[t_1', t_4']$ do not contain any timestamp t_r of a point r in the spatial domain R of pq. Further, as τ and τ' intersect there is no timestamp t_r with $r \in R$ in the interval $I = [\min(t_1, t_1'), \max(t_4, t_4')]$. This implies that pq is active for I. Consequently, by the minimal choice of t_1 and t_1' and maximal choice of t_4 and t_4' required by the definition of an activity box, we obtain $t_1 = t_1'$ and $t_4 = t_4'$. This shows that the top left corners of τ and τ' are identical. Applying these arguments to all combinations of the timestamps of pq, we obtain that the activity region L_{pq} consists of a set of staircase polygons. □

In order to partition L_{pq} into disjoint axis-aligned rectangles, we connect each concave vertex (left bend when going in clockwise order around L_{pq}) with the topmost horizontal segment h of its staircase polygon by a vertical segment. Alternatively, one may also connect these vertices to the leftmost vertical segment v by a horizontal segment. Altogether, we obtain a partitioning τ_{pq} of L_{pq} into axis-aligned rectangles.

The proof of Lemma 3 implies that each pair t_p^i and t_q^j contributes at most one vertex to the staircase polygons of L_{pq} so that the size of the partitioned polygon into rectangles is linear in the number of combinations of timestamps of p and q. Further, we observe that a pair t_p^i and t_q^j can only contribute a vertex to a staircase polygon if no other point of $\mathbf{t}_p$ and $\mathbf{t}_q$ lies temporally in $[t_p^i, t_q^j]$. We call these pairs t_p^i, t_q^j *direct neighbors*. Assume there exists a $t \in \mathbf{t}_p$ with $t_p^i < t < t_q^j$. Then the activity box defined by the bottom right corner (t_p^i, t_q^j) is always contained in the activity box defined by the bottom right corner (t, t_q^j). The case $t \in \mathbf{t}_q$ is analog, e.g., see the activity boxes (t_r, t_p^0, t_q^0, t_s) and (t_r, t_p^0, t_q^1, t_s) in Figure 5.7. Thus, the size of the partitioned polygon is in $O(|\mathbf{t}_p| + |\mathbf{t}_q|)$.

Corollary 1. *The partitioning τ_{pq} has linear size in the number of timestamps of p and q, which implies that the size of $\mathscr{S}_\alpha$ is in $O(n_\mathrm{T}^2)$, where n_T is the number of timestamps over all points in P.*

For the construction, we adapt the algorithm of Section 5.4 as follows. Assume that the edge pq is currently considered in the activity region search of the algorithm. Going through the timestamps of p and q in increasing order, we compute for each pair of direct neighbors of $\mathbf{t}_p$ and $\mathbf{t}_q$ the activity box as described in the original algorithm. For two neighbored combinations t_p^i, t_q^j and t_q^l, t_p^k with $t_p^i < t_q^j < t_q^{j+1} < \ldots < t_q^{l+1} < t_q^l < t_p^k$, we check whether the activity boxes intersect and if necessary trim the later one. Overall, the activity region can be computed in $O((|\mathbf{t}_p| + |\mathbf{t}_q|) \cdot \log |R|)$ time.

5.5.2 Point Aggregation

For dense sets of input points, we observe that pairs of points easily lie closely together such that they can be hardly distinguished visually. On the other hand, depending on the timestamps of the points, each of such points might be a vertex of an α-shape for a particular temporal range query. Hence, the α-structure might contain a vast set of edges that are hardly visually different. As in our use case we are only interested in visualizing the approximate shape of the input set, we use this observation to reduce the storage consumption by aggregating points that lie close together. Moreover, depending on how we choose the aggregation, this provides us with the possibility of querying octilinear α-shapes schematizing the input points; we call an α-shape *octilinear* if its edges are restricted to verticals, horizontals, and diagonals at $45°$. In our experiments, we show that this preserves the rough visual appearance of the α-shapes, while substantially reducing the memory consumption and construction time of the α-structures.

For aggregating the points, we use a fairly simple technique that underlies the input set P of points with a grid G and moves each point p of P to the grid point of G that is closest to p. With this, the grid points correspond to a new set of input points having multiple timestamps. As described in Section 5.5.1, we obtain from the grid points an α-structure for points with multiple timestamps; see Figure 5.8. Further, in order to obtain octilinear α-shapes we choose α and the width w of a grid cell such that $\frac{\alpha}{w} \leq \sqrt{5}$. From simple geometric observations it follows that for any grid point p its CPN only contains grid points that lie on horizontals, verticals or diagonals at $45°$ through p. To see that, assume $w = 1$. The closest grid points to p that do not lie on an octilinear line with p have distance $\sqrt{5}$ from p. Hence, with $\frac{\alpha}{w} \leq \sqrt{5}$ any α-shape consists of octilinear line segments.

5.6 Extensions

In this section, we present extensions of our approach that can be used for further decreasing its storage consumption and for implementing other variants of visualizations.

5.6.1 Bridges

Depending on the concrete choice of α, the α-shape of a point set is not necessarily a set of simple polygons, but it might contain edges that form visually unpleasant artifacts without enclosing any area; see Figure 5.9. We call such edges *bridges*. Formally, an edge pq is a *bridge* in an α-shape S_α if it also contains the reversed edge qp. For the purpose of roughly sketching the spatial extent of the point set, the removal of the bridges seems to be a reasonable approach. Instead of simply removing the bridges in a post-processing step after querying the α-structure, we delete them as follows when creating the α-structure also improving its space consumption.

For simplicity, we assume that both p and q have one timestamp each; later on, we explain how to relax this restriction to multiple timestamps per point. For an edge pq and the reversed edge qp, let τ_{pq} and τ_{qp} be the two axis-aligned rectangles describing their activity boxes in the α-structure $\mathscr{S}_\alpha$; see Figure 5.10. We assume that both activity boxes are non-empty.

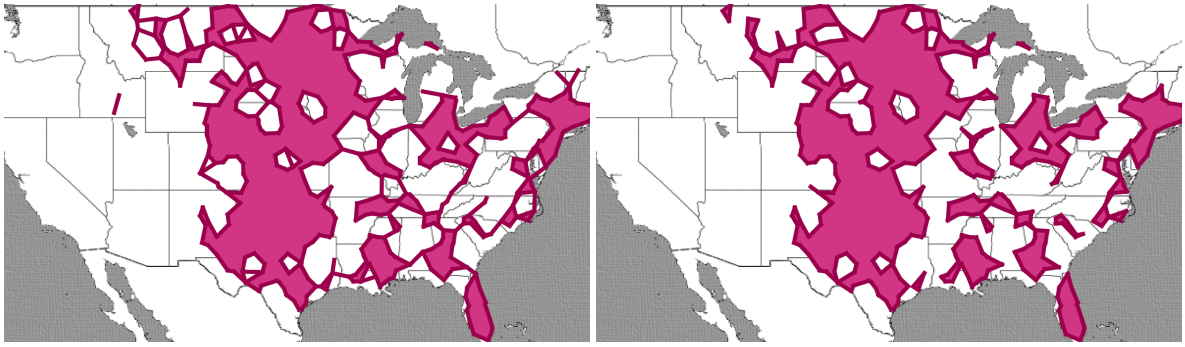
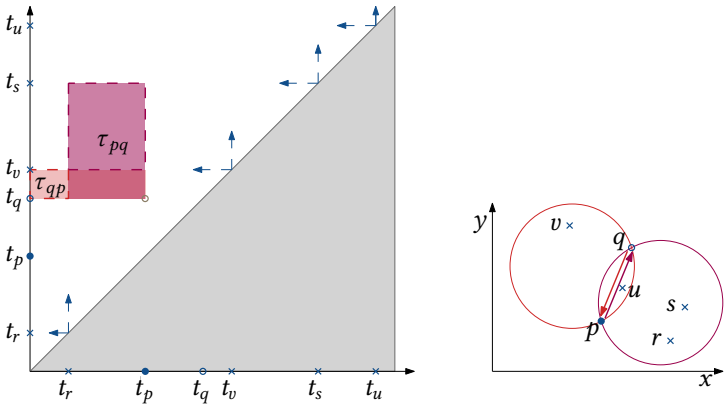


Figure 5.9: α-shape (α_{200}) with bridges (left) and without bridges (right) for $Q = Q_2$. *Data retrieved from Data.gov.Map tiles by Stamen Design, under CC BY 3.0. Map data by OpenStreetMap, under ODbL.*



(a) activity space (b) spatial setting

Figure 5.10: Activity boxes for the edges pq and qp.

Observation 1. *For two points p and q the activity boxes (if any) of the edges pq and qp, the bottom right corners coincide.*

We denote the set of all queries for which pq is a bridge by $\mathscr{B}_{pq}$; obviously, we have $\mathscr{B}_{pq} = \mathscr{B}_{qp}$. Assume that $\mathscr{B}_{pq}$ is not empty. We observe that the set $\mathscr{B}_{pq}$ is the intersecting axis-aligned rectangle of the two rectangles τ_{pq} and τ_{qp}, i.e., $\mathscr{B}_{pq} = \tau_{pq} \cap \tau_{qp}$. Further, by Observation 1 the bottom right corner of $\mathscr{B}_{pq}$ coincides with the bottom right corners of the activity boxes τ_{pq} and τ_{qp}. This implies that the symmetric difference of τ_{pq} and τ_{qp} can be partitioned into at most two rectangles τ_1 and τ_2. On account of that, we adapt the given α-structure $\mathscr{S}_\alpha$ as follows. We remove (τ_{pq}, pq) and (τ_{qp}, qp) from $\mathscr{S}_\alpha$. Further, if τ_1 is contained in τ_{pq}, we add (τ_1, pq) for the edge pq to $\mathscr{S}_\alpha$, and otherwise (τ_1, qp) for qp. Analogously, if τ_2 is contained in τ_{pq}, we add (τ_2, pq) for the edge pq to $\mathscr{S}_\alpha$, and otherwise (τ_2, qp) for qp. We note that this operation does not increase the size of the α-structure but potentially decreases the size. In our evaluation, we show that on real-world data the size is decreased by 10% on average.

In case that each point has multiple timestamps we similarly partition the symmetric difference of the two staircase polygons of pq and qp into axis-aligned rectangles. We then add this partitioning to the α-structure $\mathscr{S}_\alpha$ instead of the original partitioning of both staircase polygons. Using similar arguments as above, one can show that the partitioning can only decrease in its size.

5.6.2 Robustness

In this section, we adapt our approach to order-k α-shapes, which relaxes the restriction of empty spatial domains [Krasnoshchekov and Polishchuk, 2008, Krasnoshchekov and Polishchuk, 2014]. With this extension α-shapes become more robust against outliers also for larger values of α; see Figure 5.11. Formally, an edge pq belongs to an order-k α-shape S^k_α if p and q have a distance of at most α and its spatial domain contains exactly $k-1$ points. An edge pq is *active* for a query Q if it is contained in the order-k α-shape S^k_α of P_Q. The set L^k_{pq} is then the *order-k activity region* of pq. Analogously to α-shapes the region L^k_{pq} can be partitioned into a set τ_{pq}

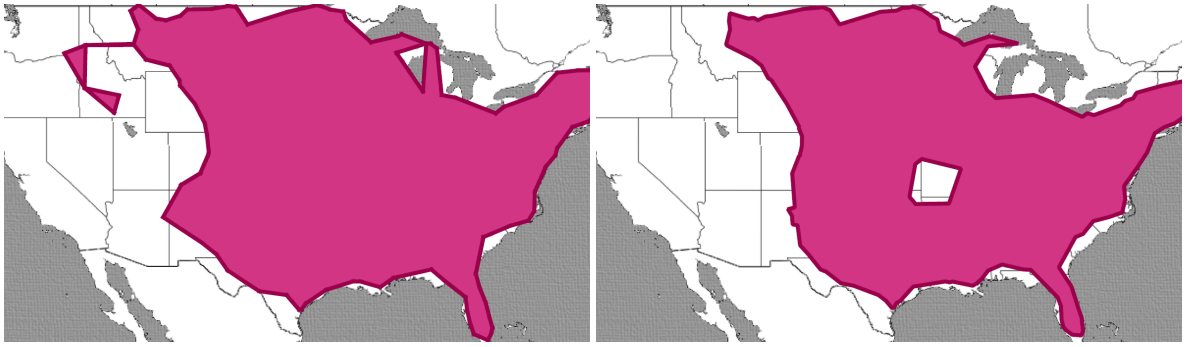
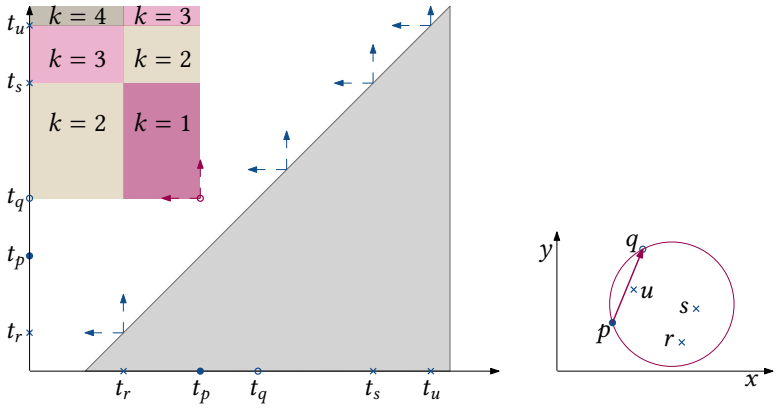


Figure 5.11: Original α-shape (left) and an order-2 α-shape (right) for $Q = Q_2$ and parameter $\alpha_{500} := 500 \cdot 10^3$. *Data retrieved from Data.gov.Map tiles by Stamen Design, under CC BY 3.0. Map data by OpenStreetMap, under ODbL.*



(a) activity space (b) spatial setting

Figure 5.12: Order-k activity regions for $k = 1, \ldots, 4$ of edge pq.

of axis-aligned rectangles. The *order-k α-structure* is then the set $\mathscr{S}_\alpha^k = \{(\tau, pq) \mid pq \in P \times P \text{ and } \tau \in \tau_{pq}\}$.

In case each point has one timestamp, the region L_{pq}^k can be partitioned into k axis-aligned rectangles or less as shown in the following. To that end, let t_p and t_q be the timestamps of p and q and assume $t_p < t_q$. Further, let T_R be the timestamps of the points contained in the spatial domain R of pq; see Figure 5.12. Let $t_1, t_2, t_3, t_4 \in \{t_p, t_q\} \cup \{t_r \mid r \in R\} \cup \{t_{\min}, t_{\max}\}$ such that $t_1 < t_2 \leq t_p < t_q \leq t_3 < t_4$. The rectangle $\tau = (t_1, t_2, t_3, t_4)$ is an *order-k activity box* of pq if there are exactly k points in $R \setminus \{p_1, p_2, p_3, p_4\}$ that have a timestamp in the time range $[t_1, t_4]$.

Lemma 4. *For an edge pq, the region L_{pq}^k can be partitioned into a set of order-k activity boxes. Further, the number of order-k activity boxes for pq is at most k.*

Proof. We observe that by definition of an order-k activity box, an edge pq is active for a query Q if and only if Q lies in an order-k activity box. Hence, the set L_{pq}^k is the union of all order-k activity boxes that exist for pq. Further, for an order-k activity box $\tau = (t_1, t_2, t_3, t_4)$ of pq there is no point r in the spatial domain of pq that has a timestamp t_r with $t_1 < t_r < t_2$ or $t_3 < t_r < t_4$. Assuming a point r lies between $t_3 < t_r < t_4$ then a query (t, t') with $t_1 < t < t_2$ contains a different number of points for $t_3 < t' < t_r$ than for $t_r < t' < t_4$. This contradicts the condition that pq has exactly $k-1$ points in its spatial domain for each query Q in τ. This implies that the order-k activity boxes of pq are disjoint. In particular, for the same reason we have for any other activity box $\tau' = (t_1', t_2', t_3', t_4')$ of pq that if $t_2 \leq t_2'$ then $t_3 \leq t_3'$; see Figure 5.12. Consequently, any query in τ' also contains the timestamps of τ. As it contains the timestamps of exactly $k-1$ other points in the spatial domain of pq, the number of order-k activity boxes of pq is at most $k-1$. □

From Lemma 4 we directly derive that for each edge $pq \in P \times P$ we need $O(k)$ extra storage, which leads to $O(knm)$ storage in total.

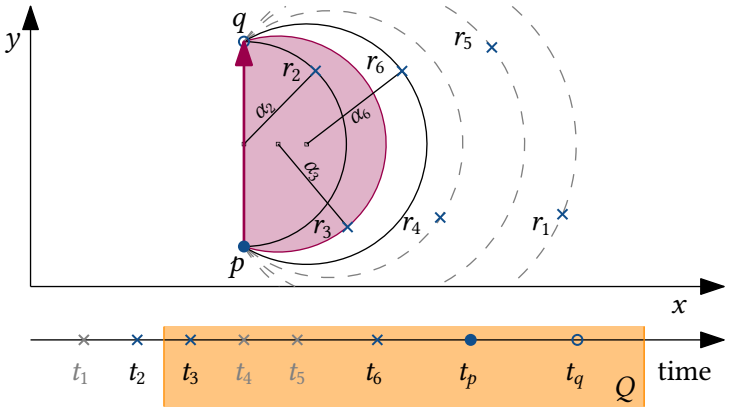


Figure 5.13: Edge in a multi-α-structure.

5.6.3 Multi-α-Structure

Up to now, we require that α is fixed before the α-structure is constructed. However, depending on the use case the exact α depends on user-specific requirements during the query phase. For example, the user might want to display multiple nested α-shapes with different α to illustrate different densities of the point sets. We shortly sketch how to extend α-structures such that they provide queries for arbitrary choices of α from a pre-defined interval $[\alpha_{\min}, \alpha_{\max}]$.

Let $pq \in P \times P$ be an edge of two points p and q with timestamps t_p and t_q and $t_p < t_q$, and let $R = \{r_1, \ldots, r_k\}$ be the points that lie in the spatial domains of pq over all $\alpha \in [\alpha_{\min}, \alpha_{\max}]$; see Figure 5.13. We denote the timestamp of r_i by t_i and assume that $r_1, \ldots, r_k$ are ordered such that $t_i < t_j$ with $1 \leq i \leq j \leq k$. We denote the radius of the circle C_i through p, q and r_i by α_i. Further, let $R_\mathrm{r} \subseteq R$ be the points on the right-hand side and $R_\mathrm{l} \subseteq R$ be the points on the left-hand side of pq. We observe that a point $r_i \in R_\mathrm{r}$ is contained in the circle C_j of any other point $r_j \in R$ with $\alpha_j \geq \alpha_i$. Similarly, a point $r_i \in R_\mathrm{l}$ is contained in the circle C_j of any other point $r_j \in R$ with $\alpha_j \leq \alpha_i$. For example in Figure 5.13, the point r_3 is contained in the circles C_6, C_4, C_5 and C_1. Hence, for any query with a start time smaller than t_3 the maximal possible $\alpha/2$ is α_3. We formalize this observation as follows. Consider a query $Q = [t, t']$ with $[t_p, t_q] \subseteq Q$. Further, let t_i be the largest timestamp with $t_i \leq t$ and let t_j be the smallest timestamp with $t' \leq t_j$. The edge pq is active for Q and a given α if and only if for each r_h with $i+1 \leq h \leq j-1$ it holds $\alpha_h \geq \alpha/2$ if $r_h \in R_\mathrm{r}$, and $\alpha_h \leq \alpha/2$ if $r_h \in R_\mathrm{l}$.

We determine for pq the smallest such i and the largest such j for which there is a query Q for which pq is active. We add the activity box (t_i, t_p, t_q, t_j) to the α-structure and annotate it as follows. Firstly, we determine the point $r_\mathrm{r} \in R_\mathrm{r}$ with $t_\mathrm{r} \in [t_p, t_q]$ and smallest α_r among all such points of R_r, and secondly the point $r_\mathrm{l} \in R_\mathrm{l}$ with $t_\mathrm{l} \in [t_p, t_q]$ and largest α_l among all such points of R_l. If $\alpha_\mathrm{r} < \alpha_\mathrm{l}$, then the edge is never active as any spatial domain of pq contains r_r or r_l. Hence, in that case, the edge pq is excluded from the α-structure. Otherwise, we additionally store the set $S_\mathrm{r} \subseteq R_\mathrm{r}$ which contains all points r_i, r_j with $t_i, t_j \notin [t_p, t_q]$, $\alpha_i < \alpha_j \leq \alpha_\mathrm{r}$ if $t_i < t_j < t_p$, and $\alpha_\mathrm{r} \geq \alpha_i > \alpha_j$ if $t_q < t_i < t_j$; for example $S_\mathrm{r} = \{r_2, r_3, r_6\}$ in Figure 5.13. Analogously, we store the set $S_\mathrm{l} \subseteq R_\mathrm{l}$ which contains all points r_i, r_j with $t_i, t_j \notin [t_p, t_q]$, $\alpha_i > \alpha_j \geq \alpha_\mathrm{l}$ if $t_i < t_j < t_p$, and $\alpha_\mathrm{l} \leq \alpha_i < \alpha_j$ if $t_q < t_i < t_j$. For the query $Q = [t, t']$ we check whether pq is active for α by checking that the spatial domain $R = R_\mathrm{r} \cup R_\mathrm{l}$ of pq with radius $\alpha/2$ is empty. To that end, we determine the smallest h with $r_h \in S_\mathrm{r}$ and $t < t_h$ as well as the largest h' with $r_{h'} \in S_\mathrm{r}$ and $t_{h'} < t'$. If $\alpha/2 < \min\{\alpha_\mathrm{r}, \alpha_h, \alpha_{h'}\}$ then all points of R_r lie outside the spatial domain of pq with radius $\alpha/2$. Analogously, we determine the smallest h with $r_h \in S_\mathrm{l}$ and $t < t_h$ as well as the largest h' with $r_{h'} \in S_\mathrm{l}$ and $t_{h'} < t'$. If $\alpha/2 > \max\{\alpha_\mathrm{l}, \alpha_h, \alpha_{h'}\}$, then all points of R_l lie outside of the spatial domain of pq with radius $\alpha/2$.

Note that for each edge pq we need to store $O(n)$ points of R in the worst case. Hence, supporting multiple α values the storage consumption increases by a linear factor. However, the worst case seems to be unlikely in practice, as this requires that the timestamps $t_1, \ldots, t_k$ and the radii $\alpha_1, \ldots, \alpha_k$ have strongly correlated orders. Further, the query time is increased only by a factor of $O(\log n)$ as we can use binary search to determine α_h and $\alpha_{h'}$.

5.7 Evaluation and Experiments

For our experiments, we use two data sets. The first data set `storm` is a set of storm events in the United States of America obtained from Data.gov. This data set consists of 59789 spatially and temporally unique events that occurred during the years 1991–2000. We used the Web-Mercator projection; see Figure 5.14. Based on

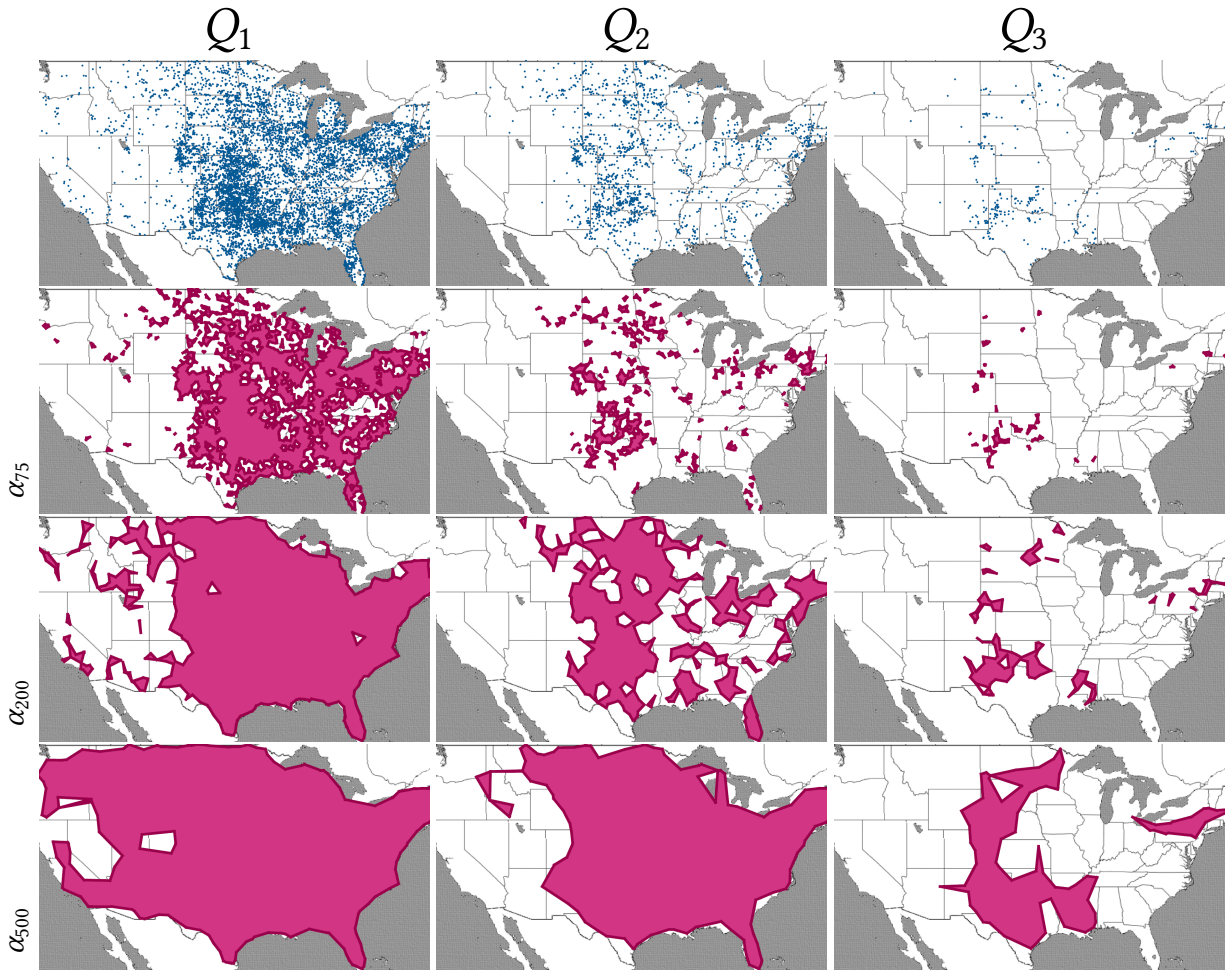


Figure 5.14: α-shapes for different queries and α values. *Data retrieved from Data.gov.Map tiles by Stamen Design, under CC BY 3.0. Map data by OpenStreetMap, under ODbL.*

preliminary experiments taking the visual appearance into account, we run our experiments with $\alpha_{75} = 75$ km, $\alpha_{200} = 200$ km, and $\alpha_{500} = 500$ km. In Figure 5.14, we show α_{75}, α_{200} and α_{500} for three time-window queries $Q_1 = [1991\text{-}01\text{-}01, 1992\text{-}01\text{-}01]$, $Q_2 = [1991\text{-}06\text{-}01, 1991\text{-}07\text{-}01]$ and $Q_3 = [1991\text{-}06\text{-}05, 1991\text{-}06\text{-}12]$. While α_{75} leads to small clusters, α_{200} and α_{500} yield few polygons, which cover large areas. The α-shape edges were obtained by querying the α-structure whereas the α-shape polygons were computed in a post-processing step.

The second data set `socialmedia` is obtained from FlickR where each data point corresponds to an image taken at a certain location and at a certain timestamp. We extracted 3 million spatially and temporally unique events that were uploaded between 2019-2021 and lie in Germany. Here, we use one configuration of the α-structure with $\alpha = 5$ km, we keep the bridges, and do not aggregate the data on a grid for our experiments.

We have implemented the construction algorithm in Java and stored the α-structure in a PostGIS database using a four-dimensional BRIN-Index applied on the activity box parameters [PostgreSQL Global Development Group, 2018]. We ran all experiments on a quad-core Intel Core i7-7700T CPU with 16 GiB RAM.

Queries We first evaluate the query phase for the α-structure of `storm` with $\alpha = \alpha_{200}$. We deem α_{200} to be an appropriate choice for the considered data set, as the resulting α-shapes lead to reasonably clustered point sets; see Figure 5.14. We have created for each i with $i \in \{1, \ldots, 5\}$ a set $\mathcal{Q}_i$ of five time-window queries that have random starting points and yield $i \cdot 10^4$ points; we denote the union of all $\mathcal{Q}_i$ by $\mathcal{Q}$. Figure 5.15a shows the average size of the α-shapes for each set $\mathcal{Q}_i$. It shows that the α-shape consists of at most 550 edges on average and is almost constant with increasing number of queried points. Hence, instead of retrieving the entire set P_Q for a query Q, the α-structure reduces the amount of processed objects by several orders of magnitude. Further, Figure 5.15b shows that the α-structure reduces the query times. A query $Q \in \mathcal{Q}_5$ on the α-structure takes on average 120 ms while querying P_Q directly takes on average 320 ms. For this evaluation, we only engineered the retrieval of the edges but did not invest special effort into the implementation of constructing the

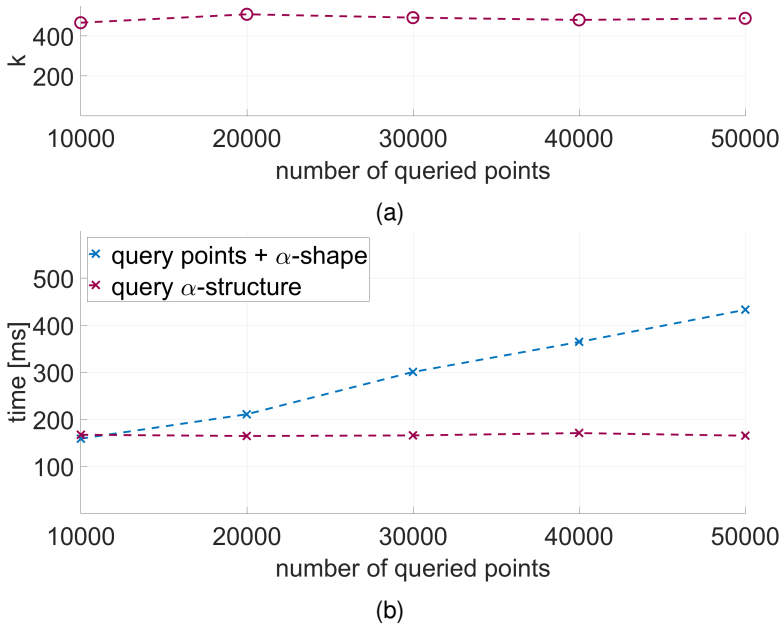


Figure 5.15: Performance of querying the α-structure (α_{200}) of `storm` data set. (a) Number of queried edges k. (b) Query phase time.

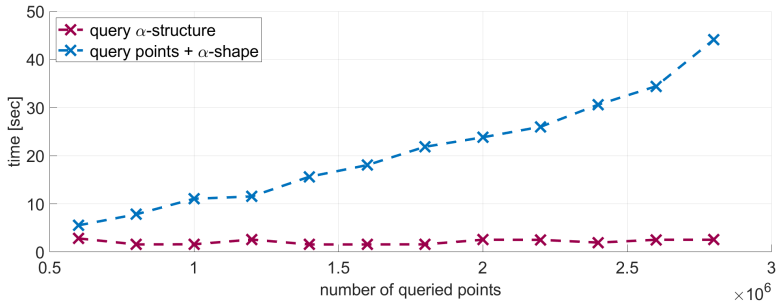


Figure 5.16: Query phase time of the α-structure of `socialmedia` data set.

polygon from the edge set, which takes 45 ms on average and 190 ms in maximum over all queries $\mathcal{Q}$. Directly constructing the α-shape from P_Q leads to similar running times of 110 ms on average and 205 ms in maximum over all $Q \in \mathcal{Q}$.

In Figure 5.16, we give the results for evaluating the α-structure for `socialmedia` data. It shows a similar trend to the results obtained for `storm` but with an overall higher magnitude. While querying the α-structure is always below 3 sec, the on-demand computation of the α-shape takes up to 44 sec. Although 3 sec is not sufficiently fast for real-time interaction, we improve the running time by a factor of 15 for the worst query times. Hence, the experiments show that α-structures are a simple and fast tool that provide the user with the possibility of querying previews of the data frequently.

Memory Consumption For evaluating the memory consumption of α-structures, we randomly sampled subinstances of both data sets. For each of these instances, we created an α-structure. Figure 5.17a and Figure 5.17b show for each instance size the number of activity boxes stored in the α-structure for `storm` and `socialmedia`, respectively. They indicate a linear growth of the α-structure. This is remarkable as in the worst case the structure has a quadratic size in the number of points. We explain this with the fact that with an increasing number of points and therefore with increasing density, it becomes substantially more likely that a point lies in the spatial domain of an edge having its timestamp in between the two-timestamps of the corresponding points of the edge.

For `storm`, we performed a more detailed analysis considering the α value and the influence of deleting bridges on the memory consumption; see Figure 5.17a. As to be expected, the greater α, the more the α-structure consumes memory. The reason is the greater number of points in the circles of potential neighbors for a greater α. Still, for all considered values of α, the number of edges is higher than the number of points by

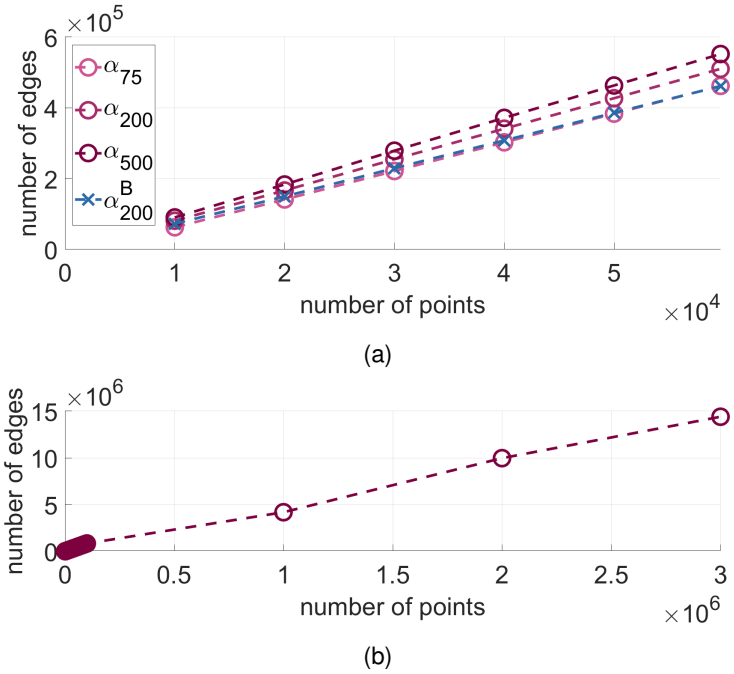


Figure 5.17: Memory consumption for both data sets `storm` and `socialmedia`. (a) Memory consumption for the data set `storm` comparing different α values and a version where we deleted the bridges (α^B_{200}). (b) Memory consumption for the data set `socialmedia` of the α-structure ($\alpha = 5$ km, keeping bridges, no point aggregation).

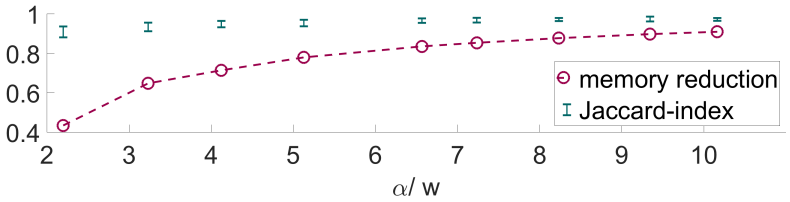


Figure 5.18: Memory reduction of schematized α-structure (α_{200}) and shape similarity for schematized α-shapes, i.e., the range of Jaccard index over all queries in $\mathcal{Q}$. On the x-axis we display the ratio between α and the grid width w.

at most a factor of 10. Further, deleting bridges reduces the memory consumption for α_{75} by around 12%, for α_{200} by 10%, and for α_{500} by around 8% on average. Hence, removing bridges helps both improving the visual appearance (see Figure 5.9) and the memory consumption.

Aggregated points As another possibility of reducing the memory consumption, we aggregated the points on a grid and performed experiments for `storm`; see Section 5.5.2 and Figure 5.8. For $\alpha = \alpha_{200}$, Figure 5.18 shows the *compression* for different choices of the grid width w, i.e., the proportion of activity boxes for $\frac{\alpha_{200}}{w}$ with respect to the activity boxes in the α-structure constructed on the same point set without aggregation. With decreasing value of $\frac{\alpha}{w}$ the compression increases, e.g., for $\frac{\alpha}{w} = \sqrt{5}$, which corresponds to octilinear layouts, only 40% of the activity boxes remain in the α-structure. As the compression influences the visual appearance of the α-shapes, we computed for each query Q of the set $\mathcal{Q}$ the α-shape S^g_α for each chosen ratio $g = \frac{\alpha}{w}$. Using the Jaccard index we compare S^g_α to the α-shape S_α for the according point set without aggregation, i.e., for each S^g_α and S_α we consider $\frac{|S^g_\alpha \cap S_\alpha|}{|S^g_\alpha \cup S_\alpha|}$, where $|S^g_\alpha \cap S_\alpha|$ ($|S^g_\alpha \cup S_\alpha|$) is the area of the intersection (union) of S^g_α and S_α; see Figure 5.18. We note that S^g_α approximates S_α well, even for a strong compression such as $g = \sqrt{5}$.

Construction Time We evaluated the construction time for both data sets `storm` and `socialmedia`; see Figure 5.19. The experiments show that even for the largest data set (`socialmedia` with 3000000 events) the construction time is manageable when considering that the construction is done in a pre-processing phase.

We analyzed the construction time more in detail for the data set `storm`. The construction time strongly relies on the chosen value of α. While for α_{75} and α_{200} the algorithm needs less than two minutes, for α_{500} the

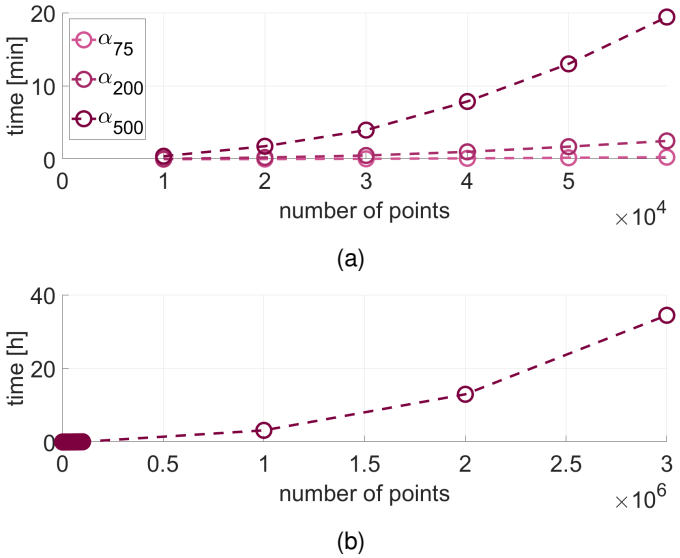


Figure 5.19: Evaluation of the construction of an α-structure. (a) Construction time for the data set `storm`. (b) Construction time for the data set `socialmedia`.

construction takes about 20 minutes. This increase is reasoned in the higher number of points in the CPN for α_{500}. As the construction is only done once in a pre-processing step, we deem the obtained running times to be practical.

Order-k α-Structures and Multi-α-Structures As the original α-structure and its practicability is the focus of the evaluation, we only shortly analyze multi-α-structures and order-k α-structures giving first indications of their usage; a detailed analysis is planned for an extended version of our work. As to be expected, the memory consumption of multi-α-structures increases substantially, as for each edge it becomes much more likely that it belongs to the α-structures. Preliminary experiments show that the size increases at least by a factor of 10 in comparison to the original α-structure.

For order-k α-structures the memory consumption increases by a factor of 5.36 for α_{200} with respect to the original α-structure. Despite the increased memory consumption, we feel that order-k α-shapes provide a helpful tool for aggregating point sets and removing outliers; for example see Figure 5.11, which shows that the α-shape is reduced to the main set of the points excluding outliers.

5.8 Conclusion

Overall, we presented the design and construction of a data structure that provides the edges of α-shapes for time-window queries on event sets. Further, we showed how to use this data structure to provide the user with schematized and especially octilinear α-shapes. Our experiments showed that the memory consumption of the data structure is linear with the number of events, which makes it applicable in practice. Further, we showed that the query time is shorter than using a straightforward implementation.

6 θ-Structure for Grid-Based Event Visualization

This chapter on the θ-structure is largely extracted from a joint work with Benjamin Niedermann, Jim Diederich, Yannick Orgeig, Johannes Oehrlein, and Jan-Henrik Haunert which was published at ACM SIGSPATIAL'20 [Bonerath et al., 2020c]. Preliminary results were presented in the M.Sc. thesis by Jim Diederich [Diederich, 2019].

Further, we also look at a second version of the θ-structure. The θ^*-structure is technically closely related but it handles the visualization of public transportation networks instead of events that are points in space and time. This section is extracted from a joint work with Yu Dong and Jan-Henrik Haunert which was submitted to ICC'23 [Bonerath et al., 2023a]. Preliminary results were presented in the M.Sc. thesis by Yu Dong [Dong, 2021].

Abstract

The visualization of spatiotemporal data helps researchers understand global processes such as animal migration. In particular, interactively restricting the data to different time windows reveals new insights into the short-term and long-term changes in the research data. Inspired by this use case, we consider the visualization of point data annotated with timestamps. We pick up classical, grid-based density maps as the underlying visualization technique and enhance them with an efficient data structure for arbitrarily specified time-window queries, e.g., triggered by a time-slider interface. The running time of the queries is logarithmic in the total number of points and linear in the number of actually colored cells. In experiments on real-world data, we show that the data structure answers time-window queries within milliseconds, which supports the interactive exploration of large point sets. Further, the data structure can be used to visualize additional decision problems, e.g., it can answer whether the sum or maximum of additional weights given with the points exceed a certain threshold. We have defined the data structure generally enough to also support multiple thresholds expressed by different colors.

6.1 Introduction

Interactive, map-based visualizations of spatiotemporal data are an important tool when it comes to understanding underlying processes. By supporting the retrieval of visualizations for arbitrary time windows they enable the user to investigate both global and local patterns in the data. As an example take the yearly migration of birds as illustrated in Figure 6.1. The underlying data contains over 8 million data points each representing the occurrence of one bird at one day within the time range from 17/05/2013 to 31/08/2016. For the visualization of such data, we use a grid-based density map with a discrete set of colors. Figure 6.1(a) shows the living area of the birds over the entire time range by aggregating the given point data into cells. Restricting the time window to a certain extent helps to identify more details as shown in Figure 6.1(b)–(c). Moreover, when sliding a time window of arbitrary but fixed size over the data we obtain a sequence of visualizations such as in Figure 6.2 illustrating the movement of the birds.

In this chapter, we present a data structure, which we call *θ-structure*, that provides the possibility of retrieving such density maps for arbitrarily chosen time windows in real-time. Hence, the user can specify any extent and position of the time window to investigate both short-term and long-term processes, e.g., with a time-slider interface. The proposed data structure answers such time-window queries within milliseconds even for large data sets, which enables a pleasant user experience for the time-slider interface. We invite the reader to try out our prototypical implementation of such a system on `www.geoinfo.uni-bonn.de/densitymaps`.

Formally, we model the time-window queries as the following *decision problem*, which is solved for each cell of the density map.

Decision Problem: Time-Window Query.
Given: *A set P of spatiotemporal points, a threshold θ and a time window Q.*
Question: *Does the number of points of P that lie in Q exceed the threshold θ?*

Hence, when answering a time-window query the θ-structure provides all cells that need to be colored. To achieve fast query times, the θ-structure efficiently encodes and stores the answers for all possible time-window

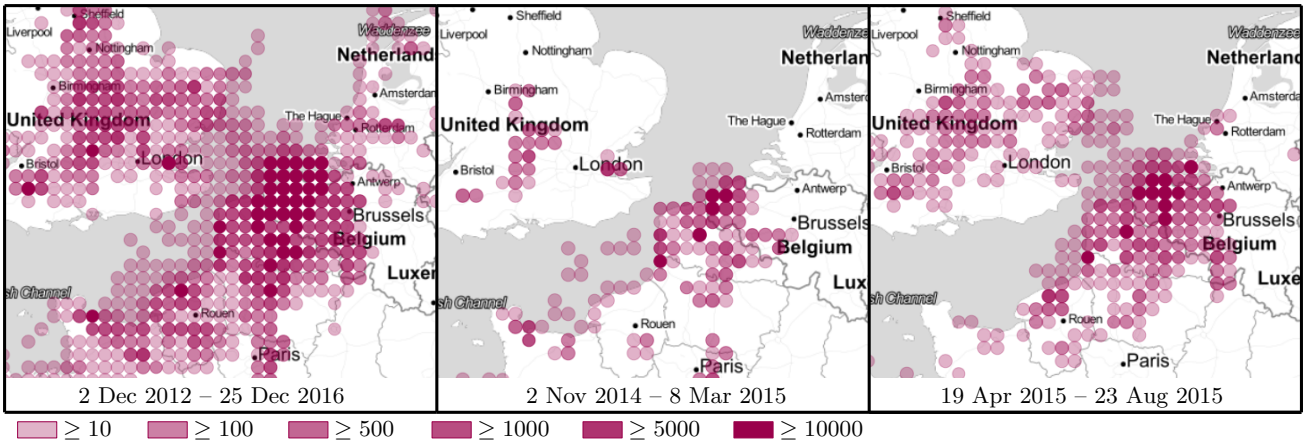


Figure 6.1: Density maps showing the *number of occurrences* of gulls (Larus argentatus and Larus fuscus) in Western Europe. Left: the entire time range for the available data. Middle: in the winter some gulls migrate further south. Right: in the summer their population has its maximum. *Map tiles by Stamen Design, under CC BY 3.0. Map data by OpenStreetMap, under ODbL.*

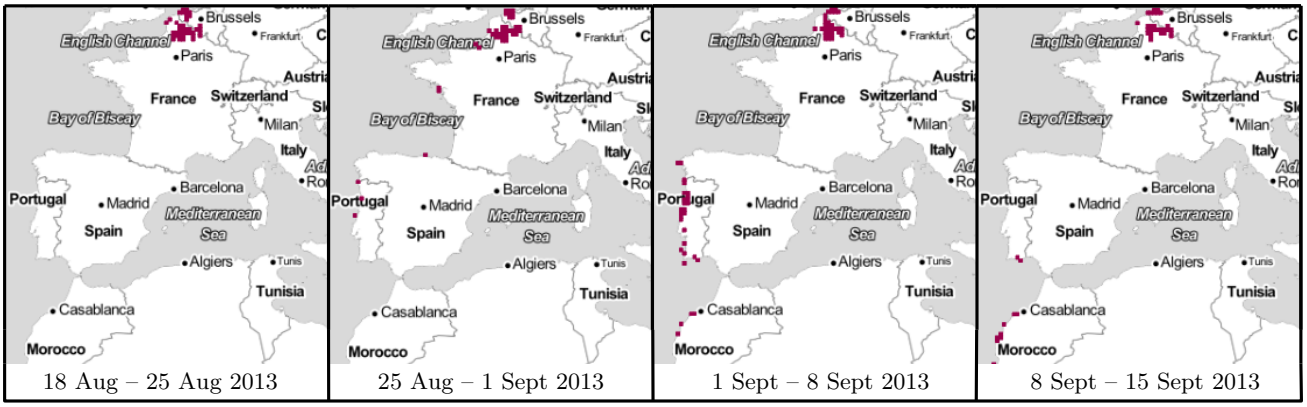


Figure 6.2: Density maps showing the migration of gulls from Western Europe to Morocco within four weeks. Colored cells show at least 10 occurrences of gulls. *Map tiles by Stamen Design, under CC BY 3.0. Map data by OpenStreetMap, under ODbL.*

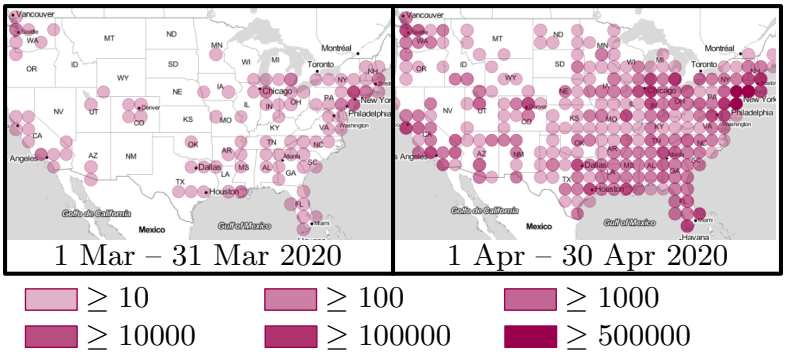
queries in a tree-based search structure. When answering a query the θ-structure only considers those cells that are actually colored. This leads to an output-sensitive query time that is based on the number of returned cells. Moreover, the data structure is space efficient and can be built within a minute even for large data sets.

We define the θ-structure general enough so that it can be used for any *monotone decision problem*. For such problems, it holds for any two queries Q and Q' with $Q \subseteq Q'$ that Q is a *yes*-instance if Q' is also a *yes*-instance. For example, assume that each point is annotated with an additional weight. Deciding whether the sum or the maximum of the point weights in a cell exceeds a given threshold is such a monotone decision problem. For the sum, we consider the spread of Covid-19 as a use case as shown in Figure 6.3a. For the maximum, we consider the *soil moisture index*[1] (SMI), which is an important indicator for droughts; see Figure 6.3b.

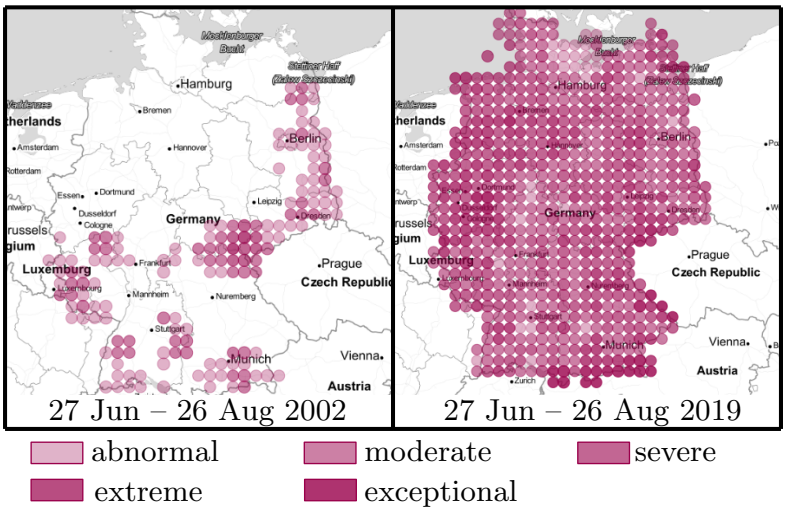
We emphasize that the θ-structure is not restricted to grid-based density maps but can be applied to any partitioning of the map. This is particularly interesting for creating choropleth maps, which use geographic or administrative regions as cells. For example, by mapping the events of the Covid-19 cases on administrative regions, one can use the θ-structure to directly create the choropleth map.

The chapter is structured as follows. After discussing related work in Section 6.2, we present in Section 6.3 a formal model of the θ-structure for one color. In Section 6.4 we show how queries can be accelerated using fractional cascading. In Section 6.5 we generalize the θ-structure to both a discrete set of colors. Finally, in Section 6.6 we present our experimental evaluation on real-world data.

[1] For further reading and a detailed interpretation of the SMI we refer to `https://www.ufz.de/droughtmonitor`.



(a) Covid-19 cases



(b) droughts

Figure 6.3: Examples of other data sets. (a) Maps showing the total number of Covid-19 cases. (b) Maps showing the maximum *soil moisture index*. *Map tiles by Stamen Design, under CC BY 3.0. Map data by OpenStreetMap, under ODbL.*

6.2 Related work

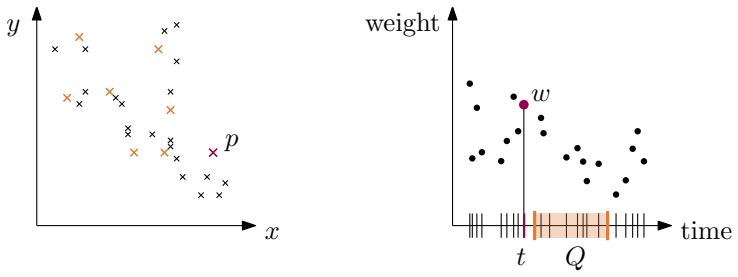
For an overview of point set representations, tree data structures, fractional cascading, and time-windowed data structures, we refer to Chapter 2 and Chapter 3, respectively.

6.3 θ-Structure

In the following, we introduce the theoretical model for time-windowed density maps. Based on this we introduce the θ-structure. We further explain how to query this data structure and provide a construction and update algorithm.

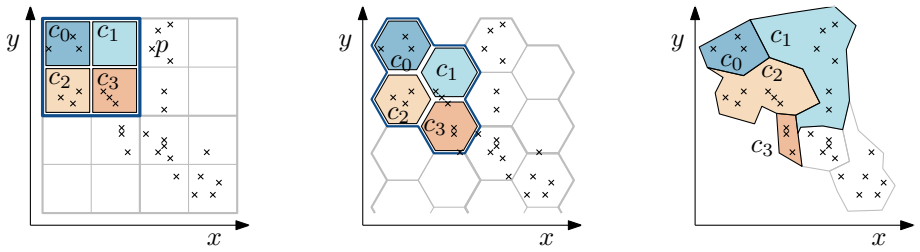
Model. Let P be a set of n points in $\mathbb{R}^2$ where each point $p \in P$ is associated with a timestamp $t \in \mathbb{R}_{\geq 0}$ and a weight $w \in \mathbb{R}_{\geq 0}$. We call the triplet $e = (p,t,w)$ an *event* and denote the set of all events by $E = \{e_1, \ldots, e_n\}$; see Figure 6.4. We assume that the points are contained in a simple polygon M which represents the map. Further, let $G = \{c_1, \ldots, c_m\}$ be a set of m polygons that partition M; see Figure 6.5. We call $c \in G$ a *cell*. For grid-based density maps, M is the bounding box of P and the cells of G are equally sized, axis-aligned rectangles that cover P. On the other hand, for data related to administrative borders, it might be useful for the visualization to let P be the set of the administrative regions. For a given partition G, each event is contained in exactly one cell $c \in G$ and we denote the set of all events of cell c by $E_c \subseteq E$.

For our application, we compute the density map for a time-window query Q. More precisely, let $Q = [t_1, t_2]$ be a *time-window query* with $t_1 \leq t_2$. Further, we denote the set of all events that are contained in cell $c \in G$



(a) spatial setting (b) weights and time

Figure 6.4: The events are given as a point set. Each point p is annotated with a timestamp t and optionally with a weight w. The core problem is to evaluate for an arbitrary query Q all points (orange) whose timestamps lie in Q.



(a) rectangular grid (b) tessellation (c) arbitrary

Figure 6.5: Partitions of the map for various types of cells.

and in the time window Q by $E_{c,Q}$, i.e., for each $(p,t,w) \in E_{c,Q}$ it holds $p \in c$ and $t \in Q$. To color the cells of the density map, we are given a decision function $D\colon G \times \mathbb{R}^2 \to \{\text{true}, \text{false}\}$, which decides for a cell $c \in G$ and a query $Q = [t_1, t_2]$ whether c is colored or not; we say that a cell $c \in G$ is *colored* for Q if $D(c,Q) = \text{true}$. The density map for decision problem D and time-window query Q displays all cells c for which $D(c,Q) = \text{true}$.

In this work, we restrict ourselves to decision problems for which $D(c,Q) = \text{true}$ implies $D(c,Q') = \text{true}$ for all queries Q and Q' with $Q \subseteq Q'$. In that case, we call D a *monotone decision problem*. In the following, we list examples of monotone decision problems:

- **Counting Problem.** The decision problem D_{count} answers the question of whether the number of elements in the time window Q and region c is larger than a prescribed threshold $\theta \in \mathbb{R}_{\geq 0}$.

$$D_{\text{count}}(c,Q) = \begin{cases} \text{true} & \text{if } \sum_{e \in E_{c,Q}} 1 \geq \theta, \\ \text{false} & \text{else.} \end{cases} \tag{6.1}$$

  This is a monotone decision problem, since the number of elements in $E_{c,Q'}$ is always larger or equal to the number of elements in $E_{c,Q}$ or any $Q \subseteq Q'$.

- **Sum Problem.** The decision problem D_{sum} answers whether the sum of all weights in Q and region c is larger than a threshold θ.

$$D_{\text{sum}}(c,Q) = \begin{cases} \text{true} & \text{if } \sum_{(p,t,w) \in E_{c,Q}} w \geq \theta, \\ \text{false} & \text{else.} \end{cases} \tag{6.2}$$

  It is a monotone decision problem since all weights are non-negative and hence for a query Q' that contains Q the sum is always larger than the sum for Q.

- **Maximum Problem.** The third example for monotone decision problems answers the question if there exists any event in $E_{c,Q}$ for which the weight is larger than the threshold θ.

$$D_{\max}(c,Q) = \begin{cases} \text{true} & \text{if } \max\{w \mid (p,t,w) \in E_{c,Q}\} \geq \theta, \\ \text{false} & \text{else.} \end{cases} \tag{6.3}$$

  This is a monotone decision problem since by enlarging the time-window query the maximal weight cannot decrease. Similarly, the problem of whether a weight smaller than a threshold exists in a time window is also a monotone decision problem.

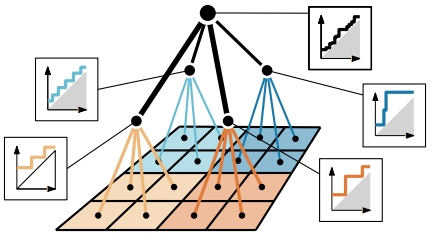


Figure 6.6: The θ-structure is a search tree built on top of the given grid. Each node is associated with a time structure that is queried to early exclude cells that are not reported.

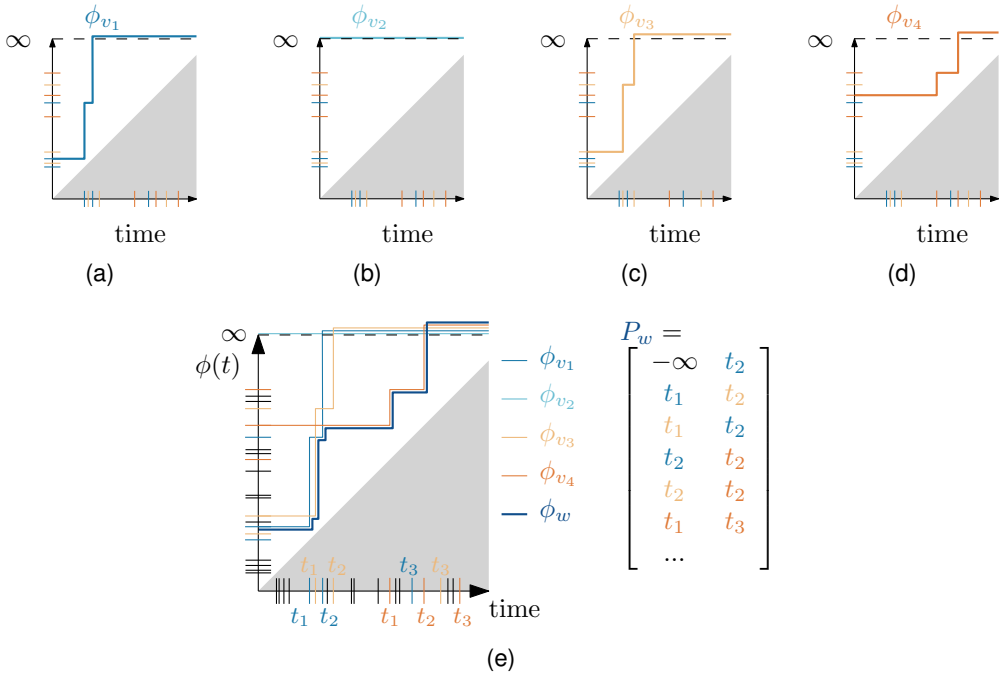


Figure 6.7: Time functions for D_{count} with $\theta = 2$ stored at each node of the θ-structure. (a)-(d) Time function for leaf nodes v_1, v_2, v_3, v_4 of Figure 6.5. (e) Time function for internal node w and its child nodes v_1, v_2, v_3, and v_4.

An example of a non-monotone decision problem is D_{avg}, which answers whether the average of the weights of all events contained in a cell and a time window is larger than some value θ. This is not monotone since the average of $\{w \mid (p,t,w) \in E_{c,Q}\}$ can be larger than the average of $\{w \mid (p,t,w) \in E_{c,Q}\}$ with $Q \subset Q'$.

For a given monotone decision problem D we define the *time function* $\Phi_c(t)$ of the cell c such that it provides for a query start time t the earliest point in time for which D is fulfilled; see Figure 6.7(a) and Figure 6.7(b). For a time window $Q = [t_1, t_2]$ the polygon c is colored if $\Phi_c(t_1) \leq t_2$. Note that the region $\{(t_1,t_2) \in \mathbb{R}^2 \mid \Phi_c(t_1) \leq t_2\}$ is the *activity region* of cell c. A simple approach for the computation of a density map for a time-window query Q is to query the time function of each of the m cells in G. This requires $O(m \log n)$ time.

In the following, we present a data structure, which we call θ-structure, that can be queried in $O(d \log m \log n)$ time, where d is the number of colored cells. In Section 6.4 we improve the query time to $O(d \log m + \log n)$. This data structure builds a tree on top of the grid such that each leaf contains one cell; see Figure 6.6. We assume that the tree has height $O(\log m)$; in our experiments, we used a standard quadtree for this purpose [de Berg et al., 2008]. The core idea is to associate each node with a time structure that allows us to efficiently exclude cells that are not reported for the given query.

We now describe the details of the θ-structure. Let H be a tree that contains for each cell $c \in G$ a leaf and has height $O(\log m)$; in the case that G is a rectangular grid structure, we use a quadtree. For each node $w \in H$ we define a *time function* $\phi_w(t)$ as follows. At first, assume w is a leaf node of H. Then, the node w corresponds to a cell $c \in G$ and we set $\phi_w(t) = \Phi_c(t)$ for all $t \in \mathbb{R}$. In the case that w is an internal node, let $v_1, \ldots, v_k$ be the child nodes of w in H. The time function ϕ_w of w is the minimal value of the time functions of $v_1, \ldots, v_k$ for each t; see Figure 6.7.

$$\phi_w(t) = \min\{\phi_{v_i}(t) \mid 1 \leq i \leq k\} \tag{6.4}$$

Hence, $\phi_w(t)$ describes the first query end time for which at least one descendant leaf cell is colored.

For our data structure, we store $\phi_w(t)$ in each node w of H. More precisely, let $t_1,\dots,t_I$ be the set of all timestamps where each t_i with $1 \le i \le I$ is contained in a descendant leaf node of w and ordered such that $t_i \le t_j$ for $1 \le i \le j \le I$. We denote the sequence $\langle -\infty, t_1, \dots, t_I, \infty \rangle$ by T_w. We store ϕ_w as a table P_w, where the first column corresponds to T_w and the second column contains $\phi_w(t)$ for each $t \in T_w$. We call P_w the *timetable* of w. Altogether, we obtain a search tree H whose nodes are annotated with timetables; we call H a *θ-structure*.

Querying For a time-window query $Q = [t_1, t_2]$ we descend in the θ-structure H starting at the root node of H. Let w be the currently considered node of H. If $\phi_w(t_1) \le t_2$ and w are a leaf node, we report the cell of w. Otherwise, if w is an internal node, we descend to its child nodes. In case $\phi_w(t_1) > t_2$ we stop the search in this branch of H because there is no leaf node in the subtree of w that fulfills the decision problem.

Descending from the root to one leaf node takes $O(\log m)$ time. Deciding for a node w whether $\phi_w(t_1) > t_2$ is fulfilled, takes $O(\log n)$ time. Let d be the number of reported cells. Hence, overall querying takes $O(d \log m \log n)$ time.

Construction and Update We construct the θ-structure in a bottom-up approach. First, we create a tree H on top of the grid G, i.e., the leaves of H contain the cells. We create for each cell $c \in G$ its time function Φ_c by sorting the events E_c in chronological order and by applying one linear sweep to obtain $\Phi_c(t)$ for each $(p,t,w) \in E_c$. We note that Φ_c is constant at any other time that does not coincide with a timestamp of an event. For each leaf u of H we set $\phi_u = \Phi_c$, where c is the cell of u. For an internal node we create ϕ_u by *merging* the time functions $\phi_{v_1},\dots,\phi_{v_k}$ of its child nodes $v_1,\dots,v_k$. More precisely, we compute $\phi_u(t) = \min\{\phi_{v_i}(t) \mid 1 \le i \le k\}$ with $0 \le t \le \infty$, which is the lower envelope of the time functions $\phi_{v_1},\dots,\phi_{v_k}$. Utilizing a linear sweep on the events of $v_1,\dots,v_k$ in increasing order of their timestamps, we create ϕ_u in linear time of the number of events contained in the sub-tree of u. As the cells are disjoint, in each level of H we need $O(n)$ time in total to create the time functions. Hence, we need $O(n \log n)$ for once sorting the events and then $O(n \log m)$ time to create H on top of G. Note, that we assume that the time for solving the decision problem itself is neglectable, e.g., it may not be NP-hard. The tree uses $O(n)$ space per level and has m leaves so that it uses $O(n \log m + m)$ space in total.

Theorem 3. *The θ-structure takes $O(n \log m + m)$ space, can be constructed in $O(n(\log n + \log m))$ time, and can be queried in $O(d \log m \log n)$ time, where d is the number of cells that fulfill D.*

In some of the use cases, e.g., the spreading of Covid-19, the data is not known in advance but it is updated on a daily basis. In particular, the new events occur after the events that are already contained in the θ-structure H. We shortly sketch how to update H. For each new event $e = (p,t,w)$ we identify the cell c and its leaf node u in H. We update ϕ_u for e and determine any event $e' = (p',t',w')$ for which $\phi_u(t')$ has been changed. We observe that this is only the case if $\phi_u(t') = \infty$ held before. Further, for each such event e' and for the event e we follow the path P from u to the root of H. At each encountered node v on this path, we update the time function $\phi_v(t)$ with respect to the currently considered event. Each of these updates can be performed in $O(\log n)$ time per node by updating the lower envelope, correspondingly. Altogether, inserting a single event may cost $O(n \log n \log m)$ time, but each event e' triggers such updates only once. Hence, over all events, we obtain an amortized running time $O(\log n \log m)$ per event.

6.4 Fractional Cascading

Fractional cascading is a speed up technique for searching [Chazelle and Guibas, 1986a,b, de Berg et al., 2008]; see Chapter 3. It does not increase the space consumption, but a search takes $O(J + \log n)$ time instead of $O(J \cdot \log n)$ time for J reported elements and n is the overall number of elements. In the following, we discuss how to incorporate fractional cascading into the θ-structure. For the θ-structure we are given a node w with its timetable P_w and a child node v with the timetable P_v. Due to the construction of P_w and P_v the arrays are sorted and $P_v \subseteq P_w$; see Figure 6.8. We extend the timetable P_w with a column for the child node v that contains for each timestamp $t \in T_w$ a pointer to the row of P_v of the first timestamp that is equal to or larger than t. When querying P_w for a query $Q = [t_1, t_2]$, we first perform a binary search in P_w for t_1 and if $\phi_w(t_1) < t_2$ we directly evaluate $\phi_v(t_1)$ by using the pointer instead of performing a second binary search for t_1 in P_v. The next theorem summarizes the results.

Theorem 4. *The θ-structure with fractional cascading takes $O(n \log m + m)$ space, can be constructed in $O(n(\log n + \log m))$ time, and can be queried in $O(d \log m + \log n)$, where d is the number of cells that fulfill D.*

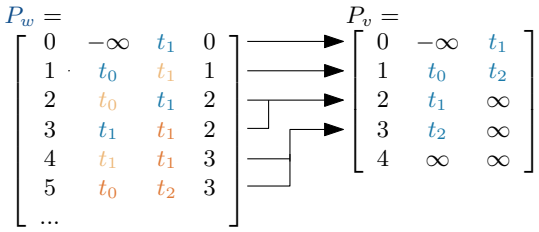
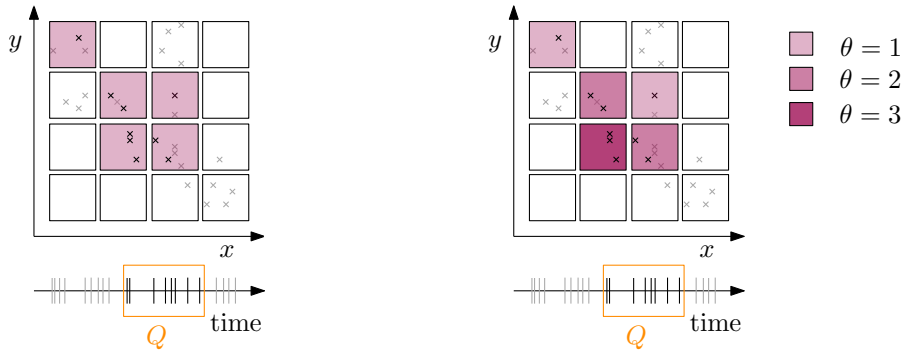


Figure 6.8: Illustration of fractional cascading. The timetable of each node w is extended by an additional column for each child node v with pointers to the rows of the timetable of v.



(a) one color problem

(b) multi-color problem

Figure 6.9: Example of different types of density map.

6.5 θ-Structure with Multiple Colors

Instead of using one color often multiple colors are desired to visualize the information; see Figure 6.9. For example, for the counting problem, we want to color-code the number of events in the time window for all cells that exceed at least a minimal threshold. This can be done by visualizing the results of the counting problem for several thresholds.

Choosing the color of the cell can be seen as a second step in the query phase, i.e., in the first step we collect all cells to be colored using the θ-structure, and in the second step, we then decide which color is used for the cell. To that end, we assume that for each cell c we are given a color function $f\colon G \times \mathbb{R}^2 \to \{1,\dots,h\}$ that determines for a cell c and a time-window query $Q \subseteq \mathbb{R}^2$ one of h color values. More precisely, $f(c,Q)$ is a color in the range from 1 to h if $D(c,Q) = \text{true}$. In case $D(c,Q) = \text{false}$, the value of f is undefined, which we denote by $f(Q) = \perp$. We distinguish two versions that differ in the supported decision functions.

Version 1: counting, sum and maximum problems We first describe an approach that only works for decision problems based on quantities such as the *counting*, *sum*, and *maximum* problem, but not on monotone decision problems in general. In the following, we first explain the approach for the decision problem D_{sum} and then explain the adaptions for D_{count} and $D_{\max}$. We assume that the color $f(c,Q)$ for a cell c and a query Q is defined as follows.

$$f(c,Q) = \begin{cases} \perp & \text{if } D_{\text{sum}}(c,Q) = \text{false}, \\ T[\sum_{(p,t,w)\in E_{c,Q}} w] & \text{otherwise}, \end{cases} \tag{6.5}$$

where T is a simple lookup-table that maps a value $W \in \mathbb{R}$ to one of the h colors, i.e., $T[W] \in \{1,\dots,h\}$. Hence, the computational problem reduces to determining the value of the sum $\sum_{(p,t,w)\in E_{c,Q}} w$.

To that end, we enrich the θ-structure H for the decision problem D_{sum} as follows. For each leaf node v of H we additionally store its events E_v in a classical 1-dimensional range tree R_v [de Berg et al., 2008]. More precisely, R_v is an ordered binary tree that stores each event $e \in E_v$ in one of its leaves such that the events appear in their temporal order when traversing the leaves from left to right (i.e., doing a pre-order traversal on R_v). Further, each internal node contains the event of the rightmost leaf of its left sub-tree. Let u be an arbitrary node of R_v and let $w_1,\dots,w_k$ be the weights of the events that occur in the subtree of u. We annotate u with the weight $w_u = \sum_{i=1}^{k} w_i$.

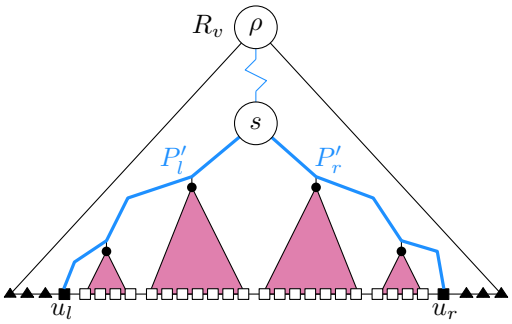


Figure 6.10: Illustration of a range tree R_v. The path from ρ to u_l and the path from ρ to u_r have a common prefix that ends at s. The suffixes P'_l and P'_r separate the queried events (squares) from the others (triangles).

When answering a time-window query $Q = [t_1, t_2]$ on H, we perform the same procedure as described in Section 6.3 to identify all cells that are colored for Q. For each such cell c we then perform the following procedure to obtain $\sum_{(p,t,w)\in E_{c,Q}} w$. Let R_v be the range tree that is stored in the leaf of v that represents c. Starting at the root ρ of R_v we search for the leftmost node u_l of R_v with timestamp $t_1 \leq t_l$ and for the rightmost node u_r of R_v with timestamp $t_r \leq t_2$; see Figure 6.10. Let P_l be the path that connects ρ with u_l and let P_r be the path that connects ρ with u_r. After a (possibly empty) prefix starting at ρ the paths P_l and P_r split at a common node s in R_v. Let P'_l and P'_r be the remaining suffixes of P_l and P_r that start at s, respectively. We collect all right child nodes of the nodes in P'_l and all left child nodes of the nodes in P'_r that neither belongs to P'_l nor to P'_r; we denote the set of the collected nodes by C (see black dots in Figure 6.10). For c we then return $f(c) = T[\sum_{u\in C} w_u + w_{u_r} + w_{u_l}]$ as color.

We treat the counting problem as a special case of the sum problem by defining the weight of each event as 1. For the maximum problem, we store at each internal node of R_v the weight $w_u = \max\{w_w \mid w \text{ is a child node of } u\}$. Hence, in that case we return $f(c) = T[\max\{w_u \mid u \in C \cup \{u_l, u_r\}\}]$ as color. Other decision problems based on quantities can be solved similarly.

Assuming that a lookup in T takes $O(\log h)$ time, the time-window query runs in $O(d\log m + d\log n + d\log h)$ time: $O(d\log m + \log n)$ time for collecting the d cells to be colored and $O(d\log n + d\log h)$ time for evaluating the color function for all d cells. Further, in addition to the $O(n\log m + m)$ space for the θ-structure we need $O(n+h)$ space for the range trees and lookup table in total.

Theorem 5. *The multi-color θ-structure for quantitative problems takes $O(n\log m + m + h)$ space, can be constructed in $O(n(\log n + \log m))$ time, and can be queried in $O(d\log m + d\log n + d\log h)$, where d is the number of colored cells and h is the number of colors.*

We note that we can define the color pallet on demand in the query phase, as long as we keep the first threshold fixed. Further, instead of using a lookup table, we can use a continuous color pallet by applying a linear transformation. Moreover, for the counting and sum problem, we can improve the query time to $O(d\log m + \log n + d\log h)$. To that end, we store in each cell c all events in an array A_c in increasing order. Further, for each event, we store the cumulative weight up to that event. When answering a query Q on the θ-structure we use fractional cascading to find the first and last event of Q in A_c for each cell c. The difference of their cumulative weights is then the desired value $\sum_{(p,t,w)\in E_{c,Q}} w$.

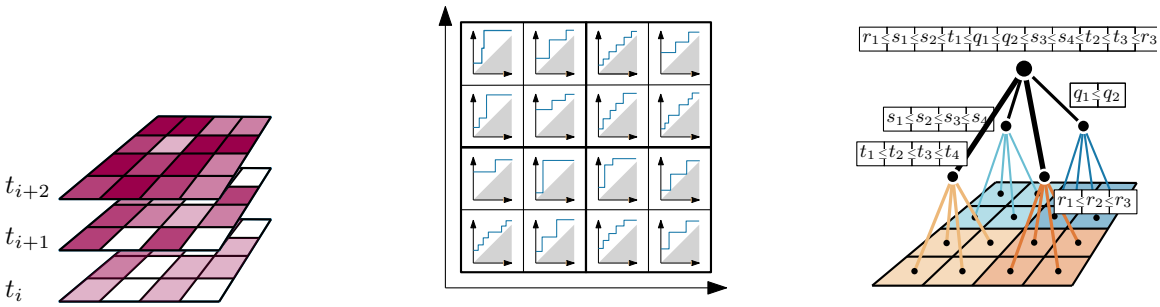
Version 2: general monotone decision problems We now present an approach that supports any monotone decision problem. More precisely, for each color $i \in \{1,\dots,h\}$ we are given a decision problem D_i, which decides whether a cell c can be colored with i for a given query Q. As we want to obtain a monotone color gradient, we require for each color i with $1 \leq i < h$ that $D_{i+1}(c,Q) = \text{true}$ implies $D_i = \text{true}(c,Q)$. If a cell $c \in G$ and a query Q fulfills D_i but not D_{i+1}, the color function f assigns the color i to c, i.e.,

$$f(c,Q) = \begin{cases} \bot & \text{if } D_1(c,Q) = \text{false for } 1 \leq i \leq h \\ 1 & \text{if } D_1(c,Q) = \text{true},\ D_i(c,Q) = \text{false for } 2 \leq i \leq h \\ \vdots & \\ h & \text{if } D_i(c,Q) = \text{true for } 1 \leq i \leq h \end{cases} \tag{6.6}$$

In order to query f on G, we adapt the θ-structure H by extending the timetable of all leaf nodes of H. Let v be a leaf node with event set F and timetable P. Let $\phi_v^1, \phi_v^2, \dots, \phi_v^h$ be the time functions for the event

Table 6.1: Properties of variants of the θ-structure: one color with and without fractional cascading (FC) and the two versions for multiple colors. n =number of events, m =number of cells, d=number of reported cells, h =number of colors.

	Construction	Storage	Query Time
One Colour:			
without FC	$O(n(\log m+\log n))$	$O(n\log m+m)$	$O(d\log m\log n)$
with FC	$O(n(\log m+\log n))$	$O(n\log m+m)$	$O(d\log m+\log n)$
Multiple Colours:			
version 1	$O(n(\log m+\log n))$	$O(n\log m+m+h)$	$O(d\log m+d\log n+d\log h)$
version 2	$O(n(\log m+\log n+h))$	$O(n\log m+m+nh)$	$O(d\log m+\log n+d\log h)$



(a) alternative 1 (b) alternative 2 (c) alternative 3

Figure 6.11: Illustration of alternative approaches.

set F concerning the decision problems $D_1, D_2, \ldots, D_h$, respectively. We observe that $\phi_v^1(t) \le \phi_v^2(t) \le \ldots \le \phi_v^h(t)$ for $t \in \mathbb{R}$. We extend P with a column for each time function $\phi_v^2, \ldots, \phi_v^h$. Hence, this data structure has size $O(n\log m+mh)$; assuming that $h \in O(1)$ we obtain $O(n\log m+m)$ storage consumption.

For a time-window query $Q=[t_1,t_2]$, we first search in H for all leaf nodes that fulfill D_1 using fractional cascading; see Section 6.4. Let v be a leaf node that fulfills D_1 and let t be the first timestamp of T_v that is equal to or larger than t_1. We perform a binary search in the sorted array $\phi_v^1(t), \ldots, \phi_v^h(t)$ to determine the color i for which $D_i = \text{true}$ and $D_{i+1} = \text{false}$. The query time is $O(d\log m+\log n+d\log h)$ where d is the number of nodes that fulfill D_1.

Theorem 6. *The multi-color θ-structure for arbitrary monotone decision problems with h colors has $O(n\log m+m+nh)$ space, can be created in $O(n(\log n+\log m+h))$ time and queried in $O(d\log m+\log n+d\log h)$ time, where d is the number of colored cells.*

We can use this setting to visualize more advanced sequences of decision problems. In the running example of bird observations, the first decision problem could be about the number of birds, the second about the number of gulls, and the third about the number of Larus argentatus, i.e. a specific species of gulls.

6.6 Evaluation and Experiments

In this section, we present the evaluation of our approach. Table 6.1 gives an overview of the properties of the variants of the θ-structure.

6.6.1 Comparison with Alternative Approaches

It is tempting to improve the query time, by defining the data structure differently. However, this easily leads to an increase in storage consumption. We discuss three apparent alternatives for our approaches showing that the balance between storage consumption and the query time is impaired. We assume that the events are given as a sequence $e_1, \ldots, e_n$ ordered by their timestamps $t_1, \ldots, t_n$.

Alternative 1. We construct n density maps, namely for each timestamp t_i one map M_i; see Figure 6.11a. For each cell, we store the cumulative weights of the events up to this timestamp. For example, for the counting

Table 6.2: The data sets used in the experiments. For each data set the tables show the number of events, the number of grid cells, the cell size in km, whether the events have weights, and the first and last timestamp, as well as the resolution of the data.

Data	BIRDDATA	COVID19DATA	DROUGHTDATA
Events	$8.47 \cdot 10^6$	$0.34 \cdot 10^6$	$19.21 \cdot 10^6$
Grid Cells	648×393	752×2037	139×102
Cell Size	10×10	10×10	10×10
Weights	no	yes	yes
First Event	17/05/2013	22/01/2020	16/01/1951
Last Event	31/08/2016	17/06/2020	16/12/2019
Resolution	days	days	months

problem, we store the number of events that have occurred so far in this cell up to this timestamp. For a time-window query $Q = [t,t']$ we first determine the two density maps M_i and M_j such that t_i is the earliest event with $t_i \leq t$ and t_j is the latest timestamp with $t' \leq t_j$. Afterward, we decide for each cell in G and its cumulative weights w_i and w_j in M_i and M_j whether $w_j - w_i$ exceeds the given threshold. If this is the case, we report that cell. We observe that this only works for the sum and counting problem, but not for the maximum problem. Determining M_i and M_j needs $O(\log n)$ time and traversing all cells needs $O(m)$ time, which leads to $O(m + \log n)$ query time. Due to its high storage consumption, which lies in $\Theta(m \cdot n)$, this alternative is most suitable for event sets that are based on a few timestamps (e.g., ,any events occur at the same time). Further, as the running time is independent of the number of reported cells, it is mostly suitable for data that most of the queries report almost all cells. However, for heterogeneous data sets, e.g., the data set on bird observations, the θ-structure clearly prevails over this alternative due to its output-sensitive query time and its better space consumption.

Alternative 2. We refrain from building the search tree upon the grid, but only consider the grid such that each cell $c \in G$ has a time function ϕ_c; see Figure 6.11b. For a time-window query Q we iterate through all cells of G and evaluate each time function. Compared to the θ-structure the storage consumption decreases by a logarithmic factor to $O(n+m)$ and the query, which lies in $O(m \log n)$, is not output-sensitive anymore. In our experiments, we show that the superimposed search tree is a decisive speed-up.

Alternative 3. Based on the grid cells we build a search-tree $\mathscr{T}$ that has for each cell $c \in G$ a leaf that contains the timestamps of c in increasing order; see Figure 6.11c. Further, each internal node of $\mathscr{T}$ contains the timestamps of its child nodes in increasing order. For a time-window query $Q = [t,t']$ we start at the root of $\mathscr{T}$. We determine for a node v of $\mathscr{T}$ the earliest timestamp τ with $t \leq \tau$ and the latest timestamp $\tau' \leq t'$ of v. If the number of events in v between τ and τ' does not exceed the given threshold, we stop the search for this node. Otherwise, if v is an internal node, we descend to its child nodes. If v is a leaf, we report the corresponding cell. For supporting the sum problem we store for each event its cumulative weight, i.e., the weights of all events in the same node up to that timestamp. Altogether, the data structure has the storage consumption $O((n+m)\log m)$. Utilizing binary searches and fractional cascading a query can be answered in $O(m + \log n)$ time. We observe that, in the worst case, the search considers all leaves even if the number of output cells is 0. Hence, this alternative is not output sensitive. Further, the maximum problem is not supported by this structure. Altogether, it is not a replacement for the θ-structure.

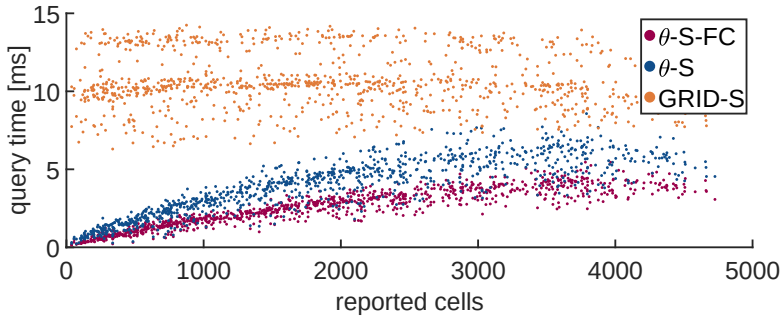
6.6.2 Experiments

In this section, we describe the experiments on real-world data analyzing the query times of the θ-structure.

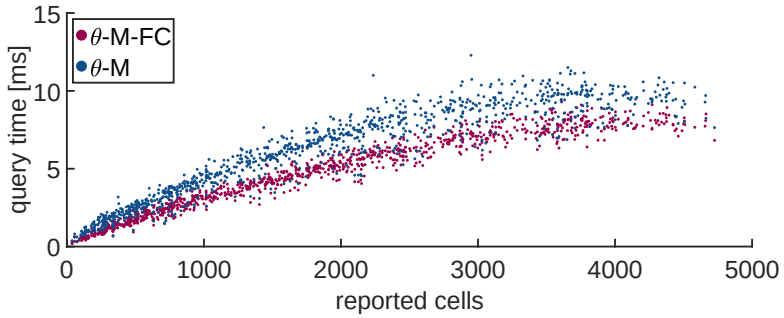
Experimental Setup. We have considered three data sets that differ in size, extent, and type. Table 6.2 gives an overview.

- BIRDDATA. The data set consists of events that represent the occurrences of single birds. The data is derived from the *Global Biodiversity Information Facility*[2] [Stienen et al., 2017].
- COVID19DATA. The data set describes the spreading of the Covid-19 disease. The data was collected from multiple sources for the countries *Germany*, *Italy*, *France*, *England*, and *USA*. Each event represents the number of Covid-19 infections that were reported on one day in one of the administrative regions of

[2] gbif.org



(a) one color problem and BIRDDATA



(b) multi-color problem and BIRDDATA

Figure 6.12: Query time for the counting problem.

the countries. The data comprises 3660 administrative regions in total. For more details and the list of sources, we refer to `www.geoinfo.uni-bonn.de/densitymaps`.

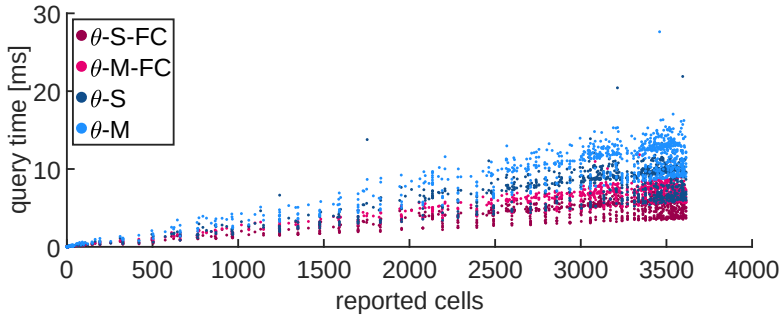
- DROUGHTDATA. The data set describes the soil moisture index (SMI) observed in Germany. This index is an indicator for droughts. It is given as raster data, which we have translated into point data by considering the centers of the cells. Despite its rasterized appearance, we consider this data set in our experiments as it is large and each event has a weight.

For the density maps, we used a grid to partition the bounding box of the data. Each cell has a size of 10 km $\times$ 10 km, which is finer than the cell sizes used for illustration in Figures 6.1–6.3.

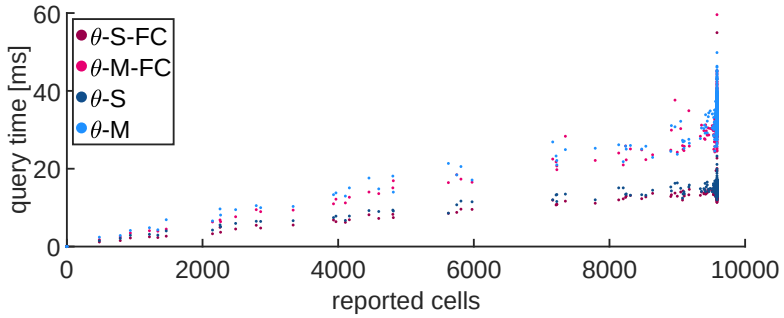
In the following evaluation, we focus on four versions of the θ-structure: the θ-structure with a single color (θ-S), the θ-structure with a multi-color pallet (θ-M), the θ-structure with fractional cascading and a single color (θ-S-FC), and the θ-structure with fractional cascading and a multi-color pallet (θ-M-FC). For each of the data sets, we computed the θ-structure for each of the four variants, whereas we solved the counting problem for the BIRDDATA, the sum problem for the COVID19DATA, and the maximum problem for the DROUGHTDATA. Since for the DROUGHTDATA the SMI is defined between 0 and 1, where 0 means exceptional drought and 1 that there is no drought, we inverted the scale and used $1 - \text{SMI}$ for the weights of the data. We define the thresholds for each data set as θ_{BIRDDATA}=10, $\theta_{\text{COVID19DATA}}$=10 and $\theta_{\text{DROUGHTDATA}}$= 0.7. For the multi-color problem, we used a lookup table with six colors.

We implemented the data structure in Java and ran the experiments on an Intel(R) Xeon(R) W-2125 CPU clocked at 4.00GHz with 128 GiB RAM. We considered 1100 randomly chosen time-window queries. To prevent influences of the warm-up phase of the virtual machine for Java, the first 100 queries are not considered in the evaluation. Further, we removed six outliers from the query measurements (>85ms) which we also explain with the use of Java.

Counting problem (BIRDDATA) For the counting problem, we evaluate the query time first for the θ-structures with one color and afterward for those with multiple colors. In order to prove the necessity of a more advanced data structure, we implemented a simple solution that consists of a grid where each grid cell is enhanced by an unsorted list of timestamps of the contained events. For a query, we search in all grid cells and iterate over the whole time list. The query time for this variant varies between 182.1 and 408.8 ms. Hence, this is not a suitable solution for interactive visualizations and motivates more elaborated approaches.



(a) sum problem and COVID19DATA



(b) maximum problem and DROUGHTDATA

Figure 6.13: Query time for the sum and maximum problem.

Figure 6.12a shows the query times for θ-S, θ-S-FC for the test queries, which are ordered by the number of reported grid cells. We compare the θ-structures to an alternative approach, which we call *GRID-S*. This approach consists of a grid, where each grid cell is augmented by the time function; see Subsection 6.6.1 – Alternative 2. The experiments show that the query time of GRID-S is in the range of 6.3 and 14.3 ms and that it does not depend on the number of reported cells. For θ-S and θ-S-FC the query time shows a clear dependency on the number of reported cells. Hence, for time-window queries with small output, the query time is substantially smaller but also for large time windows we observe a clear difference. For θ-S the queries take maximal 8.6 ms and for θ-S-FC 5.5 ms. On average the query time of θ-S reduces to 36.4% and the query time of θ-S-FC reduces to 23.9% of the query time of GRID-S.

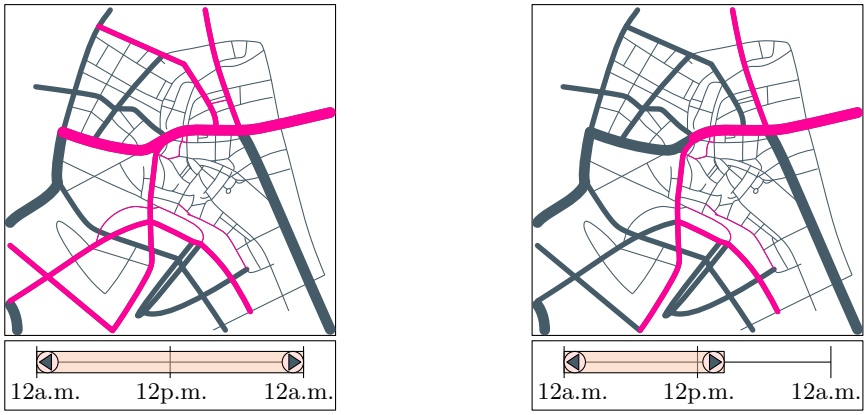
The query times for the multi-color problem are between 0.3 and 12.3 ms for θ-M and between 0.2 and 9.3 ms for θ-M-FC; see Figure 6.12b. This is on average 2.2 ms slower than for the respective θ-structures with one color. The evaluation shows that the θ-structure leads to a substantial improvement of the running time.

The construction for all four data structures θ-S, θ-M, θ-S-FC, θ-M-FC took only between 31 and 33 seconds for the BIRDDATA.

Sum problem (COVID19DATA) For the sum problem, we compare the query times of θ-S, θ-M, θ-S-FC, and θ-M-FC; see Figure 6.13a. The fastest variant is θ-S-FC with 4.1 ms on average and 7.6 ms query time on maximum. The θ-structure without fractional cascading θ-S is 1.8 ms slower than θ-S-FC on average. For the multi-color versions, the query time is 2.2 ms greater on average.

Due to the comparably small size of COVID19DATA, the construction for θ-S, θ-M, θ-S-FC, and θ-M-FC only took 3 to 5 sec.

Maximum problem (DROUGHTDATA) Figure 6.13a shows the measured query times of θ-S, θ-M, θ-S-FC and θ-M-FC. For this data set the number of reported cells is for most time-window queries rather large. This is reasoned in the fact that we solve the maximum problem and further, in the data and the SMI score itself. Also, the query time is compared to the two previously presented data sets, greater which we explain with the fact that this is the largest data set and the number of reported cells is significantly higher. On average, the fastest variant is θ-S-FC with 13.7 ms on average and 55.0 ms query time at maximum. The additional running time obtained by the multi-color pallet is on average 19.2 ms. This is noticeably larger than for the two other data sets BIRDDATA and COVID19DATA because more cells are reported on average. The difference between the θ-structure with and without fractional cascading θ-S-FC and θ-S is comparably small with 1.3 ms.



(a) largest time window (b) small time window

Figure 6.14: The public transportation network and the user interface for two time windows. The road segments that are frequently served are drawn in pink. Between (a) and (b) the users have slid the right boundary of the time window.

For the DROUGHTDATA the construction time of all four variants θ-S, θ-M, θ-S-FC, θ-M-FC is between 64 and 67 seconds.

Summary Overall, we showed for three different real-world data sets of different sizes both an application and an evaluation of the θ-structure. Compared to the alternative approaches, the θ-structure decreases the query time clearly and allows a real-time interaction, e.g., with a time-slider interface. Especially, the fact that the query time is output-sensitive is an advantage of our data structure. Further, we showed that using fractional cascading improves the query time measurably. The multi-color version increases the query time slightly but it is still applicable for interactive visualizations.

6.7 Excursion: θ^*-Structure for Public Transportation Data

In this section, we present the θ^*-structure which is a data structure based on the θ-structure for the visualization of public transportation networks. It is extracted from the paper published at ICC'22 [Bonerath et al., 2023a].

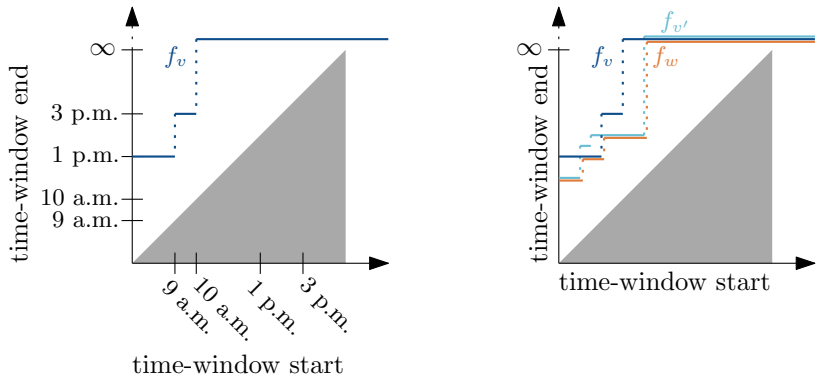
For many people, the public transport network plays a major role in their daily lives. Therefore, careful planning is of utmost importance. This means that decision-makers must be able to easily inform themselves about the spatial and temporal patterns of connectivity. The most common visualizations of public transport data are (1) the schedule tables for transportation lines at each station and (2) a map with the routes of all transportation lines. Although both visualizations work well for certain applications, e.g., informing users of their next route, they do not provide a high-level overview of the spatiotemporal patterns of the entire network. We focus this work on bus networks but it can be easily extended to networks that also contain trams, etc.

In this work, we look at an interactive visualization approach that enables users to explore the spatiotemporal patterns of the public transportation network.

1. **spatial patterns:** Which parts of the city are served, i.e. , which roads are traversed by buses?
2. **temporal patterns:** In which time windows are which parts of the city served?

Driven by these two criteria, we introduce the concept of frequently served road segments, i.e., we say a road segment is *frequently served* for a time window if it is traversed by at least θ buses in the time window. We consider θ to be a given threshold for the network. When a user explores the public transportation network, we display all frequently served road segments for the queried time window. In a more enhanced variant of the visualization, one can either use more than one threshold or encode the number of buses that traversed the road for all roads that have been traversed by more than θ buses.

As described before, we enable the user to explore the network with a time-slider interface; see Figure 6.14. In the following, we give a formal definition of our problem.



(a) time function of Equation 6.7 (b) time function of Equation 6.8

Figure 6.15: Time function. (a) The time function for one leaf node of the θ^*-structure. (b) The time functions of two sibling nodes f_v and $f_{v'}$ and their parent node f_w.

Frequently Served Roads for Time-Window Query.
input:

1. *The road segments R of a public transportation network where each road segment is annotated with the timestamps of bus traversals.*
2. *A pre-defined threshold θ that specifies the minimum number of bus traversals such that a road segment is frequently served.*
3. *A user-defined time-window query $Q = [t, t']$.*

Output:

The set of all road segments R_Q that were traversed by at least θ buses in the time window Q.

In order to allow a pleasant user experience, when exploring the frequently served road segments of a city, a user should be able to receive the visualization in real time for a time-window query. Motivated by movies with a frame rate of 24 images per second, we want to achieve a response time of roughly 40 milliseconds.

To solve this problem of real-time response, we contribute a new version of the θ-structure that enables a real-time exploration with the range-slider. We call this data structure θ^*-structure.

In contrast to the θ-structure, our θ^*-structure is a binary tree. We have adopted the time function for each node. The main contrast to the θ-structure is that there is no straightforward mapping between the road segments and the leaf nodes of the θ^*-structure. We will discuss several versions and (experimentally) evaluate these.

6.7.1 Build a θ^*-Structure

In this section, we assume that we are given a mapping between the leaf nodes $U = \{u_1, \ldots, u_n\}$ of the binary tree $\mathscr{B}$ and the road segments R. In a very simple variant, one could use a random order of the road segments. In order to simplify the notation and without loss of generality, we assume that road segment r_i is mapped to leaf node u_i for $1 \leq i \leq n$.

The *time function* $f_i(t)$ of a leaf node u_i reports for any time-window start time the earliest time-window end time such that the road r_i is frequently served, i.e., it is traversed by θ buses. Again, as for the θ-structure, the region $\{(t', t'') \mid f_i(t') \leq t''\}$ is the activity region of the road segment r_i. Take a road segment with four-timestamps $T_i = \{9\text{ a.m.}, 10\text{ a.m.}, 1\text{ p.m.}, 3\text{ p.m.}\}$ and a threshold $\theta = 3$ as example. Equation 6.7 and Figure 6.15a give the time functions for v.

$$f_i(t) = \begin{cases} 1\text{ p.m.} & \text{if } t \leq 9\text{ a.m.} \\ 3\text{ p.m.} & \text{if } 9\text{ a.m.} < t \leq 10\text{ a.m.} \\ \infty & \text{if } 10\text{ a.m.} < t. \end{cases} \tag{6.7}$$

Now, we introduce the time function f_w of an internal node w of $\mathscr{B}$. Let $u_i, \ldots, u_j$ be the leaf nodes that are descendants of w. We want the time function f_w to report for any time-window start time the earliest time-window end time such that at least one road segment $r \in \{r_i, \ldots, r_j\}$ is frequently served.

We compute the time functions from the bottom of $\mathscr{B}$ to the top. For the leaf nodes, we can compute the time function as described above. For an internal node w, let v and v' be its children for which we have already computed the time functions f_v and $f_{v'}$, respectively. Then, the time-function f_w is the lower boundary of the time

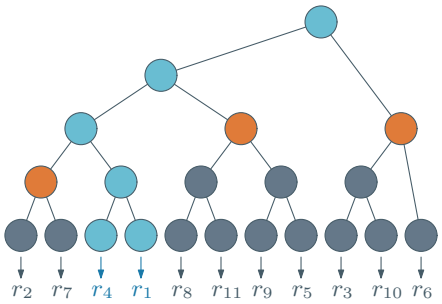


Figure 6.16: The query procedure for a time-window query Q. Blue nodes report `true`, orange nodes report `false`, and grey nodes are not queried. The road segments of the two blue leaf nodes correspond to the frequently served road segments for Q.

Algorithm 2: QUERY

Input: θ^*-structure $\mathscr{B}$, time-window query $[t_{\text{start}}, t_{\text{end}}]$
Output: frequently served road segments R_Q for $[t_{\text{start}}, t_{\text{end}}]$
let v be the root node of $\mathscr{B}$
if v reports `true` for Q **then**
  if v is leaf node of $\mathscr{B}$ **then**
    add corresponding road segment of v to R_Q
  else
    let $\mathscr{B}'$ and $\mathscr{B}''$ be the subtrees of $\mathscr{B}$ rooted at the children of v
    add result of QUERY$(\mathscr{B}', [t_{\text{start}}, t_{\text{end}}])$ to R_Q
    add result of QUERY$(\mathscr{B}'', [t_{\text{start}}, t_{\text{end}}])$ to R_Q

functions f_v and $f_{v'}$; see Figure 6.15b.

$$f_w(t) = \min\{f_v(t), f_{v'}(t)\} \tag{6.8}$$

Overall, we can compute the θ^*-structure in $O(nm \log m)$ time where m is the number of road segments and n is the overall number of timestamps. The θ^*-structure has asymptotic size of $O(nm \log m)$.

6.7.2 Querying a θ^*-Structure

In the following, we describe how we query a θ^*-structure for a time window $[t_{\text{start}}, t_{\text{end}}]$. For a node v of the binary tree, we say that v *reports* `true` *for* Q if $f_v(t_{\text{start}}) \leq t_{\text{end}}$ and otherwise v *reports* `false` *for* Q. For v being an internal node reporting `true` means that there is at least one descendant leaf node of v that reports `true`. For v being a leaf node reporting `true` means that the road segment that corresponds to v is frequently served for Q. The query approach is straightforward: we start at the root node of the tree and traverse it downwards as long as we either reach a leaf node or a node reports `false`; see Algorithm 2.

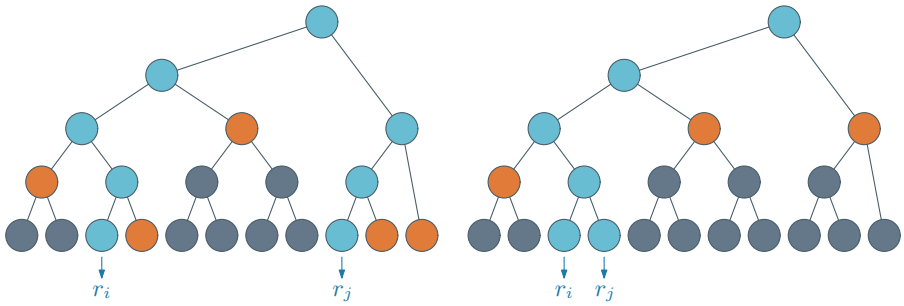
The running time of checking whether a node reports `true` or `false` for a time-window query, depends on the time needed to evaluate $f_v(t_{\text{start}})$. Since f_v is a staircase function, we can store it as an array F_v where each row corresponds to the leftmost point of a horizontal segment of f_v. Equation 6.9 is the array for the time function given in Equation 6.7.

$$F_v = \begin{bmatrix} -\infty & 1\text{ p.m.} \\ 9\text{ a.m.} & 3\text{ p.m.} \\ 10\text{ a.m.} & \infty \end{bmatrix} \tag{6.9}$$

For evaluating $f_v(t_{\text{start}})$, we perform a binary search on the first column of F_v for the first value that is larger than t_{start}. For a data set of m road segments with n timestamps over all road segments, and d road segments that need to be reported for a query Q, the asymptotic query time of Algorithm 2 is $O(d \log n \log m)$.

6.7.3 Building a Good θ^*-Structure

In the following, we discuss how to improve the θ^*-structure such that the query time is reduced. First, we discuss what is a good mapping between the road segments and the leaf nodes. This does not improve the asymptotic query time but in the real-world experiments, we show its real-world relevance. Secondly, we show how to



(a) 13 nodes queried (b) 10 nodes queried

Figure 6.17: Query where road segments r_i and r_j are frequently served. (a) r_i and r_j are mapped to neighboring nodes. (b) r_i and r_j are mapped to nodes that are far apart.

incorporate the technique fractional cascading [Chazelle and Guibas, 1986a,b] which improves the asymptotic and experimentally evaluated query time.

Mapping In the following, we want to analyze which order of the leaf nodes of the θ^*-structure $\mathscr{B}$ is advantageous for the query time. In Section 6.7.2, we described how $\mathscr{B}$ answers a time-window query. Figure 6.17 shows the query procedure for two different mappings between the road segments and the leaf nodes. It shows that a time-window query Q can be answered more efficiently if all frequently served road segments for Q lie closely together. Then, we can exclude large parts of the θ^*-structure in the query procedure. Intuitively, it would be good to find a mapping between the road segments and the leaf nodes of the θ^*-structure such that neighboring leaf nodes correspond to road segments that are frequently served for the same time-window queries. In the following, we provide an approach that we call `max-order` to generate such a mapping. In the experimental evaluation, we compare this to a random order.

For the `max-order` we compute the maximal vertical distance between pairs of time functions f_v and $f_{v'}$. Let $[t_1,t_2]$ be the temporal range where the time functions f_v and $f_{v'}$ are larger than $-\infty$ and smaller than ∞. Then, we define the maximal vertical distance `max-dist` of the time function f_v and $f_{v'}$ as follows

$$\texttt{max-dist}(f_v, f_{v'}) = \max_{t\in[t_1,t_2]} |f_v(t) - f_{v'}(t)| \tag{6.10}$$

For the mapping between the road segments and the leaf nodes, we start with a randomly selected road segment and associate it with the first node v. Then, we compute the maximal vertical distance of f_v and the time functions of all other road segments. We associate the road segment with the smallest maximal vertical distance to the second node w of the leaf nodes. In the next step, we compute the maximal vertical distances to the time function f_w of node w. We associate the road segment with the smallest maximal vertical distance to the third node of the leaf nodes. We repeat this procedure until all road segments are associated to a leaf node.

Fractional Cascading As described for the θ-structure, we also implemented fractional cascading for speed up.

6.7.4 Experiments

In the following, we describe our experiments on real-world data from the public transportation network of Bonn. First, we give an overview of the pre-processing steps that need to be done to transform the data into our θ^*-structure. Secondly, we evaluate the construction and query times obtained with the different versions.

Data Pre-Processing For our experiments, we used the public transportation network of the city of Bonn, Germany obtained by the VRS GmbH[3] under the data license Deutschland Zero Version 2.0. The network contains (i) 6880 stations, (ii) 567 bus, tram, and train routes which are a sequence of stations, and (iii) 121866 trips which consist of the sequence of stations defined by the routes enriched with stop times. Based on the geometries provided by the routes, we processed the data into a graph. To receive the road segments annotated with the bus-traversal timestamps, we performed several pre-processing steps, i.e.,

1. computing the traversal times for the road segments in between stations by linear interpolation
2. merging timestamps of trips where they traverse the same road segment

[3] https://www.vrs.de/

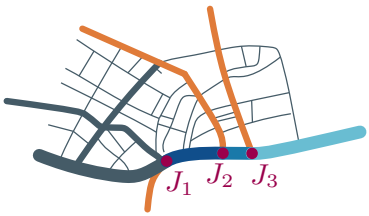


Figure 6.18: Pre-processing of the road network. All grey roads are never traversed by a bus and are neglected. The brown roads are each one road segment. The road that consists of the three blue road segments is split at the junctions J_1, J_2, and J_3. At each junction either a bus line is added to the road segment or leaves the road. Hence, we receive the three blue road segments.

3. partitioning road segments where routes come in or leave the street; see Figure 6.18.

After the pre-processing phase, we have 29159 road segments with at least one, on average 24, and at maximum 2354 bus traversal timestamps. We call this data set `Bonn`.

Although the `Bonn` transportation network is large, there are of course much larger networks. In order to simulate these, we have extended the `Bonn` data set. In more detail: we added the road segments of Bonn, which are not used in the real-world transportation network, and provided them with one to 200 artificial bus traversal times. Thus we end up with 102599 road segments. We call this larger data set `Bonn-Extended`.

Baselines and Versions of the θ^*-Structure In the following, we compare the query times for computing the solution of Problem Frequently Served Roads for Time-Window Query. In particular, we evaluate the different versions of the θ^*-structure that we discussed throughout the chapter

- `theta*-structure`: Here, we assigned the road segments in a random order to the leaf nodes of the θ^*-structure. We build and query the θ^*-structure as described in Section 6.7.1 and Section 6.7.2.
- `theta*-structure-max`: In this version, we assigned the road segments to the leaf nodes according to `max-order` as described in Section 6.7.3. In order to speed up the construction time, we did only compute the maximal vertical distance to 1000 randomly chosen road segments and chose the one with the smallest distance for computing the mapping between road segments and leaf nodes.

We compare these versions of the θ^*-structure to two simple baselines that do not use the θ^*-structure:

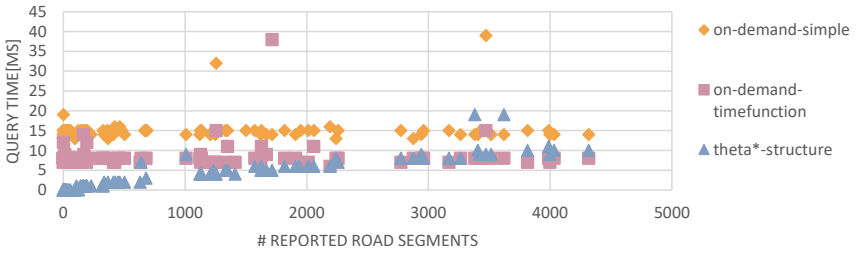
- `on-demand-simple`: Here, we count for each road segment the number of timestamps that are contained in the time window and then, report all road segments that are frequently served.
- `on-demand-timefunction`: Here, we pre-compute the time function for all road segments. For a time-window query, we evaluate the time function of all road segments and report all frequently served road segments on demand.

Construction Time The construction of the θ^*-structure is done in a pre-processing step. Hence, the computation time is not critical for the application. Nevertheless, a short construction time is more pleasant. The construction of `theta*-structure` took 2 seconds for `Bonn` and 20 seconds for `Bonn-Extension`. The construction of `theta*-structure-max` took 232 seconds for `Bonn` and 31 minutes for `Bonn-Extension`.

Query Times We performed the experiments with 100 synthetically generated time-window queries. Figure 6.19 shows the query time evaluation for `Bonn` and Figure 6.20 for `Bonn-Extension`. The experiments show that `theta*-structure` outperforms the two baseline approaches `on-demand-simple` and `on-demand-timefunction`. Especially, for time windows that report smaller numbers of road segments our data structure is more efficient. For the larger data set `Bonn-Extension`, our data structure still shows query times below 25 milliseconds while the two baseline approaches are consistently above 35 and above 60 milliseconds, respectively. Thus, our data structure can allow exploration in real-time (as described in the introduction 40 milliseconds corresponds to 24 frames per second) while this is critical or not possible with the baseline approaches.

6.8 Conclusion

In this chapter, we presented the θ-structure, which can be used for rapidly answering time-window queries for density maps. This enables the user to interactively explore large spatiotemporal data sets and to identify both local and global patterns. In our experiments on real-world data with over 19 million data points and fine-granular grids, we achieved query times in the range of milliseconds, which we deem to be sufficient for interactive use.

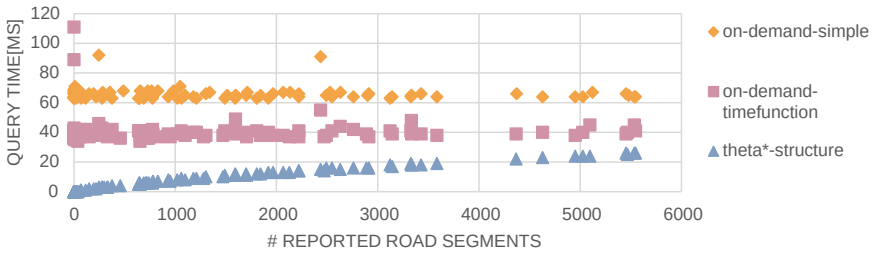


(a) query times for the θ^*-structure and baseline approaches

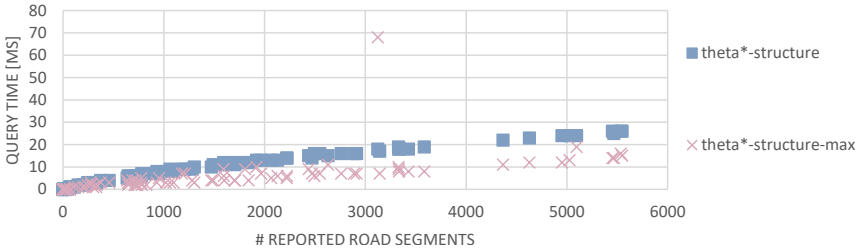
(b) query times for θ^*-structure with and without `max-order`

Figure 6.19: Query time experiments for data set `Bonn`.

We use quadtrees to combine spatial and temporal queries, which allows us to display parts of the map or to support zooming. We have mainly focused on density maps with rectangular grids, but the θ-structure also supports other partitions of the map such as those used in choropleth maps. In particular, we can define for each cell an individual threshold, e.g., depending on the number of inhabitants. We emphasize that we can also use different types of tree structures as shown with the θ^*-structure. Here, we use a binary tree as the basis. As long as the height of the tree is logarithmic in the number of cells, we obtain the same query times.



(a) query times for the θ^*-structure and baseline approaches



(b) query times for θ^*-structure with and without `max-order`

Figure 6.20: Query time experiments for data set `Bonn-Extended`.

7 λ-Structure: Map Labeling for Event Data

In the following chapter, we present results from joint work with Anne Driemel, Jan-Henrik Haunert, Herman Haverkort, Elmar Langetepe, and Benjamin Niedermann which is currently under review. We present a time-windowed data structure for map labeling. The focus of this work is on the stability criteria during basic interactions with the time-slider. We show that the construction of the data structure is NP-hard and provide an approximation algorithm and a greedy heuristic.

Abstract

User interfaces for inspecting sets of spatiotemporal events often use a combination of a map and a time-slider. The time-slider allows the user to specify a time window, which is used to filter the data. Filtered events are visualized on the map, e.g., with textual or iconic labels. Since the number of filtered events can be large and we want to ensure a clear visualization, not all events that have passed the filter are annotated with a label. The task is to decide which points to label for a user-specified position of the time-slider. Drawing upon previous work on map labeling, we assume that every label is associated with a weight expressing its priority and we aim to select a set of labels of maximum total weight while forbidding overlaps between selected labels. In addition to this, we consider the stability of the labeling during certain basic interactions with the time-slider as important for a pleasant user experience. As basic interactions we consider continuously moving the entire time window, symmetrically scaling it, and dragging one of its endpoints. Our approach is to pre-process the events into a data structure that we call λ-structure which consists of an *activity diagram* that encodes which labels are displayed for which time windows. We introduce an optimization problem asking for an activity diagram that maximizes the total weight of the displayed labels over all time windows, subject to constraints forbidding label-label overlaps and ensuring the stability of the labeling. Specifically, we consider two stability requirements: (1) during a basic interaction, a label should appear and disappear at most once; (2) if a label is displayed for a time window Q then it is also displayed for all the time windows contained in Q and containing its timestamp. As finding such an activity diagram is NP-hard, we propose efficient constant-factor approximation algorithms for unit-square and unit-disk labels as well as a fast greedy heuristic for arbitrarily shaped labels. In experiments on real-world data, we compared the results of the non-exact algorithms with optimal solutions, which we obtained by integer linear programming.

7.1 Introduction

Map labeling is a standard technique for visualizing spatial data. It refers to annotating a map with text and icons. Recently research on map labeling has dealt with consistency and stability constraints for interactive

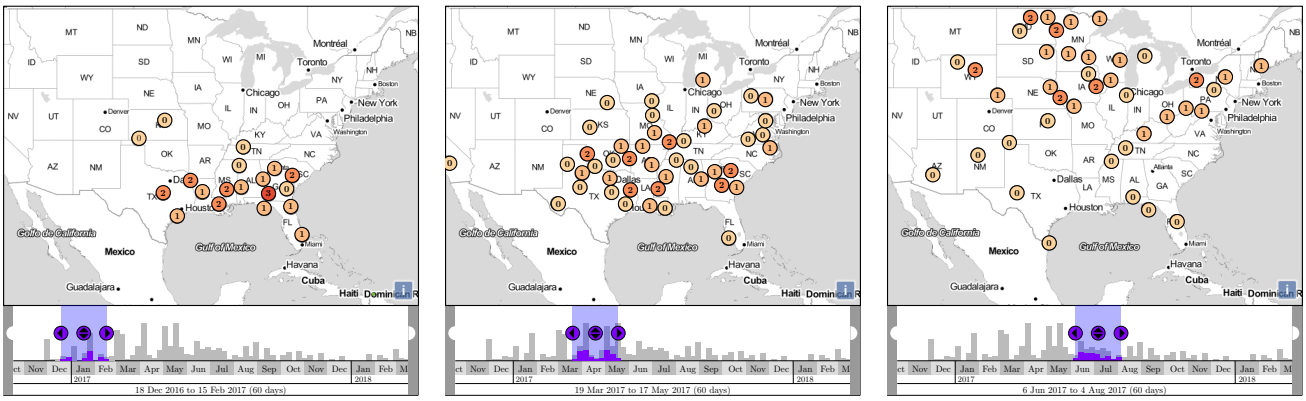
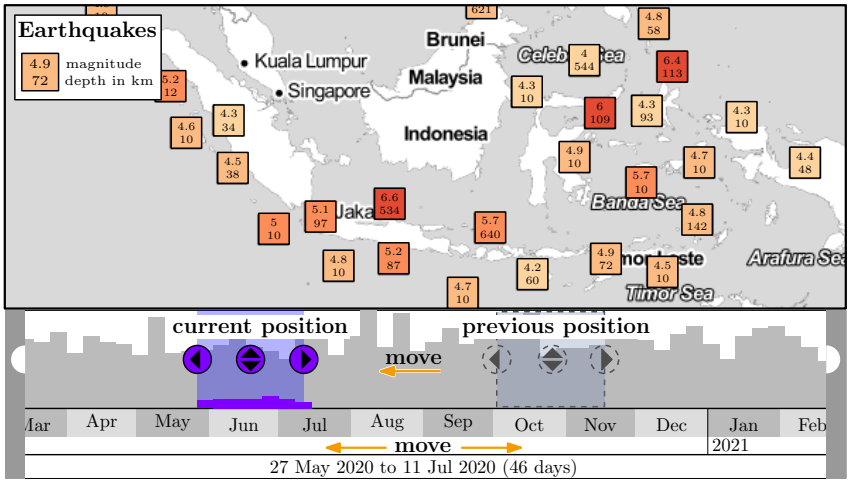
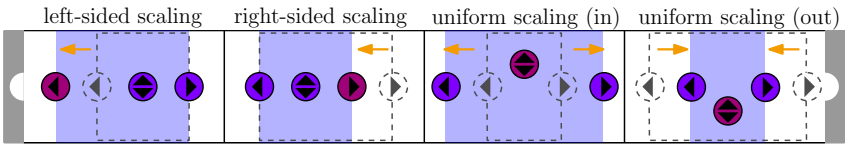


Figure 7.1: Tornadoes in the year 2017 in the United States. The tornadoes particularly exist (left) in the southeast in winter, (middle) in the midwest in spring, and (right) in the north of the United States in summer. The colors and numbers indicate the strengths of the tornadoes. *Map tiles by Stamen Design, under CC BY 3.0. Map data by OpenStreetMap, under ODbL.*



(a) panning



(b) left-sided, right-sided, and uniform scaling

Figure 7.2: Earthquakes in 2020 in Southeast Asia and basic interactions with the time-slider. *Map tiles by Stamen Design, under CC BY 3.0. Map data by OpenStreetMap, under ODbL.*

maps [Zhang et al., 2020, Bahrdt et al., 2017, Peng et al., 2020, Meijers et al., 2020]. In this article, we consider interactive maps for the exploration of spatiotemporal events.

As examples for spatiotemporal events, take natural phenomena (e.g., earthquakes, storm events or the migration of animals) or cultural events (e.g., concerts) where every event corresponds to a location and a timestamp. A typical task that a user wants to solve with the data is to search for an event that lies in a time window. For example, take a singer who has several concerts at different locations and times. Then a typical task of a music fan is to find a concert in a certain month that has a good location. We want to point out that there is data similar to spatiotemporal events with similar user tasks. Take a tourist who searches for a hotel in a foreign city. Every hotel corresponds to a location and a price-per-night (instead of a timestamp) and the tourists search for hotels that lie in their desired price range. Note that we consider spatiotemporal events for a simple notation in the remainder of this chapter but all our models and approaches can be used for similar data sets.

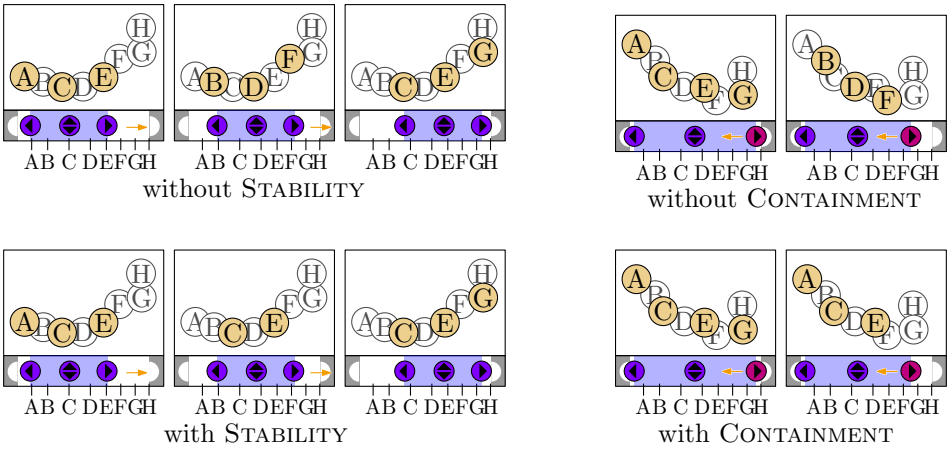
A common visualization approach for the described task is an interface that consists of a map displaying the events' locations and possibly additional information as well as a filter tool to search for events in time windows. In Figure 7.1, we show our exemplary interface that allows users to specify a time window and receive the visualization.

The events are displayed with annotations on the map that are either simple (e.g., in the use-case of tornadoes, we have used circular annotations showing the tornadoes' strengths) or more complex (e.g., diagrams, icons, or plots showing additional data) and that are placed at the locations of the events. Figure 7.1 shows three maps of the United States annotated with circular annotations displaying the occurrences of tornadoes in winter, spring, and summer. To have a clear visualization and avoid an occluded appearance of the map, only a selection of events is displayed. We call a selection of annotations where no two annotations overlap and whose timestamps lie in a time window a *labeling* of that time window.

Commonly, the filtering for a time window is implemented by time-sliders [Andrienko and Andrienko, 1999]. In our interface, illustrated in Figure 7.1, we introduce a *timeline* that consists of a time axis as well as a a time-slider (purple rectangle) that represents the time window. We allow user interaction with a time-slider that enables the following basic interactions:

(1.) *panning:* continuous translation of the time window (see Figure 7.2a),

(2.+3.) *left-sided and right-sided scaling:* continuous change of the left or right boundary of the time window, respectively (see Figure 7.2b),



(a) panning (b) right-sided scaling

Figure 7.3: Sequences of map frames for two basic interactions. The colored labels are displayed, while the white labels and the timestamps are shown only for illustration.

(4.) *Uniform Scaling:* continuous change of both boundaries of the time window in opposite directions, such that the center of the time window remains the same (see Figure 7.2b).

When the user performs a basic interaction, a sequence of map frames is generated and displayed where each map frame shows the labeling of one time window. These map frames form an animation of the map showing the occurrences of the events over time. In the following, we discuss our approach for selecting the labels for each time window.

At first, we aim for reproducibility, which means that the labeling of a time window should be the same no matter what kind of basic interaction has happened before.

Then, we would like to have a good selection of labels for every time window. In static map labeling a typical strategy is to show a maximally large selection of overlap-free annotations to obtain a high information density while preserving the clearness of the visualization, e.g., see Yoeli [1972]. For our scenario, we assume that the events are of different importance weight. We want to maximize the sum of the weights of the displayed labels integrated over all time windows while for every time window no two displayed labels overlap (similar to Been et al. [2006]). This objective allows us, e.g., in the hotel-use case to display as many potential options to the user as possible.

However, by simply summing the weights of the displayed labels integrated over all time-window queries, the user may experience undesirable flickering effects from frame to frame. That means a single annotation may appear and disappear repeatedly. Unless additional requirements on the labelings are enforced, this can happen even within a single basic interaction. For example, in Figure 7.3a (upper row), the labels A, C, and E appear and disappear multiple times although their timestamps lie in the time window for the entire sequence. Such flickering distracts the user and, therefore, we require that the sequence of presented labelings is stable. In detail, we require that during one basic interaction an annotation may appear and disappear only once; Figure 7.3a (lower row).

Also, we want to ensure that the users are able to isolate a single event by systematically shrinking the time window. For example, in Figure 7.3b (upper row) the labels A, C, and E disappear repeatedly although they are still contained in the time window. The users may think that A, C, and E are not contained in the time window after shrinking. Figure 7.3b (lower row) shows a solution that allows the isolation of events. To summarize, we require the following properties.

- REPRODUCIBILITY. The labeling of a time window Q is always the same. That means it is independent of the basic interactions that happened before.
- MAXINFO. Integrated over all possible time-window queries the sum of the weights of the displayed labels is maximized.
- STABILITY. Changing the time window by one basic interaction, a label appears and disappears at most once.
- CONTAINMENT. If a label of an event is displayed for a time window Q then it is also displayed for all the time windows that are contained in Q and contain the timestamp of the event.

Due to REPRODUCIBILITY, our approach is to pre-compute a data structure that encodes for every time window which labels are displayed. With the properties STABILITY and CONTAINMENT, we enforce the consistency of

time-window labelings: STABILITY avoids flickering effects, while CONTAINMENT allows the user to isolate events. As we show later, CONTAINMENT subsumes STABILITY in our formal model. Without the properties STABILITY and CONTAINMENT, the property MAXINFO can be implemented by optimizing each query independently. Hence, requiring STABILITY and CONTAINMENT substantially changes the problem.

Our Contribution In this chapter, we present and discuss

1. a **new model** for consistent map labeling during time-slider interactions that tackles the properties REPRODUCIBILITY, STABILITY, CONTAINMENT, and MAXINFO.
2. **algorithms** for computing solutions that comply with our model and can be loaded into standard data structures that enable efficient retrieval of maps during interaction. We show that queries for maps essentially correspond to rectangle-stabbing queries, and hence, we can propose to use STR-packed R-trees [Leutenegger et al., 1997].
3. an **experimental evaluation** based on a comparison of our methods with a baseline method that does not incorporate any consistency criteria.

We invite the reader to try out our prototypical implementation at `https://www.geoinfo.uni-bonn.de/twl`. We want to emphasize that although this implementation of the visualization exists, our contribution is not a visualization system but the model and algorithms for consistent map labeling.

7.2 Related Work

For a review of related work for map labeling techniques that are related, we refer to Chapter 2.

We want to add that, from a more technical point of view, our work is related to data structures that aim at improving query times for the interactive visualization of spatiotemporal data, e.g., the standard Data Cubes [Gray et al., 1997], or more recent variants such as NanoCubes [Lins et al., 2013] or TimeLattice [Miranda et al., 2018]. Note that these data structures do not take consistency criteria for the visualization into account. The focus is solely on the improvement of query time with less additional memory consumption.

7.3 λ-Structure

In the following, first, we introduce our model in a formal way. Secondly, we discuss the structural results that come with our model.

7.3.1 Problem Definition

We assume that we are given n spatiotemporal *events* $e_1, \ldots, e_n$. Each event e_i is represented by a point p_i in the plane, a timestamp t_i when the event occurred, and a weight $w_i \in \mathbb{R}^+$ reflecting the importance of the event. We assume that the events are ordered such that $t_1 \leq \ldots \leq t_n$. Let $E = \{e_1, \ldots, e_n\}$ be the set of all events. For each event we are given a *label* ℓ_i which is a geometric object in the plane, e.g., an axis-aligned square or disk centered at p_i; see Figure 7.1 and Figure 7.2. We say that two displayed labels (and correspondingly their events) are in *conflict* if the labels overlap in the plane.

Due to the application scenario, where the user changes the time-window query by moving sliders, we are given a minimal slider position $t_{\min}$ and a maximal slider position $t_{\max}$ such that $t_i \in [t_{\min}, t_{\max}]$ for each $1 \leq i \leq n$. We call a time interval $Q = [t', t''] \subseteq [t_{\min}, t_{\max}]$ a *time-window query*.

For a time-window query Q, let E_Q^* be the subset of E that contains each event $e_i \in E$ for which t_i lies in Q, i.e., $t_i \in Q$. Let L_Q^* denote the set of labels of events in E_Q^*. Note that, displaying all labels in L_Q* might lead to label overlaps. Let $L_Q \subseteq L_Q^*$ such that no two labels of L_Q overlap. We call L_Q a *time-window labeling* or more shortly a *labeling*. Figure 7.4 illustrates a time-window query and a labeling. Further, we call a label $\ell \in L_Q$ an *active label* of Q, i.e., it is displayed for Q. We denote the set of events that corresponds to labels in L_Q by E_Q.

Assuming we are considering only a single time-window query, we get the standard static labeling problem. To implement MAXINFO in this static case, one would look for a labeling L_Q that maximizes either the number of active labels or the sum of their weights $\sum_{e_i \in E_Q} w_i$. However, our use-case is not limited to one time-window query but the user may interact with the time-slider. We want to emphasize that an optimization of the labeling for each time-window query independently might lead to a violation with REPRODUCABILITY, STABILITY, and CONTAINMENT. In the following, we introduce our data structure and show how to optimize MAXINFO while requiring REPRODUCIBILITY, STABILITY, and CONTAINMENT for the time-slider interaction.

To realize REPRODUCIBILITY, we pre-compute the labelings of all time-window queries in advance and store them in a specially defined data structure. As described before, we introduce an activity region τ_i for every event

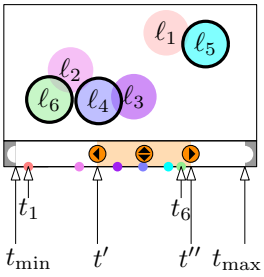


Figure 7.4: Map and time-slider for events $e_1,\ldots,e_6$, time-window query $Q=[t',t'']$ and labeling for $[t',t'']$ consisting of ℓ_4,ℓ_5,ℓ_6 (illustrated with black stroke).

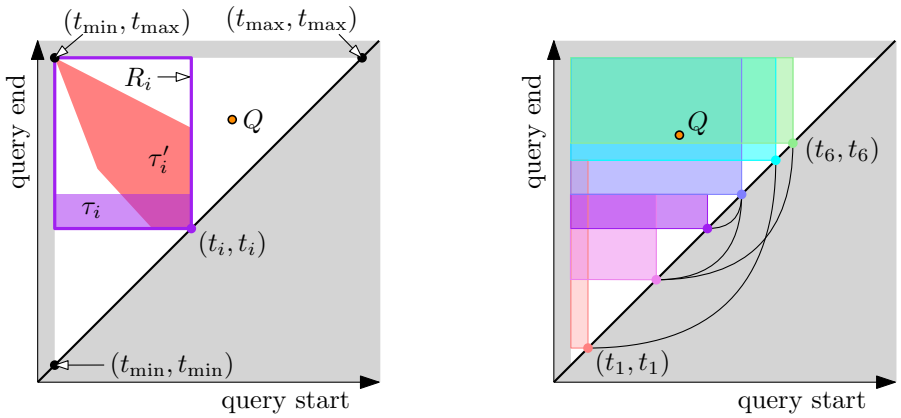


Figure 7.5: Time-window query Q and query region (white triangle). (a) Activity range R_i, activity regions τ_i and τ_i', τ_i' is never part of an optimal activity diagram. (b) Valid activity diagram of events presented in Figure 7.5; pairs of conflicting events are connected with a curve

e_i and store the pair in the λ-structure. As shown in Figure 7.5, each time-window query $Q=[t',t'']$ corresponds to a point (t',t'') in the plane. For brevity depending on the context we interpret Q either as interval $[t,t'']$ or point (t',t'') in the plane. Each point representing a time-window query lies in the triangle spanned by $(t_{\min},t_{\min})$, $(t_{\min},t_{\max})$, and $(t_{\max},t_{\max})$, which we call the *query region*; see Figure 7.5a. We call the line through $(t_{\min},t_{\min})$ and $(t_{\max},t_{\max})$ the *main diagonal* of the activity diagram. An event e_i corresponds to a point (t_i,t_i) on the main diagonal of the activity diagram. We observe that the label ℓ_i can only be active for queries that lie in the rectangle R_i that is spanned by (t_i,t_i) and $(t_{\min},t_{\max})$; we call R_i the *range* of e_i. For the proposed data structure, we pre-compute for each event e_i a region τ_i in R_i; see the colored regions in Figure 7.5b. We call τ_i the *activity region* of e_i. The activity region τ_i exactly contains the queries for which the label ℓ_i can be active. To enforce the properties STABILITY and CONTAINMENT, we can only allow activity regions of certain shapes. At first, we formalize the property CONTAINMENT. An activity region τ_i is *monotone* if, for two time-window queries Q and Q' in the range of e_i with $Q'\subseteq Q$, it holds that Q' lies in τ_i if Q lies in τ_i. Figure 7.5a illustrates two monotone activity regions. Clearly, an event with a monotone activity region fulfills the property CONTAINMENT and we show later that it also satisfies property STABILITY. Roughly speaking such an activity region of an event e_i is the union of a set of axis-aligned rectangles whose bottom-right corners are (t_i,t_i), where t_i is the timestamp of e_i; for proof see Lemma 5. Later we argue that we can reduce activity regions to be rectangles as illustrated in Figure 7.5.

Let $T=\{\tau_1,\ldots,\tau_n\}$ be a set of monotone activity regions of the events $E=\{e_1,\ldots,e_n\}$. A query Q on T yields the set $L_Q^T=\{\ell_i \mid Q\in\tau_i \text{ with } 1\leq i\leq n\}$. We call T an *activity diagram* of the events E. Further, it is *valid* if each query Q on T yields a time-window labeling L_Q^T. This is equivalent to requiring that no two activity regions τ_i and τ_j intersect when ℓ_i and ℓ_j are in conflict.

In the following, we formalize the property MAXINFO. Let $E=\{e_1,\ldots,e_n\}$ be a set of spatiotemporal events. We call $v(\tau_i)=w_i\cdot\text{area}(\tau_i)$ the *volume* of τ_i where $\text{area}(\tau_i)$ is the area of τ_i and we call $v(T)=\sum_{i=1}^n v(\tau_i)$ the *total volume* of T. The absolute volume $v(T)$ corresponds to the sum of weights of displayed labels integrated over all possible time-window queries, and hence, maximizing $v(T)$ leads to property MAXINFO.

TIMEWINDOWLABELING summarizes our problem setting to optimize MAXINFO while guaranteeing REPRODUCIBILITY and CONTAINMENT. Later, we show that TIMEWINDOWLABELING also guarantees STABILITY.

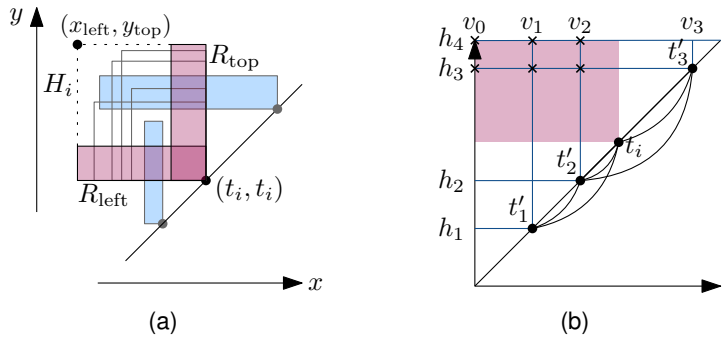


Figure 7.6: (a) Proof of Lemma 5. Illustrated is event e_i with timestamp t_i; range R_{left} with minimal x-coordinate x_{left} and R_{top} with maximal y-coordinate y_{top} of e_i; and rectangle H_i spanned by the lower-right corner (t_i, i_i) and the upper-left corner $(x_{\text{left}}, x_{\text{top}})$. Any activity region (blue) of other events that intersects H_i also intersects either R_{left} or R_{top}. (b) Discretization of solution space. Shooting vertical and horizontal rays from the timestamps induces a grid that is the basis for the candidate activity regions.

TIMEWINDOWLABELING.

Given: *A set $E = \{e_1, \ldots, e_n\}$ of spatiotemporal events with labels; the bounds $t_{\min}$ and $t_{\max}$ of the activity diagram.*

Find: *The λ-structure with a valid activity diagram $T = \{\tau_1, \ldots, \tau_n\}$ of monotone activity regions for E that maximizes $\sum_{i=1}^{n} w_i \cdot \text{area}(\tau_i)$, where $\text{area}(\tau_i)$ is the area of τ_i in the activity diagram.*

We say that an optimal solution for TIMEWINDOWLABELING is an *optimal* λ-structure with an *optimal* activity diagram; see Figure 7.5.

7.3.2 Structural Results

We show that for solving TIMEWINDOWLABELING it suffices to consider rectangular activity regions.

Lemma 5. *The activity regions of an optimal activity diagram T for* TIMEWINDOWLABELING *are axis-aligned rectangles whose bottom right corners lie on the main diagonal of the activity diagram.*

Proof. We first prove that the property CONTAINMENT implies that the activity region τ_i of any event $e_i \in E$ is the union of a set of axis-aligned rectangles whose bottom-right corners are (t_i, t_i), where t_i is the timestamp of e_i. Afterward, we show that optimizing the property MAXINFO implies that τ_i is a single axis-aligned rectangle whose bottom-right corner is (t_i, t_i).

Assume that $\tau_i \in T$ of e_i is a monotone activity region and, hence, satisfies the property CONTAINMENT. Consider an arbitrary time-window query $Q = [t', t'']$ that lies in τ_i. For any query $Q' \subseteq Q$ it holds that it lies in the axis-aligned rectangle R_Q spanned by (t', t'') and (t_i, t_i). By the property CONTAINMENT it follows that R_Q is part of the activity region τ_i. Hence, we obtain that τ_i is the union of a (possibly infinitely large) set A_i of axis-aligned rectangles whose bottom-right corners are (t_i, t_i).

Now assume that T is optimal with respect to the property MAXINFO, and hence it is maximal with respect to the total area of the activity regions. Among all rectangles in A_i let R_{top} be a rectangle whose top side has a maximal y-coordinate and let R_{left} be a rectangle whose left side has minimal x-coordinate; see Figure 7.6a. Let y_{top} and x_{left} be the corresponding y-coordinate and x-coordinate, respectively. We observe that any activity region that intersects the rectangle H_i spanned by $(x_{\text{left}}, y_{\text{top}})$ and (t_i, t_i) also intersects either the rectangle R_{top} or R_{left}. Hence, for any event $e_j \in E$ that is in conflict with e_i its activity region τ_j cannot intersect H_i. Thus, as T maximizes the total volume, which increases with the area of the activity regions, the rectangle H_i is part of τ_i. By the extremal choice of R_{top} and R_{left}, we further obtain that τ_i is exactly H_i. Thus, we obtain the statement of the lemma. □

Due to Lemma 5 we can discretize the solution space such that for each event we can choose its activity region from $O(n^2)$ rectangles. We define for each event $e_i \in E$ a *candidate set* C_i as follows. Let $e'_1, \ldots, e'_k$ be the events that are in conflict with e_i and let $t'_1, \ldots, t'_k$ be the timestamps of these events, respectively. Further, for each event e'_j with $1 \leq j \leq k$, let v_j be the vertical segment that connects (t'_j, t'_j) with $(t'_j, t_{\max})$. Similarly, let h_j be the horizontal segment that connects (t'_j, t'_j) with $(0, t'_j)$; see Figure 7.6b. Further, let v_0 be the vertical line through $(t_{\min}, 0)$ and let h_{k+1} be the horizontal line through $(0, t_{\max})$. Let S be the set of pairwise intersection points between $h_1, \ldots, h_k, h_{k+1}$ and $v_0, v_1, \ldots, v_k$. For e_i the candidate set C_i contains the axis-aligned rectangles that are spanned by (t_i, t_i) and the intersection points of S that lie in the range R_i of e_i.

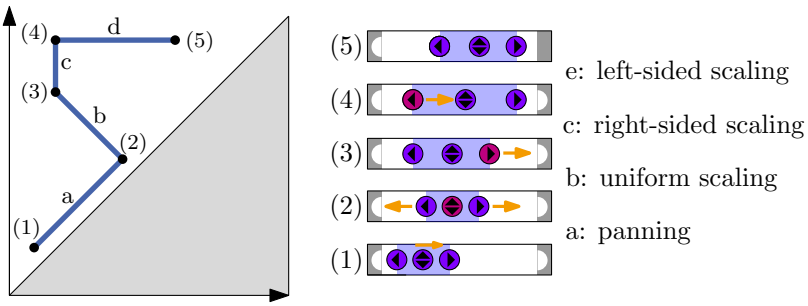


Figure 7.7: Illustration of a query path. (1)–(5): the time windows at the vertices of the query path.

Lemma 6. *Let T be an optimal activity diagram for E. For each event $e_i \in E$ its activity region $\tau_i \in T$ is a rectangle in the candidate set C_i.*

Proof. We show that for each optimal activity diagram of E, the activity region τ_i of an event $e_i \in E$ is contained in C_i. By Lemma 5, τ_i is an axis-aligned rectangle whose bottom right corner is (t_i, t_i). A similar statement holds for any event e_j that is in conflict with e_i. In particular, the horizontal line supporting the lower boundary and the vertical line supporting the right boundary of an activity region in the optimal solution are fixed a priori. Assume for the sake of contradiction that the upper boundary of τ_i lies on a horizontal line that is not defined by any of the timestamps $t_1, \ldots, t_n$. Then, either we can extend τ_i upwards, or there exists an activity region τ_j that blocks τ_i from above. However, since the bottom boundary of this other activity region is fixed at t_j, the latter cannot be the case. Therefore, we could extend τ_i implying that the solution is not optimal. A symmetric argument can be made for the left boundary of τ_i. Therefore, it must be that $\tau_i \in C_i$. □

Coherent Interactions In the following, we argue that the property CONTAINMENT subsumes STABILITY, which implies that a solution of TIMEWINDOWLABELING also satisfies the property STABILITY. As described before, we assume that the time-window query is chosen interactively using sliders. We allow the user to choose the time window $Q = [t', t'']$ by four basic interactions: *panning*, *left-sided scaling*, *right-sided scaling*, and *uniform scaling*. In more detail, when panning the time window by Δ the resulting time-window query is $[t' + \Delta, t'' + \Delta]$. Further, the left-sided scaling changes the left boundary t' of the time window to $t' + \Delta$, and the right-sided scaling changes the right boundary t'' of the time window to $t'' + \Delta$ by an amount Δ. Finally, the uniform scaling changes the time window by an amount Δ in both directions to $[t' - \Delta, t'' + \Delta]$. We note that Δ can also be negative.

Consider the interaction of a user with the time-sliders in an activity diagram. The sequence of time-window queries issued by the user forms a trajectory in the activity diagram; we call it a *query path*. We observe that a query path consists of a sequence of vertical, horizontal, and diagonal segments such that each segment corresponds to a basic interaction. Figure 7.7 gives an exemplary query path and the according basic interactions. Hence, for optimal activity diagrams, it follows from Lemma 5 that the property STABILITY is satisfied: for a basic interaction a label appears and disappears only once, as the intersection of a segment with a rectangle is at most a single segment.

7.4 Complexity and Exact Solution

In this section, we prove that constructing an optimal activity diagram (and therewith, an optimal λ-structure) is NP-hard and present an ILP formulation for TIMEWINDOWLABELING. Once we have constructed the activity diagram, we can efficiently answer time-window queries utilizing rectangle-stabbing queries, i.e., for a time query Q we return all labels whose activity regions contain Q.

7.4.1 Computational Complexity

First of all, we show that TIMEWINDOWLABELING is NP-hard by providing a reduction from a closely related static labeling problem.

Theorem 7. *Let E be a set of events with either unit-disk or axis-aligned unit-square labels. It is NP-hard to find an optimal activity diagram of E, even if each event $e \in E$ has weight 1.*

Proof. Given a set S of unit disks or axis-aligned unit squares, it is NP-hard to find a set $S' \subseteq S$ of maximum cardinality such that no two elements in S' intersect [Fowler et al., 1981]. We call the version of the problem with squares MISS (i.e., every input set of squares). By showing that TIMEWINDOWLABELING contains MISS as a special case, we prove that TIMEWINDOWLABELING is NP-hard as well. More precisely, every instance of MISS can be solved by constructing and solving a corresponding instance of TIMEWINDOWLABELING. We set $t_{\min} = 0$ and $t_{\max} = 2$ and annotate every square in S with the same timestamp $t_i = 1$ for our construction procedure (or *reduction*). This means that an intersection between any two activity regions is allowed if and only if the two corresponding labels intersect. Therefore, if we were given an optimal solution to the instance of TIMEWINDOWLABELING, we could simply return the set of all labels with non-empty activity regions as an optimal solution to the MISS instance. The proof for unit disks works analogously. □

7.4.2 ILP Formulation

Due to Theorem 7, we pursue solutions based on ILP formulations. The general idea of an ILP formulation is to formalize the given optimization problem as a linear objective function subject to linear inequality constraints. As the variables are integers in an ILP formulation, solving it is NP-hard in general [Garey and Johnson, 1979]. However, there are powerful solvers that often can be used to solve such formulations in practice. For the purposed ILP formulation we interpret TIMEWINDOWLABELING as a problem of finding a maximum-weight selection on rectangles that represent possible activity regions. Lemma 6 yields that it is sufficient to consider the rectangles contained in the set C_i of candidates for each event e_i. For each event $e_i \in E$ and each rectangle $r \in C_i$ we introduce a binary variable $x_{i,r}$, which we interpret such that $x_{i,r} = 1$ if and only if the rectangle r is selected as activity region τ_i for e_i. For each event e_i we enforce that at most one activity region can be selected by the constraint

$$\sum_{r \in C_i} x_{i,r} \leq 1.$$

For every pair of distinct events $e_i,\ e_j \in E$ that are in conflict, we need to ensure that their labels are not displayed at the same time. We formalize this by introducing for every pair $(r, r') \in C_i \times C_j$ where r and r' intersect a constraint

$$x_{i,r} + x_{j,r'} \leq 1.$$

At last, our aim is to maximize the total volume of the activity diagram. This corresponds to maximizing the following objective

$$\sum_{e_i \in E} \sum_{r \in C_i} w_i \cdot \operatorname{area}(r) \cdot x_{i,r}.$$

We obtain the optimal activity diagram T_{ILP} for E by setting for each event $e_i \in E$ its activity region τ_i as the rectangle $r \in C_i$ with $x_{i,r} = 1$.

Theorem 8. *The set T_{ILP} is an optimal activity diagram for E.*

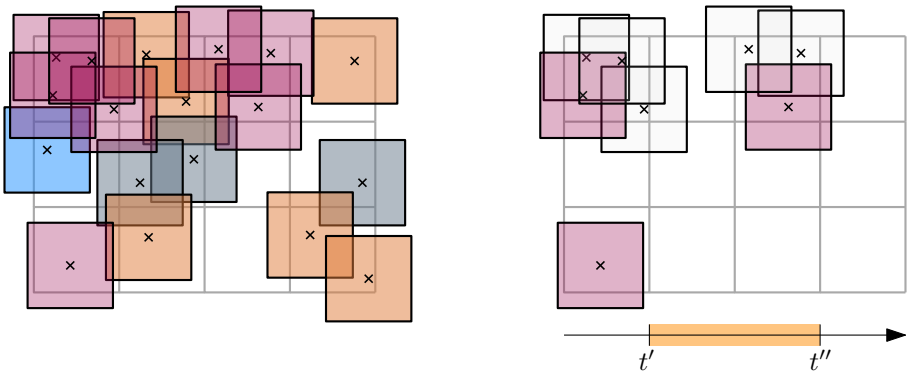
With our ILP formulation, we can replace the volume of the activity diagram with other measures. For example, the square root of the area of the activity regions could lead to very small activity regions being avoided in the optimal solution.

7.5 Non-Exact Solutions

As TIMEWINDOWLABELING is NP-hard, we tackle the problem with faster algorithms that guarantee REPRODUCIBILITY, STABILITY and CONTAINMENT while they might not perform best for MAXINFO. We present approximation algorithms that guarantee a certain approximation factor α, i.e., the ratio between the total volume of the obtained activity diagram and the total volume of an optimal activity diagram is at most the approximation factor. Further, we present a greedy heuristic.

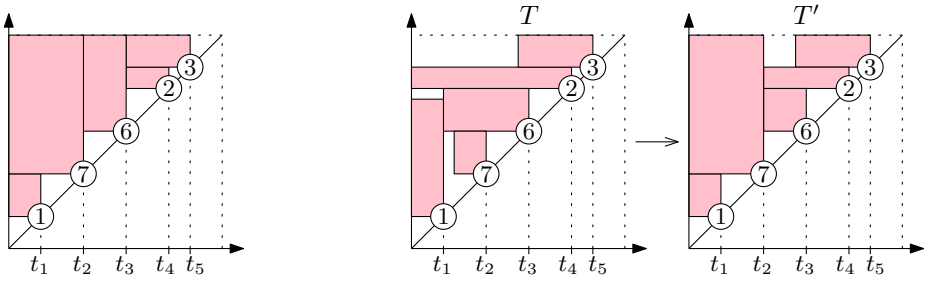
7.5.1 Approximation Based on a Partitioning Scheme

In this section, we discuss approximation algorithms for specific types of labels: (i) the labels are axis-aligned rectangles of equal width and equal height, or (ii) the labels are disks of equal size. For rectangular labels, we prove an approximation factor of 4, and for disk-shaped labels an approximation factor of 7. Note that for any problem instance, i.e., for any input to TIMEWINDOWLABELING, the map can be scaled even with different scale



(a) partitioning of labels (b) labeling for time-window query

Figure 7.8: Approximation algorithm for unit-square labels (see Figure 7.10a). (a) Labels of one color correspond to one subset. (b) Assume that the solution for the red subset is best. Then, the labeling of a time-window query contains only labels from the red subset.



(a) optimal solution (b) exchange argument in proof of Lemma 7

Figure 7.9: Proof of Lemma 7 illustrated for events $e_1,\ldots,e_5$ with timestamps $t_1,\ldots,t_5$, respectively. The weights of the events are given as numbers in the circles. (a) Due to the process, we insert the activity regions in the following order: e_2,e_1,e_3,e_5,e_4. (b) An arbitrary activity diagram T and a solution T' that is based on T but where the activity region of e_2 is maximized. The objective of T' is larger.

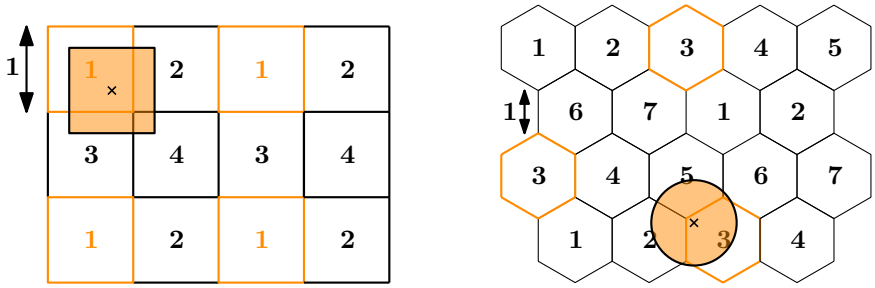
factors in the x- and y-dimension without changing the structural properties such as the intersection relationships between labels. Therefore, we assume, without loss of generality, that our labels are either unit squares or unit disks.

Both approximation algorithms are based on the idea of (1) partitioning the given set of events into subsets that can each be solved efficiently, (2) computing an optimal solution for each subset, and (3) returning the solution of the subset with highest objective value. Figure 7.8 gives an exemplary instance, its partition and a labeling resulting from the approximation algorithm. To make it more clear, in the solution of the approximation algorithms only events from the subset with highest objective value have a non-empty activity region. The activity regions of labels of all other subsets are empty.

For static map labeling, *line stabbing* is a common technique for partitioning a given set of labels into subsets whose label-label conflict graph is an interval graph [Agarwal et al., 1998]. In a similar fashion, we apply a partitioning scheme yielding subsets whose label-label conflict graph is a clique, i.e., every label is in conflict with every other label of the clique. We can solve such instances efficiently. We first present an exact algorithm for cliques and then show how it is used in our approximation algorithms.

Lemma 7. TIMEWINDOWLABELING *can be solved in $O(n)$ time if the label-label conflict graph is a clique and the events are given in the order of their timestamps.*

Proof. Let e_i be an event of maximum weight. We maximize the activity region for e_i by placing its upper-left corner in the upper-left corner $(t_{\min},t_{\max})$ of the activity diagram. We apply the same procedure first to the activity diagram spanned by the temporal interval $[t_{\min},t_i]$ and upper left corner $(t_{\min},t_i)$ and secondly to the activity diagram spanned by the temporal interval $[t_i,t_{\max}]$ and upper left corner $(t_i,t_{\max})$; see also Figure 7.9a. Note that these are two independent parts of the diagram.



(a) square grid (b) hexagonal grid

Figure 7.10: Grids used to partition a set of (a) unit-square labels and (b) unit-disk labels into four and seven subsets, respectively.

Selecting the activity region for e_i as large as possible is correct, as can be seen with a simple exchange argument (which can be applied recursively); see Figure 7.9b. Let T be an arbitrary solution. Suppose we construct a new solution T' by adding the largest possible activity region for e_i to T and clipping all activity regions in T overlapping it. Since the label-label conflict graph is a clique, the clipped parts of the activity regions are pairwise disjoint and, thus, any clipped part is now covered by a unique section of a newly gained part of the region of e_i. Since e_i has maximum weight, the total volume can only increase. Since the same argument applies if T is an optimal solution, maximizing the activity region of e_i is correct.

The algorithm can be implemented as a pre-order traversal through a binary tree containing the events and satisfying (i) the search-tree property for the timestamps and (ii) the max-heap property for the volumes. Such a tree is called a Cartesian tree and can be built in $O(n)$ time if the elements are given in order [Gabow et al., 1984]. A pre-order traversal through the tree requires $O(n)$ time, leading to $O(n)$ time in total. □

The partitioning of events and their labels into cliques is based on a square grid in the case of unit squares and a hexagonal grid in the case of unit disks, where every edge has length one. We illustrate both grids in Figure 7.10. Every label ℓ is assigned to the grid cell containing the center point of ℓ. The lexicographical order ($<$) of the cell's center points is used to break ties, meaning that a label whose center lies on the boundary between two cells c_1 and c_2 with $c_1 < c_2$ is assigned to c_1. With this, the grid with the assigned labels has the following properties:

(i) For every cell, the conflict graph of the assigned labels is a clique, i.e., all assigned labels are pairwisely in conflict.

(ii) For the square grid, there exists an assignment that maps each grid cell to one of the numbers $1,2,3,4$, such that no two labels assigned to different cells with the same number intersect (see Figure 7.10a).

(iii) The latter holds for the hexagonal grid and numbers $1,\dots,7$, as shown by Chan et al. [2008a] in another context (see Figure 7.10b).

Because of (i), the events that are assigned to the same grid cell can be solved optimally with the algorithm described in the proof of Lemma 7. Because of (i) and (ii) or (iii), respectively, we can combine the resulting labelings for grid cells with the same number into a single labeling (in which no two labels overlap). Let $T_1,\dots,T_k$ be the k solutions that we obtain, where $k=4$ for square labels and $k=7$ for disk labels. Among these solutions, we return a solution of maximum total volume, leading to the following result.

Theorem 9. *The algorithm based on a partitioning scheme approximates* TIMEWINDOWLABELING *with factor* 4 *for unit squares and with factor* 7 *for unit disks. If the events are given in the order of their timestamps, the algorithm can be implemented to run in* $O(n+d)$ *time, where* d *is the number of grid cells.*

Proof. Let $T^\star$ be an optimal solution. By applying the partitioning scheme of our algorithm to the labels, we partition $T^\star$ into k solutions, which we denote with $T_1^\star,\dots,T_k^\star$. For $i=1,\dots,k$, it holds that $v(T_i) \geq v(T_i^\star)$, since (i) both $T_i^\star$ and T_i are solutions for the same set of labels and (ii) T_i is optimal. Therefore, $\sum_{i=1}^{k} v(T_i) \geq v(T^\star)$ and, thus, at least one of the solutions $T_i,\dots,T_k$ has total volume greater or equal $v(T^\star)/k$.

Constructing the grid and assigning every label to its cell requires $O(n+d)$ time. For each cell, the corresponding instance of n' labels can be solved in $O(n')$ time with the algorithm for cliques. Since the instances are disjoint, solving all instances amounts to $O(n)$ time. □

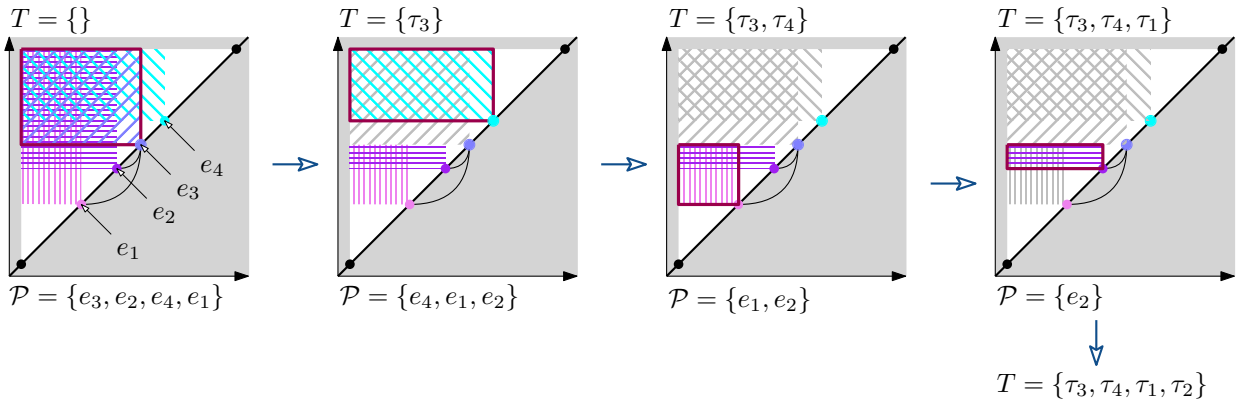


Figure 7.11: Greedy algorithm for four events e_1, e_2, e_3, e_4 and equal weights. There is a label conflict for the pairs (e_1, e_3) and (e_2, e_3).

7.5.2 Greedy Heuristic

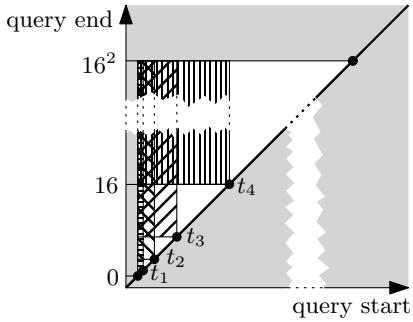
In this section, we present a greedy heuristic for computing a valid activity diagram which is not based on a partitioning scheme. For illustration see Figure 7.11. At first, we present the algorithm, secondly we discuss the running time, thirdly we compare the result the greedy heuristic and the approximation algorithm for an exemplary set of events and at last, we show that the greedy heuristic can not guarantee any approximation ratio.

The idea of the greedy heuristic is to successively select activity regions that yield the largest gain. While doing so, it maintains for each event that has not yet been placed in the activity diagram its maximal potential activity region. Each time a new event is selected and placed in the diagram, all remaining activity regions that are in conflict with this event are trimmed and their potential contribution is updated accordingly. More in detail, we initialize for each event $e_i \in E$ its largest possible activity region τ_i, i.e., the region that is spanned by $(t_{\min}, t_{\max})$ and (t_i, t_i) and further, its volume as $v(\tau_i) = w_i \cdot \text{area}(\tau_i)$. We initialize a priority queue $\mathscr{P}$ of events increasingly ordered by their volumes and the empty solution set T; see step 1 of Figure 7.11. Then, we remove the first event e_i from $\mathscr{P}$ (with largest volume) and add τ_i to the solution set T. For each event e_j that is in $\mathscr{P}$ and that is in conflict with e_i we trim τ_j to the largest possible activity region $\tau_j' \subseteq \tau_j$ that does not intersect τ_i. Finally, we update the volume of e_j in $\mathscr{P}$ to $w_j \cdot \text{area}(\tau_j')$, possibly changing the position of e_j in the order of $\mathscr{P}$. We repeat this process of removing the first event in $\mathscr{P}$ until $\mathscr{P}$ is empty. Then we return the valid activity diagram T.

As for the running time, note that each time an event e_i is removed from $\mathscr{P}$ we trim the activity regions and update the volumes of $O(n)$ events that are in conflict with e_j. Trimming takes $O(1)$ time and updating $\mathscr{P}$ takes $O(\log n)$ time per conflicting event if we implement $\mathscr{P}$ as a binary heap. Hence, we need $O(n^2 \log n)$ time in total.

The greedy heuristic works for arbitrarily-shaped labels, as long as it can check the existence of conflicts between labels efficiently. However, it cannot guarantee an approximation factor as good as that of the partitioning-based algorithm from Section 7.5.1. Consider the following input with 15 events, each with weight 1 and a square-shaped label of size 6×6. The center points of the labels are: (0,0), (6,0), (0,6), (6,6), (4,4), (3,3), (9,3), (3,9), (9,9), (7,7), (6,6), (12,6), (6,12), (12,12), and (10,10). The timestamps are: 8, 8, 8, 8, 8.002, 16, 16, 16, 16, 16.001, 21, 21, 21, 21, and 20.999, respectively, and $[t_{\min}, t_{\max}] = [0, 24]$. For this input, the greedy heuristic achieves a total activity region size of less than 207.107, whereas the optimal solution has a total activity region size of at least $900.025 > 4.34 \cdot 207.107$. Thus, the approximation factor of the greedy heuristic is at least 4.34, whereas the partitioning scheme guarantees an approximation factor 4.

In the following, we show that the ratio between the objective value obtained by the greedy heuristic and the optimal objective value is not bounded by a constant value α. We use a set of n events of different weights to construct input instances that show $\alpha \geq \frac{n}{2}$. Recall that in contrast to the greedy heuristic the approximation algorithm presented in Section 7.5.1 guarantees a constant approximation ratio. Let E be a set of events. Let $E_i \subseteq E$ be the set of events that are in conflict with $e_i \in E$. Then, let $b \in \mathbb{R}$ such that for any two events $e_i, e_j \in E$, we have $1/b \leq w_i/w_j \leq b$. We call b the *degree of unbalance* of E. Choose an interval $[1, b]$ from which to pick the weights, such that b is an integral power of two, larger than 1. Let n be $\log_2 b$. Let $t_{\min} = 0$ and $t_{\min} = b^2$. We create n events $e_1, ..., e_n$ in the time window $[0, b^2]$, where e_n has weight $\frac{1}{b-1}$; the events e_j, for $j \in \{1, ..., n-1\}$, have weight 2^{-j}; each event e_j, for $j \in \{1, ..., n\}$, has timestamp 2^j, and all labels have the same location; see Figure 7.12 for $b = 16$. Note that e_n has maximum volume $\frac{1}{b-1} 2^n (b^2 - 2^n) = \frac{1}{b-1} b(b^2 - b) = b^2$, whereas each other event e_j has maximum volume $2^{-j} 2^j (b^2 - 2^j) = b^2 - 2^j$. The optimal solution would contain at least the right half of each event's maximum possible region (and for e_1, also the left half); the total volume will be roughly $\frac{1}{2} n b^2$.



e	t	$w(e)$
e_1	2^1	$1/2$
e_2	2^2	$1/4$
e_3	2^3	$1/8$
e_4	2^4	$1/(16-1)$

Figure 7.12: Instance with arbitrarily bad approximation ratio for $b = 16$.

More precisely, the total volume of this solution would indeed be:

$$\begin{aligned}&\frac{b^2}{2}+\sum_{j=2}^{n-1}\frac{b^2-2^j}{2}+\left(b^2-2\right)\\&\quad=\frac{(n+1)b^2}{2}-2^{n-1}=\frac{(n+1)b^2-b}{2}\\&\quad=\frac{nb^2+b(b-1)}{2}>\frac{nb^2+nb}{2}.\end{aligned}$$

In the last step, we used $b-1 \geq \log_2 b = n$. The greedy heuristic, however, would first give e_n its maximum possible region. This reduces the maximum height of the activity region of each other event e_j from b^2-2^j to $2^n-2^j = b-2^j$; thus its maximum volume is reduced to $2^{-j}(b-2^j) = 2^{-j}b-1 = 2^{n-j}-1 < 2^{n-j}$, and the maximum total volume of all events is reduced to less than $b^2+\sum_{j=1}^{n-1} 2^{n-j} < b^2+2^n = b^2+b$. Thus, the greedy heuristic's solution is worse than the optimal solution by a factor of at least:

$$\alpha = \frac{(nb^2+nb)/2}{b^2+b} = \frac{n}{2}.$$

Note that the factor $n/2$ is reached under the condition $n = \log_2 b$, or conversely, $b = 2^n$. In other words, the construction requires events whose weight differences are exponential in n.

7.5.3 Combining Partitioning Scheme and Greedy Heuristic

The approximation algorithms based on the partitioning scheme yield labelings in which the labels cover the underlying grid in an undesirable systematic pattern. We, therefore, enhance the activity diagram T_0 from these algorithms using the greedy heuristic. To that end, we copy the initial solution of the approximation T_0 to T. For every event e that has no activity region in T_0, we add the largest possible activity region τ to the priority queue $\mathscr{P}$ as before and use its volume as the sorting key for $\mathscr{P}$. More precisely, before we add τ to $\mathscr{P}$ we trim τ such that it does not intersect any activity region τ' in T_0 that corresponds to a label ℓ' that overlaps the label of e. The remaining part of the greedy heuristic remains unchanged. Hence, it successively adds activity regions in $\mathscr{P}$ to T until the solution is maximal. In particular, trimming activity regions ensures that the result is a valid activity diagram. We note that filling up T_0 does not negatively affect the approximation quality stated in Theorem 9. Other greedy heuristics for improving the quality of the solution are also possible.

7.6 Evaluation and Experiments

In this section, we evaluate the presented model and algorithms on real-world data. We first consider the construction phase of our approach, in which the activity diagram is created. Afterward, we consider the query phase. To use real-world query paths for the evaluation, we conducted a study with users.

7.6.1 Experimental Setup

We considered two different data sets. The first data set $\mathbf{E}_{\text{tornado}}$ contains 5 900 spatiotemporal events of tornadoes in the years 2015–2019. It is obtained from the National Oceanic and Atmospheric Administration of the United States[1]. The second data set $\mathbf{E}_{\text{bird}}$ contains 10 000 observations of two different species of gulls

[1] Available under public domain at `https://www.spc.noaa.gov/wcm/`.

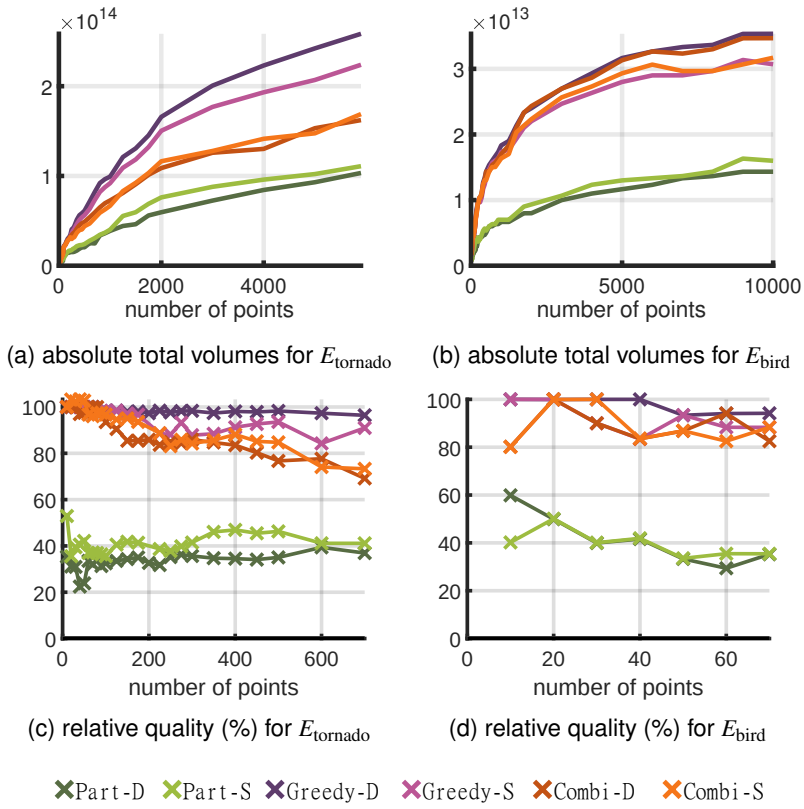


Figure 7.13: Results for the quality of the data structures obtained by the approximation algorithm (Part-D, Part-S), by the greedy heuristic (Greedy-D, Greedy-S), and by the combination of the approximation and greedy (Combi-D, Combi-S). (a),(b) Total volumes for E_{tornado} and E_{bird}, respectively. (c),(d) The relative quality for E_{tornado} and E_{bird}, respectively; see Section 7.6.2.

in Western Europe and at the west coast of Africa in the year 2015 [Stienen et al., 2017]. For the data set of tornadoes, we rated each event e by its storm strength $z \in \{0,1,2,3,4\}$. More precisely, we set $w(e) = 2^z$, thus favoring strong over weak tornadoes. For the occurrences of gulls, each event was rated with the same weight. We note that the data sets have different spatial structures. While the observations of gulls form large clusters, the occurrences of tornadoes appear more evenly distributed.

We evaluate the algorithms discussed in Section 7.5.1–7.5.3 by comparing their solutions to an exact solution of TIMEWINDOWLABELING obtained by the ILP formulation presented in Section 7.4. We consider both unit disks (D) and unit squares (S) as labels, obtaining eight variants for the computation of the activity diagram: the exact solutions ILP-D and ILP-S, the approximations Part-D and Part-S by the partitioning scheme, the solutions Greedy-D, and Greedy-S of the greedy heuristic, and the combined approaches Combi-D and Combi-S.

For the evaluation of the construction phase (Section 7.6.2) we sampled subsets of different sizes where a subset of size k contains the first k events. For the evaluation of the query phase (Section 7.6.3 and Section 7.6.4) we have reduced both data sets to 2000 events E_{tornado} and E_{bird} each by randomly sampling them using a uniform distribution. This maintains the overall structure of the data but enables us to evaluate our non-exact algorithms using a slow ILP-based approach. In Section 7.6 we show that our non-exact algorithms actually run on much larger sets.

The implementations were done in Java, and the ILP formulations were solved by Gurobi 9.1. We ran the experiments on an Intel(R) Xeon(R) W-2125 CPU clocked at 4.00GHz with 128 GiB RAM.

7.6.2 Construction Algorithm Evaluation

In this section we compare the quality and the running time of our algorithms for the previously presented subsets of the data sets.

Table 7.1: Construction times for the occurrences of tornadoes (700 and 5900 events) and the observation of gulls (70 and 10000 events).

algorithm	**tornadoes**		**gulls**	
	700	5900	70	10000
ILP-D	58.43 min	–	6.68 h	–
Part-D	14.98 ms	24.34 ms	19.41 ms	153.02 ms
Greedy-D	23.62 ms	1.67 sec	31.82 ms	15.02 sec
Combi-D	27.32 ms	1.69 sec	4.99 ms	14.71 sec
ILP-S	23.14 h	–	6.31 h	–
Part-S	1.01 ms	4.68 ms	1.15 ms	9.05 ms
Greedy-S	23.76 ms	1.68 sec	2.61 ms	14.95 sec
Combi-S	23.85 ms	1.66 sec	2.62 ms	13.25 sec

Quality of the Activity Diagrams For a non-exact algorithm, we assess the constructed activity diagram T by comparing it to an optimal solution, i.e., we consider its *relative quality*, which is defined as

$$\frac{\text{total volume of activity diagram } T}{\text{total volume of optimal activity diagram}} \cdot 100\%$$

For E_{tornado} the experiments show that the relative qualities of Greedy-D and Greedy-S are at least 84.27%, Part-D and Part-S are at least 22.45%, and Combi-D and Combi-S are at least 69.16%; see Figure 7.13a. We have obtained similar results for E_{bird}; see Figure 7.13b. As the relative quality could be computed only for small data sets, we also consider absolute total volumes; see Figure 7.13c and Figure 7.13d.

Hence, although the greedy heuristic does not give any guarantees, it gives solutions of high quality in practice. Both Combi-D and Combi-S have a head-to-head race with Greedy-D and Greedy-S, respectively, without a clear winner. Although the partitioning scheme has a theoretical approximation factor, the greedy heuristic provides substantially better results for both real-world data sets and clearly prevails over the partitioning scheme. Altogether, we suggest running both the greedy heuristic as well as the combined approach to take the better solution of both.

Running Time From the sampled data we consider the largest data set for which the ILP formulation could be solved and the largest data set that has been sampled; see Table 7.1. The running times of the non-exact algorithms lie in the range of milliseconds and seconds. In particular, the approximation algorithms are clearly faster than the greedy heuristics. We deem the running times to be sufficiently small, as the activity diagrams are only created once in advance. In contrast, as solving the ILP formulations may take hours, even for small data sets, they are less applicable in practice.

7.6.3 User-generated Query Paths

To acquire query paths for the evaluation of our algorithms and model under realistic conditions, we conducted an online study[2]; see Section 7.6.4. It gives us the possibility of analyzing the usage of the basic interactions and of assessing the consistency of our model. Note that we do not aim for a user evaluation of our model with this study.

The main visualization component of our interface is an interactive map that supports time-window filtering; see Figure 7.1 for the map for the tornado events. For the bird observation events, the labels' color encoded (orange and blue) the two different species of gulls. The study is split into two phases. In the first phase, the participants were asked to do a tutorial. During this tutorial the participants solved exercises on a data set of earthquakes to get familiar with the interaction of the map; see Figure 7.2a. In the second phase, the actual experiments were performed. The participants were asked to solve eight tasks: four tasks on each data set.

1. Find a time window that roughly spans one year, starts at some day in December 2019 and ends at some day in December 2020.
2. Find a time window that spans between 30 and 40 days and shows at least 10 tornadoes in 2016.
3. Find a time window such that the majority of tornadoes (at least 20) occurred to the right of the blue line, while only few tornadoes occurred to the left of the blue line.

[2]Available at `https://www.geoinfo.uni-bonn.de/twl/study/`.

Table 7.2: Statistic for the 41 participants of the second run of the study.

Age	16-25:	22	26-35:	11
	36-45:	4	46-55:	2
	56-65:	1	66-99:	1
Gender	female:	13	male:	26
	other:	0	no answer:	2
Personal use of maps	daily:	5	weekly:	12
	monthly:	9	less:	15
Professional use of maps	yes:	10	no:	31

Table 7.3: Number of participants answering the questions on personal opinion (1 = worst score and 5 = best score).

	Scoring				
Question	1	2	3	4	5
Is the use of the time window intuitive?	1	2	3	14	21
Is the use of the timeline intuitive?	0	2	4	16	19
Is the visual connection between timeline and geographic map intuitive?	1	1	2	10	27
How useful are such interactive maps to explore data?	1	0	2	11	27
Would you like to use such interactive map in practice?	1	0	3	10	27

4. In which time period did the highlighted tornado (blue circle) occur? The event occurred in the [first half/ second half] of [January/ February/ ... /December] in [2015/ 2016/ .../ 2019].
5. Find a time window that spans between 20 and 25 days and shows at least 20 occurrences of gulls.
6. Checkmark the correct answers.
   - ☐ Some blue gulls stay in Western Europe for the whole year.
   - ☐ Some blue gulls migrate to Africa in winter.
   - ☐ Some orange gulls stay in Western Europe for the whole year.
   - ☐ Some orange gulls migrate to Africa in winter.
7. Find a time window for which all occurrences of gulls (at least 6) lie above the blue line.
8. Find a time window for which most occurrences of orange gulls lie below the blue line.

The study and hence, all tasks can be found under: `https://www.geoinfo.uni-bonn.de/twl/study/`. As the study was conducted online, we paid special attention to the correct execution by the participants.

- The screen resolution was automatically checked.
- Full screen mode was enforced.
- The participants were asked to compare the interactive map with a screenshot confirming that it is displayed correctly.
- In the tutorial we ensured that the participants used each control at least once, thereby checking its functionality automatically.

User Feedback: Although the only purpose of the study was to get real-world query paths, we asked the users to fill in two questionnaires: one on personal preferences concerning the interactive map and one on their background using maps. We conducted the study twice. The first time we did a pilot study with nine participants which led us to change the appearance of the time-sliders. Up to that point, the controls for the time window were placed on a separate slider bar. Multiple participants noted that this is not intuitive. Based on that feedback, we switched to the current design, which is inspired by the work of Haslett et al. [1991] and Hochheiser and Shneiderman [2004]. In the second run, we conducted the actual study for eleven days with 41 participants; see Table 7.2. We consider the number of 41 participants who have not taken part in the first phase to be

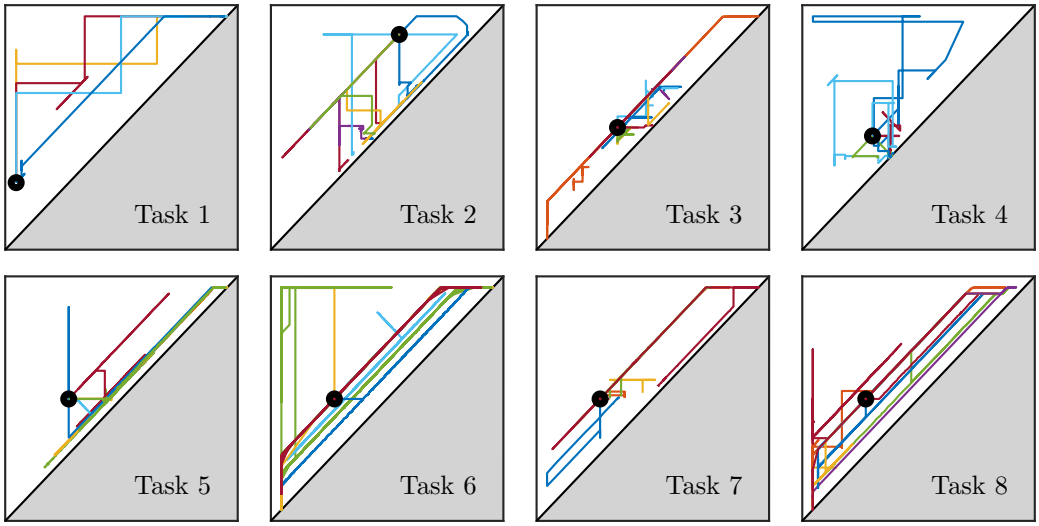


Figure 7.14: Examples of query paths recorded during the study. The query path of each participant is illustrated in a different color. The starting point of the query path is symbolized with a black disc. See Figure 7.7 for the interpretation of the paths' segments.

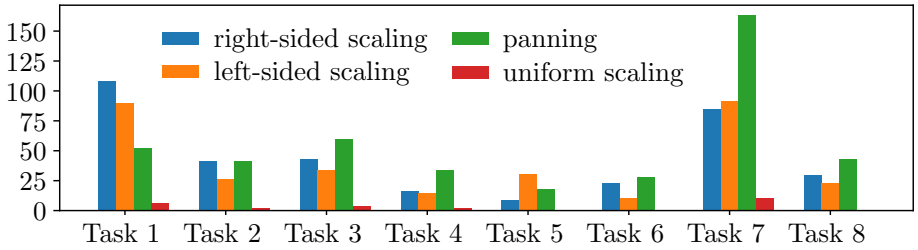


Figure 7.15: Number of basic interactions per participant.

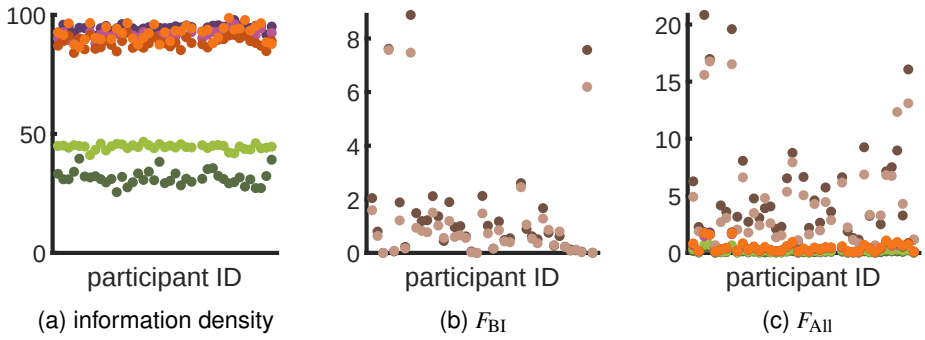
sufficient. The participants were acquired from the research as well as from non-research-related networks. We did not specify whether the participants should complete the study in one sitting. We also asked the participants in the second run for personal feedback; see Table 7.3. They mostly find the use of the time-sliders intuitive and would like to use such maps in practice. Hence, the research on such visualizations is of high relevance. In the free-text answers, the main criticism is that the number of events shown on the map should better reflect the distribution shown in the histogram. We see this as an interesting alternative objective for the scenario that the user wants to explore the spatiotemporal patterns. However, for the use-case of finding one event, e.g., hotel-use-case, we deem that the objective MAXINFO supports the user's needs better. One can also support the visualization of spatiotemporal patterns by displaying all events in the queried time-window as points on the map.

User Strategies for Tasks: Figure 7.14 shows examples of recorded query paths. For each task different strategies can be recognized. For example, in Task 1 some participants preferred left-sided and right-sided scaling (horizontal and vertical segments), while others preferred to first pan the window (diagonal segment). Figure 7.15 shows that mostly left-sided and right-sided scaling as well as panning are used as basic interactions. Uniform scaling was rarely used, which reflects some of the comments that the interaction was less intuitive. One could therefore also omit uniform scaling from the set of basic interactions.[3] The average processing time and the error rate indicate that the tasks were solvable but were still demanding; see Table 7.4. We had a detailed look at Task 6, as several participants did not solve it correctly. It turned out that the possible choices of the answer were misleading, which often resulted in answers that were almost correct. On the account of these observations, we conclude that with this study we have obtained a set of target-oriented query paths that are suitable for evaluating our algorithms.

[3]We note that this would not change our analysis, in particular, optimal activity regions will still be rectangles.

Table 7.4: The performance of the 41 participants, i.e., the number of correct answers and the average response times (in seconds).

Task	1	2	3	4	5	6	7	8
Correct	37	32	38	36	28	24	36	41
Avg. time	47	48	33	84	73	97	20	25



(a) information density (b) F_{BI} (c) F_{All}

Figure 7.16: Results for analysis of the query phase for Task 4. We compare the data structure obtained by the approximation algorithm (Part-D, Part-S), by the greedy heuristic (Greedy-D, Greedy-S), by the combination of the approximation and greedy (Combi-D, Combi-S) with on-demand implementations (OD-D, OD-S) that do not take any consistency criteria into account. (a) The ratio between the weight of the labeling with respect to the optimal weight obtained by OD-D and OD-S, respectively. (b) Average degree of flickering. (c) Average degree of flickering over all queries within a query path P.

Acquired Query Paths: Overall the query set comprises 328 query paths that consist of 136462 time-window queries in total. More specifically we recorded each time-window query sent by the discrete sampling of the web interface during the study[4]. From these time-window queries, we computed the basic interactions. Over all query paths, we obtained 10997 basic interactions.

7.6.4 Query Phase Evaluation

We analyze the information density and consistency with respect to the query paths generated by the study. We compare our presented algorithms to an *on-demand* solution, in which we compute each labeling for each time-window query without considering REPRODUCIBILITY, STABILITY, and CONTAINMENT. This is a well-established approach and we use it as our baseline for comparison. In more detail, given a time-window query, we solve a maximum-weight independent set problem for the set of labels for all events contained in the time window. We obtained these labelings by an ILP formulation and denote them by OD-D and OD-S for unit disks and unit squares as labels, respectively. We build the evaluation on the data described in the study; see Section 7.6.3.

For Task 7 the ILP formulations for OD-S could not be solved in a reasonable time. When analyzing the query paths of Task 7, large time-window queries occurred, which led to large instances of the ILP formulation and high running times. We, therefore, have excluded this task for square labels.

Information Density For each query Q issued in the study, we compare the weight of the labeling received from our pre-computed data structure with the weight of on-demand labeling that we optimized for Q without consideration of consistency (OD-D, OD-S); see Figure 7.16a for Task 4. We refer to the ratio of these weights as *information density*. For both data sets the solutions of Greedy-D to reach at least an information density of 81.92% and the ones of Greedy-S at least 78.86%. The algorithms Combi-D and Combi-S reach 62.33% and 69.54%, respectively. Part-D and Part-S drop behind with at least 16.97% and 20.44%. Hence, concerning information density, the greedy heuristic is the best choice.

[4]We used Javascript for the implementation of the web-interface and sent a time-window query for every triggered "mousemove" event.

Consistency For evaluating the consistency of the time-window labelings, we look at the change of displayed labels between labelings of two consecutive time-window queries. More formally, let $E = \{e_1, \ldots, e_n\}$ be a set of events such that t_i is the timestamp, and ℓ_i the label of event e_i. The label ℓ_i *flickers* between two queries Q and Q' if the timestamp t_i is contained in both Q and Q', and ℓ_i is either contained in L_Q or $L_{Q'}$ but not in both sets. We define the *degree $f_{Q,Q'}$ of flickering* between two consecutive queries Q and Q' as the number of labels that flicker between Q and Q', i.e., $f_{Q,Q'} = |\{e_i \in E_i \mid \ell_i \text{ flickers between } Q \text{ and } Q'\}|$. Hence, the smaller $f_{Q,Q'}$ is, the more the consistency of the visual transition from the time-window labeling L_Q to the time-window labeling $L_{Q'}$ preserved. We assess the effect of the properties STABILITY and CONTAINMENT by the average number of flickering effects per query within the basic interactions. More specifically, let $I_1, \ldots, I_k$ be the basic interactions of a query path P and let $Q_1, \ldots, Q_{r_j}$ be the queries of a basic interaction I_j. The *average degree of flickering of a basic interaction* is defined as

$$F_{\mathrm{BI}} = \frac{1}{m} \sum_{j=1}^{k} \sum_{i=2}^{r_j} f_{Q_{i-1},Q_i} \tag{7.1}$$

where m is the overall number of queries of P. We note that since ILP-D, ILP-S, Part-D, Part-S, Greedy-D, and Greedy-S are based on rectangular activity ranges, we have $F_{\mathrm{BI}} = 0$ for each of these variants. Figure 7.16 shows F_{BI} for OD-D and OD-S. Over all query paths of all participants and all tasks the maximal value of F_{BI} is 18.13 flickering effects per query for OD-D and 12.08 flickering effects per query for OD-S. Hence, within a basic interaction, which consists of 12.41 queries on average, the user encounters 1334.5 and 648.17 for OD-D and OD-S at maximum, respectively. This shows that STABILITY and CONTAINMENT eliminate unnecessary flickering effects.

We further evaluate the design of our model by counting the flickering effects between two consecutive queries over the entire query path consisting of the queries $Q_1, \ldots, Q_m$. Hence, we particularly consider the flickering effects that occur during the transition from one basic interaction to the next. The *average degree of flickering over all queries within a query path P* is defined as

$$F_{\mathrm{All}} = \frac{1}{m} \sum_{i=2}^{m} f_{Q_{i-1},Q_i}. \tag{7.2}$$

Figure 7.13c shows F_{All} for each query path generated by one user for Task 4. Over all query paths of all participants, the maximal value of F_{All} is 0.6 for Part-D, 0.61 for Part-S, 1.32 for Greedy-D, 1.24 for Greedy-S, 1.75 for Combi-D, 1.4 for Combi-S, 37.56 for OD-D, and 26.24 for OD-S. Hence, while in the on-demand solutions, many flickering effects can be observed, at least eighteen times fewer flickering effects can be observed for the constructed activity diagrams. Thus, for the target-oriented usage of our interactive map, we observe a high consistency.

Query Time In our implementation, we have used STR-packed R-trees for the query phase to process the rectangle-stabbing queries. A query took 3 μs on average and 86 μs at maximum over all considered activity diagrams and query paths. In contrast, computing the on-demand solutions OD-D and OD-S took substantially longer, e.g., for Task 6 we obtained 0.62 sec on average and 1.73 sec at maximum. We emphasize that the obtained query times of the data structure allow real-time applications as we have shown in our web application.

7.7 Conclusion

We have presented the λ-structure for map labeling that ensures consistency during basic interactions with a time-slider. As the underlying optimization problem is NP-hard, we focused on efficient non-exact algorithms. On the one hand, we developed approximation algorithms with a theoretical quality guarantee: the weight of the returned solution is at least a certain constant fraction ($1/4$ in the case of unit-square labels and $1/7$ in the case of unit disks) of the weight of an optimal solution. On the other hand, we developed an efficient greedy heuristic. Although the greedy heuristic does not have a constant approximation factor, it performs astonishingly well for real-world instances. In our experiments, the greedy heuristic achieved at least 83.41% of the quality of an optimal solution. We showed that our approach is efficient enough to compute the data structure and support queries for real-world applications. The evaluation of the generated activity diagrams for sequences of queries stemming from our study shows that our model preserves consistency between two consecutive time-window labelings while maintaining a high information density.

8 Part I: Conclusion and Outlook

Conclusion

In the first part of this thesis, we presented three time-windowed data structures for the visualization of event data. As basic visualization approaches, we used a selection of standard techniques from cartography: a polygonal representation, a grid-based density visualization, and a map labeling technique. We showed that all of these data structures allow real-time interaction via time-slider interfaces. Further, we implemented consistency criteria for the λ-structure.

In the following, we comment on how our time-windowed data structures tackle the challenges of visualizing spatiotemporal data.

The spatial visualization should be precise and accurate. With the parametrization of the α-structure (value of α) and the θ-structure (grid and threshold parameter θ), it is possible to control the shape of the resulting visualization. Hence, the α-shape of a point set and the density map of a point set can be tuned to be as accurate as desired. In the extreme case, it is the input point set for α-shapes and an arbitrarily close approximation for the density maps. Nevertheless, reproducing the input is not the goal.

For the λ-structure, we can not guarantee to display all information for every data set. Here, we always require that the visualization is overlap-free for the labels. With our optimization, we preserve as many of the important labels as possible. In particular, the user can control the selection of displayed labels by introducing weights according to their importance.

Hence, we deem that our data structures allow a precise and accurate representation of the data.

The spatial visualization needs to be clear and comprehensible. Producing clear and comprehensible visualizations of large data sets is often done by reducing the information density. With the α-structure, we cluster, aggregate, and represent the point set by a set of polygons. This leads to a simplified visualization, i.e., the polygons consist of fewer corners than the input point set. Depending on the parameter α the visualization gets more or less clear. In the extreme case, the α-shape corresponds to the convex hull.

Also, the θ-structure can be controlled by the choice of the underlying grid and the value chosen for parameter θ. For a regular grid, the resulting output is a schematized representation which is often perceived as clearer. Also, the number of colored grid cells is mostly smaller than the number of points, hence, the visualization is simpler.

For the λ-structure, we improve the clearness of the visualization by requiring that no two labels overlap. Hence, every label is comprehensibly visualized and the overall visualization is clear.

If the events have additional information, we need tools to display them. For visualizing additional information on the data, we introduced the λ-structure that provides solutions for displaying text, photos, or icons on the map. We adapted techniques from map labeling and object selection to provide clear visualizations of the additional information.

The response time needs to be small to handle real-time interactions. To achieve a small response time, we pre-process the data into data structures (α-structure, θ-structure, λ-structure). Hence, not the whole (possibly large) data set needs to be queried and processed, but we can directly obtain the visualization. As mentioned before, the number of reported geometries is most of the time considerably smaller than the input data since we want to have a clear visualization. Hence, the query time is smaller. Also, for the θ-structure, we took particular care about implementing appropriate structures that optimize the query time.

We want to emphasize that we provide theoretical analyses of the asymptotic running time for our data structures. Further, we performed real-world experiments that showed a massive improvement in the query time in comparison to on-demand implementations. Even for large data sets the query time was always below 3 seconds for the α-structure, below 60 milliseconds for the θ-structure, and below 90 microseconds for the λ-structure.

The system needs to be capable of large data sets. When designing our time-windowed data structures, we paid special attention to their storage efficiency. Particularly, we do not store the visualizations for every possible time-window query but we store the components and assemble them for a time-window query. For the α-structure we proved a memory consumption of $O(n^2)$ where n is the number of events. Our experiments with real-world data suggest a linear relationship between the number of events and the memory consumption which makes it applicable in practice. For the θ-structure we provide an asymptotic bound of $O(n \log m + m)$ where n is the number of events and m is the number of grid cells. The size of the λ-structure is linear with the number of events since we compute for every event one activity box.

When moving the time-slider the changes in the spatial visualization should not be distracting. Currently, we have implemented stability constraints for time-slider interaction only for the λ-structure. We implement this constraint by prohibiting a label to disappear and appear again during a continuous basic interaction with the time-slider interface. For the α- and θ-structure, we have so far decided against demanding stability constraints during the interaction, as this would introduce inaccuracies in the visualization. Nevertheless, we deem that it would be beneficial to explore whether these inaccuracies or distracting flickering effects have negative effects on the user.

Outlook

In the following, we list challenges and ideas that we have not faced yet.

- A follow-up to the work presented so far is the exploration of data structures that support **different filter interactions**. For example, we suggest focusing on data that corresponds to a time range instead of a point in time, e.g., a flooding event that takes a certain amount of days or the trajectory of a car trip or hike that takes some hours. Then, one would like to filter for data that covers a certain point in time, i.e., we search for one point in time instead of a time window. For this scenario, we face similar challenges as already presented, especially for big data, querying the actual data and computing the visualization on-demand can be time-consuming and not fast enough for real-time interaction.
- Beside the three presented visualization approaches, there exists a wide range of information visualization techniques. We deem it interesting to investigate which of these **additional visualization techniques** can be combined with time-window slider interfaces. Examples of other visualization techniques for spatial data are Dorling cartograms, choropleth maps, and bubble charts. But also the visualization of abstract data, for example, pie charts, drawings of graphs, etc., is worth investigating in combination with time-slider interfaces. Due to the high amount of temporal data, we suppose that for all of these visualizations there exist application scenarios where it is reasonable to combine them with a time-slider interface. We suggest investigating whether querying for such a visualization with a time-windowed query on-demand is sufficiently fast or whether it is necessary and possible to pre-compute the data into a time-windowed data structure.
- If the data volumes to be managed in the data structures continue to grow, it is certainly also important to achieve further improvements in memory consumption. This could be implemented at the expense of the precision and completeness of the visualization, i.e. an **approximation of the actual visualization** is provided. For the α-structure, we already introduced the aggregation on a grid. Similarly, one could do a spatial binning for the θ- and λ-structure. Another option is a temporal binning of the timestamps. While all these approximation approaches aim at a reduction of storage consumption, new visualization challenges arise. It is not straightforward to find an appropriate visualization that reflects the approximation error.
- As mentioned before, adaptations of the α-structure and θ-structure that ensure **consistency during basic interactions** are interesting to investigate. Enforcing consistency might introduce inaccuracies in the visualization where it is necessary to discuss a balance between accuracy and usability. It would be interesting to perform user studies that investigate which type of consistency is needed and wanted by the users. Then, formalizing such results from user experiments into mathematical models and implementing them is another challenge.

Besides these challenges and ideas that are very closely related to our presented research, we also want to comment on a higher level of interesting related research fields.

- For the presented data structures we restricted ourselves to queries (filtering) in one dimension (the time). This decision is reasonable because spatio-temporal data is very common and filtering by time dimension is very convenient. Nevertheless, for multidimensional data, applications with **multidimensional filtering** can also be relevant. While there exists research on data structures for efficient multi-dimensional filtering and visualization (e.g., NanoCubes [Lins et al., 2013], imMens [Liu et al., 2013], and Hashed-

cubes de Lara Pahins et al. [2017]), we suggest aiming at more sophisticated cartographic visualizations. For example, a polygonal aggregation of all data points that are contained in the query. The design of the data structure should be a small query-time that enables real-time interaction while having a small storage consumption.

- So far, we have assumed in our model that all data are exact. However, in reality, this is not the case, and the **data is associated with uncertainties**. We propose to give more freedom to the visualization using these data uncertainties. If, for example, there are uncertainties in the position, the labels of the λ-structure could be shifted within the limits of the uncertainties to place more labels without an overlap. If there is an inaccuracy in the time stamp, the time function of the θ-structure can be optimized. Another example is to exclude events with high uncertainties from the visualization or to add the visualization of the uncertainties to the visualization and incorporate this into the time-windowed data structures. Also, one could introduce uncertainty as another filtering dimension. That means the data structure is optimized for different uncertainty levels and time-window queries. Then, the user can comfortably explore data within a time window that fulfills the uncertainty requirements.
- Another application field where we deem it to be promising to apply presented techniques is the **visualization and exploration of knowledge graphs**. A knowledge graph is a graph that encodes information about real-world objects, entities, concepts, etc., and their relationships. Knowledge graphs have become increasingly important in recent years and are widely used, e.g., Wikidata. Visualizing knowledge graphs can be extremely difficult, as the graphs can quickly become very large (e.g., Wikidata has more than 100 million nodes). Adequate filtering and visualization can improve this immensely and enable real-time interactions.
- We deem that pursuing **interdisciplinary research** is important. Often, these projects are more difficult (e.g., different terminologies are used in the different research branches) and also publication can be challenging if there is no perfectly fitting journal or conference. Nevertheless, combining knowledge enables solving problems more holistically. For example, for geospatial visualization, combining research from computer science, cartography, and human perception allows one to develop solutions based on cartographic guidelines that can be experimentally evaluated with human perception studies and one can derive algorithms for solving the resulting formalized problem.

Part II

Hulls of Polygons

Part II: Introduction

In this part, we discuss hulls of polygons that allow us to have quickly a clear overview of possibly complex spatial data. We consider a polygon with a complex boundary as input and we want to represent the polygon by a polygonal hull. An example of such a polygonal hull of a polygon is its convex hull. Since the convex hull often covers large areas that are not covered by the input polygon, it is not a good representation. We aim for polygonal hulls that balance clearness and precision, i.e., the hull should be simple while containing the characteristics of the input polygon. In cartography, this problem is part of simplification and schematization [Hake et al., 2002]. Typically a simplified polygon has fewer corner vertices than the original polygon. A schematized polygon typically restricts the directions of its edges to a pre-defined set of directions, e.g., an octilinear polygon consists of edges with either horizontal, vertical, or diagonal directions. In this work, we require that the polygonal hull contains the input polygon.

We structure the second part as follows. First, we discuss existing work related to simplification and schematization from the fields of cartography and computational geometry. Afterward, we present shortcut hulls. With shortcut hulls, we focus on the simplification of the input polygon. Then, we give an outlook on preliminary work for hulls that simplify and schematize the input polygon. At last, we summarize this part and this thesis with an outlook and a conclusion.

9 Part II: Related Work on Simplification and Schematization

Simplification and schematization are two major research fields in cartography and computational geometry. Especially, they are part of the list of generalization operators defined by Hake et al. [2002]. Map generalization is especially important to reduce a map user's cognitive load or when generating maps of a smaller scale from existing maps. In order to fit the information on a smaller part of the display, the generalization tools aim at providing a legible and clear map that still preserves as much information as possible [Burghardt et al., 2007].

A common technique to simplify geometries is to reduce the number of vertices of the simplified geometry with respect to the input geometry. Often, the vertices of the simplified geometry are a subset of the input geometry which is called *vertex-restricted simplification*. Typically schematization is implemented by restricting the edges of the geometries to a predefined set of directions, e.g., rectilinear, octilinear, $\mathscr{C}$-directed.

Application on Real-World Data Typical data sets for simplification and schematization are administrative boundaries [Barkowsky et al., 2000, Buchin et al., 2016, van Dijk et al., 2014], building footprints [Haunert and Wolff, 2010, van Kreveld et al., 2013], or metro maps [Jacobsen et al., 2021, Nöllenburg, 2014, Wu et al., 2020]. In particular, as a motivation for outer simplifications of polygons, we point out the generalization of isobathymetric lines in sea charts where the simplified line should lie on the downhill side of the original line to avoid the elimination of shallows [Zhang and Guilbert, 2011]. A motivation for polygonal hulls of geometric graphs stems from travel-time visualization, where a common aim is to compute a polygon that contains the part of a transport network that is reachable within a given amount of time from a given origin [Baum et al., 2018, Forsch et al., 2021].

Simplification and Schematization of a Polyline For the simplification of polylines, we refer to the Douglas-Peucker algorithm [Douglas and Peucker, 1973], which is most widely applied in cartography, and similar approaches [Abam et al., 2010, Neyer, 1999, Pallero, 2013]. Other methods for polyline simplification are based on a Delaunay triangulation [Ai et al., 2017, 2006, 2014].

Simplification and Schematization of a Polygonal Subdivision A closely related field is the simplification and schematization of polygonal subdivisions [Buchin et al., 2016, Estkowski and Mitchell, 2001, Mendel, 2018, Meulemans et al., 2010, van Goethem et al., 2015]. Another method addresses not only simple polygons but a polygonal subdivision of the space [Buchin et al., 2016, van Goethem et al., 2015, Meulemans et al., 2010]. The method schematizes and simplifies this subdivision with a limited set of orientations while preserving each polygonal area and the topology.

Simplification and Schematization of a Polygon Considering a polygon as input geometry, a basic technique for simplification is the convex hull [Alegría et al., 2021, Daymude et al., 2020, Fink and Wood, 2004, Rawlins and Wood, 1987]. A schematization approach is $\mathcal{O}$-hulls [Fink and Wood, 2004] which we discussed in Chapter 2. Another example is the schematized α-shape presented in Chapter 5. Multiple other approaches for polygonal hulls of polygons exist—some of the corresponding computational problems can be solved in polynomial time [Haunert and Wolff, 2010], whereas others have been shown to be NP-hard [Haunert et al., 2008].

From a computational point of view, computing a schematic hull of a polygon is related to path-planning tasks in which the output path has to avoid a set of polygonal obstacles and the path's edges are constrained to a prescribed set of orientations [Adegeest et al., 1994, Lee et al., 1996, Mitchell et al., 2014, Speckmann and Verbeek, 2018].

Simplification and Schematization of a Set of Heterogeneous Geometries For the case that multiple geometric objects are given as input, there exist several techniques for detecting groups of objects and aggregating each group into a single object. Often a partition of the plane is used as a basis for assembling the output objects. For example, it is common to select a set of triangles from a Delaunay triangulation of a point set to derive a polygonal representation—frequently applied methods of this type are α-shapes [Edelsbrunner et al., 1983] and characteristic shapes [Duckham et al., 2008]. Another approach for delineating a point set is based

on shortest paths [de Berg et al., 2011]. Some methods yield a set of schematized polygons covering a given point set [Bonerath et al., 2019, van Kreveld et al., 2013]. Similar to α-shapes for point sets, there also exist triangulation-based methods for sets of polygons. A common aim is to compute a set of polygons such that each of them covers a group of input polygons [Jones et al., 1995, Chazelle, 1990, Li and Ai, 2010, Rottmann et al., 2021, Sayidov and Weibel, 2019, Steiniger et al., 2006]. For the case that a set of polylines is given as input, there exists work on computing an enclosing simple polygon based on a Delaunay triangulation [Ai et al., 2017].

10 Part II: Formalization

In the following, we give a formalization of the problem that we discuss in the second part of this thesis. At first, we introduce geometrical concepts and afterward, we discuss hulls and their desirable properties.

Weakly-Simple Polygons Let R be a cyclic sequence $(l_1,\dots,l_n,l_1)$ of directed straight-line segments such that the endpoint of l_i is the starting point of l_{i+1} with $1 \leq i \leq n$ and $l_{n+1} := l_1$. We call R a *polygonal ring*. A *weakly-simple polygon* P is a region whose boundary consists of a polygonal ring O, which we call the *outer ring* of P, and possibly multiple polygonal rings $H_1,\dots,H_k$, which we call the *inner rings* of P, such that there is a connected planar straight-line graph $\mathscr{G}$ that supports O and $H_1,\dots,H_k$ in the following manner: The outer ring O corresponds to the facial walk along the outer face of $\mathscr{G}$ and every inner ring H_i $(1 \leq i \leq k)$ corresponds to the facial walk along an inner face of $\mathscr{G}$; see Figure 10.1. We call an inner face of $\mathscr{G}$ whose facial walk corresponds to an inner ring of P a *hole* of P. This allows us to define the boundary, exterior, and interior of P as follows. The boundary $\partial P \subseteq \mathbb{R}^2$ of P is the point-wise union of all the straight-line segments in O and $H_1,\dots,H_k$. The exterior $\operatorname{Ext} P \subseteq \mathbb{R}^2$ of P is equal to the union of the outer face of $\mathscr{G}$ with the holes of P. The interior of P is $\operatorname{Int} P = \mathbb{R}^2 \setminus (\operatorname{Ext} P \cup \partial P)$. We call a weakly-simple polygon that has no inner rings, i.e., it has no holes, a *weakly-simple region*.

In this work, a *vertex of* P refers to a visit of a vertex of $\mathscr{G}$ during a facial walk along one of the faces of $\mathscr{G}$. This particularly means that P may have multiple vertices at the same location; see Figure 10.2. We refer to the directed straight-line segments of O and $H_1,\dots,H_k$ as *edges* of P. Further, we refer to a directed straight-line segment as *shortcut* if it connects two vertices of P and does not intersect the interior of P. We call a line segment a *tangent* of P if it has more than one point with a segment of P in common and it does not intersect the interior of P.

We make the following observations. First, not every inner face of $\mathscr{G}$ is necessarily a hole of P; see faces $g_1,\dots,g_4$ in Figure 10.1. Second, requiring $\mathscr{G}$ to be connected ensures that the holes of P are enclosed by O. Third, the edges of $\mathscr{G}$ that are neither incident to the outer face of $\mathscr{G}$ nor to the holes of P serve as connections between inner rings as well as between the outer ring and the inner rings; they can be chosen freely subject to the constraints above. Fourth, every simple polygon is a weakly-simple polygon, but weakly-simple polygons are more general than simple polygons. They can be used to describe more complex geometric objects, such as a Euclidean minimum spanning tree of a point set; see Figure 10.3.

Hulls The *hull* of a weakly-simple polygon P is another weakly-simple polygon that contains P. Let $\mathscr{C}$ be a set of directed straight-line segments. A *$\mathscr{C}$-hull* is a weakly-simple polygon (possibly with holes) whose oriented boundary consists only of segments from $\mathscr{C}$ and that contains P.

In this thesis, we consider shortcut hulls (see Chapter 11) where we restrict $\mathscr{C}$ to be a set of shortcuts. Hence, all vertices of a shortcut hull are vertices of the input polygon. Here, we consider both weakly-simple polygons and weakly-simple regions as input polygons. A shortcut hull is again a weakly-simple polygon.

Optimal Hulls Several $\mathscr{C}$-hulls might exist for a given weakly-simple polygon. We want to find hulls that simplify and/or schematize the input polygon P while preserving its main characteristics. We model this objective using cost functions for weakly-simple polygons.

1. We aim for a $\mathscr{C}$-hull Q of a weakly-simple polygon P that has a similar shape as P. Hence, we consider costs $c_{2\mathrm{d}}(P)$ for the area of Q. In particular, the value of $c_{2\mathrm{d}}(Q)$ is minimal if $P = Q$.
2. We want to simplify the shape of the input polygon. Hence, we introduce costs for the perimeter $c_{1\mathrm{d}}(Q)$ of Q. We implement $c_{1\mathrm{d}}$ as the sum of the costs for each segment of the outer ring and the inner rings (if any) of Q. One example of the cost of a segment is its Euclidean length.

We allow the user the specify a balancing factor $\lambda \in [0,1]$ with which we summarize these two cost functions into the costs $c(Q)$ of a $\mathscr{C}$-hull.

$$c(Q) = \lambda \cdot c_{1\mathrm{d}}(Q) + (1-\lambda) \cdot c_{2\mathrm{d}}(Q). \tag{10.1}$$

We call the $\mathscr{C}$-hull of P that minimizes the costs c over all admissible $\mathscr{C}$-hulls of P the *optimal $\mathscr{C}$-hull*.

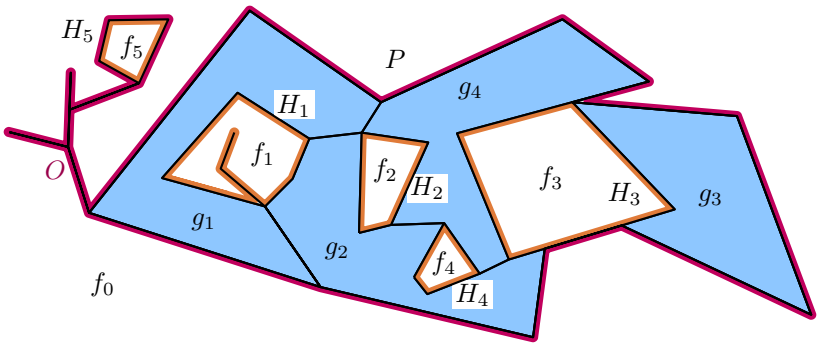
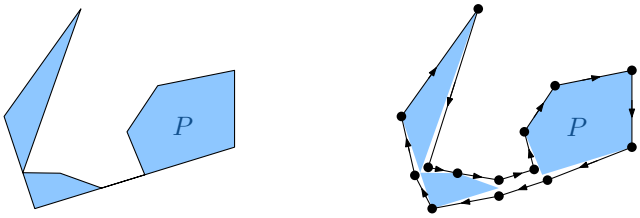


Figure 10.1: A weakly-simple polygon P with outer ring O (lilac) and five inner rings $H_1, \ldots, H_5$ (orange). The planar straight-line graph $\mathcal{G}$ (black) supports O and $H_1, \ldots, H_5$. The face f_0 is the outer face of $\mathcal{G}$. The faces $f_1, \ldots, f_5$ are the holes of P. The faces $g_1, \ldots, g_4$ are the interior of P. (For better legibility the outer ring, as well as the inner rings, are drawn with some offset.)



(a) weakly-simple polygon P (b) vertices of P

Figure 10.2: The vertices of a weakly-simple polygon. For clarity, vertices with the same spatial location are slightly displaced.

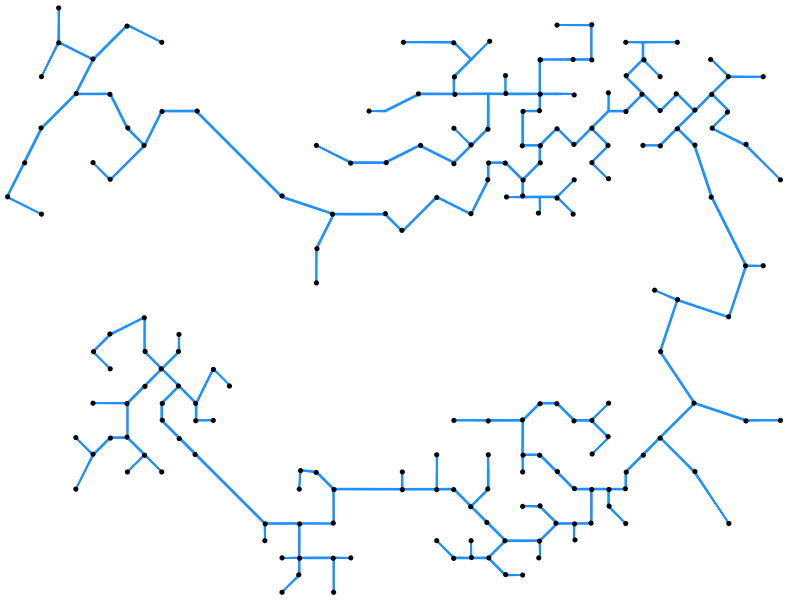


Figure 10.3: A minimum spanning tree T (blue edges) of a set of points.

11 Shortcut Hulls: Vertex-Restricted Outer Simplifications of Polygons

This chapter is mainly taken from joint work with Jan-Henrik Haunert, Joseph Mitchell, and Benjamin Niedermann. We presented it at the Canadian Conference of Computational Geometry (2021) [Bonerath et al., 2020a]. Further, an extended version is published in the journal Computational Geometry Theory & Application [Bonerath et al., 2023b]. We want to acknowledge Nikolas Schwarz who is a student at the University of Konstanz, Germany. He pointed us to (i) a mistake in the estimation of the size of C^+ where we forgot to incorporate the number of crossing components, and (ii) a problem in our analysis of the running time where we assume that the crossing components have been pre-computed prior to our algorithm. We adapted our manuscript accordingly.

Abstract

Let P be a polygon and $\mathscr{C}$ a set of shortcuts, where each shortcut is a directed straight-line segment connecting two vertices of P. A shortcut hull of P is another polygon that encloses P and whose oriented boundary is composed of elements from $\mathscr{C}$. We require P and the output shortcut hull to be weakly-simple polygons, which we define as a generalization of simple polygons. Shortcut hulls find their application in cartography, where a common task is to compute simplified representations of area features. We aim at a shortcut hull that has a small area and a small perimeter. Our optimization objective is to minimize a convex combination of these two criteria. If no holes in the shortcut hull are allowed, the problem admits a straightforward solution via the computation of shortest paths. For the more challenging case in which the shortcut hull may contain holes, we present a polynomial-time algorithm that is based on computing a constrained, weighted triangulation of the input polygon's exterior. We use this problem as a starting point for investigating further variants, e.g., restricting the number of edges or bends. We demonstrate that shortcut hulls can be used for the schematization of polygons.

11.1 Introduction

The simplification of polygons finds a great number of applications in tasks related to cartography or geographical information systems. For example in map generalization, the simplification of polygons is used to obtain abstract representations of area features such as lakes or buildings. A common technique, which originally stems from polyline simplification, is to restrict the resulting polygon Q of a polygon P to the vertices of P, which is also called a *vertex-restricted simplification* [Driemel and Har-Peled, 2013, Filtser and Filtser, 2021, Meulemans, 2014].

In this chapter, we consider the vertex-restricted crossing-free simplification of a polygon P considering only shortcuts that do not intersect the interior of P. All shortcuts satisfying this requirement are termed *possible shortcuts*. However, since there can be additional application-specific requirements, not all possible shortcuts may be *admissible*. In contrast to other work, we consider the admissible shortcuts as input for our problem and do not require special properties, e.g., that they are crossing-free or that they are within a prescribed buffer of P. The result of the simplification is a *shortcut hull* Q of P, with Q possibly having holes. We emphasize that the edges of a shortcut hull do not cross each other. Figure 11.1 shows polygons (blue area) and different choices of shortcut hulls (blue and red area)—in this example all possible shortcuts are admissible. Such hulls find their application when it is important that the simplification contains the polygon. Figure 11.2 shows the simplification of a network of lakes, whose complement is a (green) polygon; note that the lakes are connected to the exterior of the green polygon at the bottom side of the green polygon's rectangular outer boundary. In this application, it can be desirable that the simplification results in the water area being decreased, while preserving the land area, which may contain important map features. Another example covered by our approach is Figure 11.3 where we have a spanning tree as input.

The degree of the simplification of Q can be quantified in terms of its perimeter and enclosed area. While a small perimeter may indicate a strong simplification of P, a small area gives evidence that Q adheres to P. In the two extreme cases, Q is either the convex hull of P (minimizing the perimeter), or Q coincides with P (minimizing the enclosed area). We present algorithms that construct shortcut hulls of P that seek a balanced optimization,

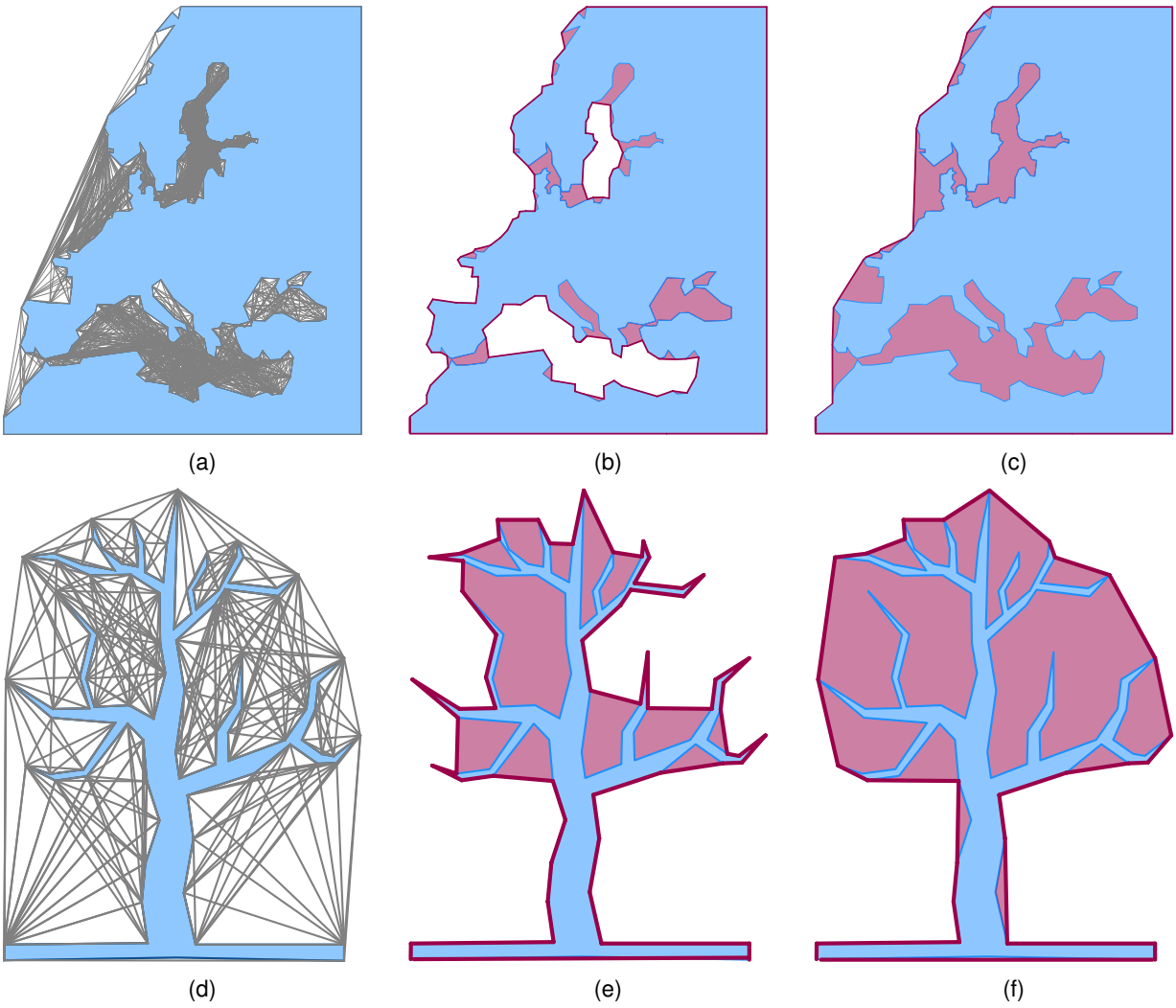
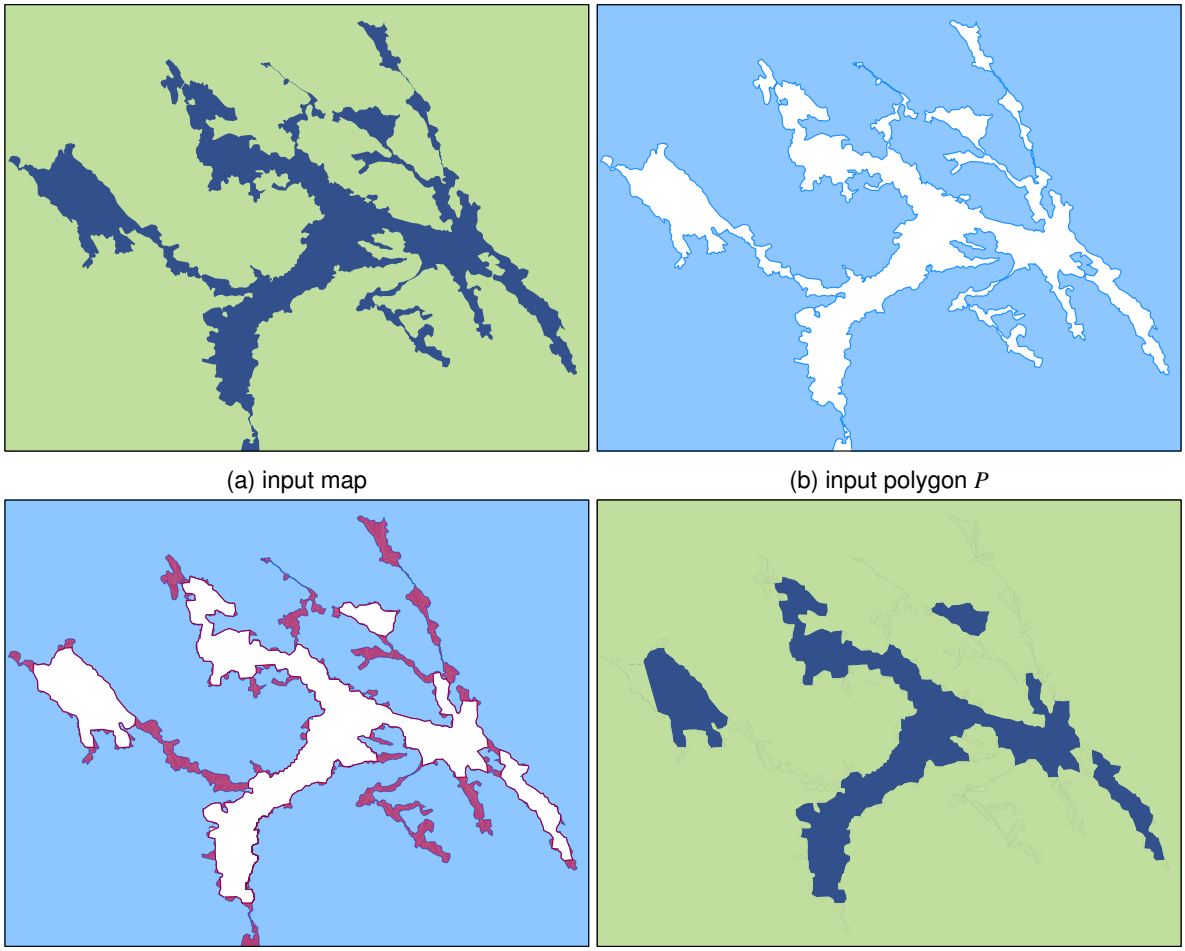


Figure 11.1: 1st column: Input polygon (blue) with a set $\mathscr{C}$ of all possible shortcuts (gray). 2nd–3rd columns: Optimal $\mathscr{C}$-hulls (blue and red area) for different choices of λ. For clarity, we omit the directions of the edges in this and a majority of the other figures.

minimizing a convex combination of these two competing criteria, with the combination specified by a parameter $\lambda \in [0,1]$, which determines the relative importance of perimeter minimization versus area minimization. We show that for the case that Q must not have holes we can reduce the problem to that of finding a cost-minimal path in a directed acyclic graph that is based on the given set of admissible shortcuts. However, especially for applications in cartography, which require the simplification of spatial structures, we deem the support of holes in the simplification as an essential key feature. For example, in Figure 11.2d the connections between the lakes are not preserved in the simplification shown, as they are very narrow, while it is desirable to preserve the large lakes. We, therefore, investigate the computation of shortcut hulls that have holes in greater detail. As input, we require a weakly-simple polygon, whose boundary is composed of polygonal rings. We formally define these concepts as follows.

Formal Problem Definition We are given a weakly-simple region P and a set $\mathscr{C}$ of directed edges within $\partial P \cup \operatorname{Ext} P$ such that the endpoints of the edges in $\mathscr{C}$ are vertices of P. We particularly require that an edge $e = pq \in \mathscr{C}$ is directed such that the round walk from p to q along e and from q to p along the boundary of P is oriented clockwise and encloses P; see Figure 11.4a. The elements in $\mathscr{C}$ are the admissible shortcuts. A *$\mathscr{C}$-hull* is a weakly-simple polygon (possibly with holes) whose oriented boundary consists only of directed edges from $\mathscr{C}$ and that contains P. We seek a $\mathscr{C}$-hull Q that minimizes a linear combination of the perimeter and the enclosed area of Q. Formally, we define the *cost* of a $\mathscr{C}$-hull Q to be

$$c(Q) = \lambda \cdot c_{1\mathrm{d}}(Q) + (1-\lambda) \cdot c_{2\mathrm{d}}(Q), \tag{11.1}$$



(a) input map (b) input polygon P

(c) optimal shortcut hull Q (d) simplified map

Figure 11.2: Simplification of a network of lakes in Sweden. From (a) to (b): the input map is transformed into an input polygon P (blue, in (b)) such that the lakes are in the exterior of P. From (c) to (d): the interior (blue and red area) of the optimal shortcut hull Q of P becomes the land area (green) of the simplified map, while the holes of Q (white) become the simplified lakes (blue).

where $\lambda \in [0,1]$ is a given constant parameter that specifies the relative importance of the perimeter $c_{1d}(Q)$ and the area $c_{2d}(Q)$ of Q. Further, Q is *optimal* if for every $\mathscr{C}$-hull Q' of P it holds $c(Q) \leq c(Q')$.

SHORTCUTHULL.
input: A weakly-simple region P and with n vertices, and a set $\mathscr{C}$ of shortcuts of P, and $\lambda \in [0,1]$
output: An optimal $\mathscr{C}$-hull Q of P (if it exists).

We point out that an optimal $\mathscr{C}$-hull of a weakly-simple polygon P exists if a subset of $\mathscr{C}$ forms a polygonal ring R that contains P. One way to ensure the existence of an optimal $\mathscr{C}$-hull is, hence, to include every edge of P in $\mathscr{C}$. However, depending on the application this is not always desirable. For example, for schematized output polygons one may only want to consider edges with specific orientations. We note that $|\mathscr{C}| \in O(n^2)$, since the edges of $\mathscr{C}$ have their endpoints at vertices of P.

We want to remind the reader that an (optimal) $\mathscr{C}$-hull may contain holes. With this it is possible to fill up narrow parts of the input polygon's exterior; see Figure 11.1b. This allows us to reduce the perimeter of the computed hull without making the new covered area too large. Nevertheless, we also provide a solution for the special case that the computed $\mathscr{C}$-hull must not have any hole.

Further, we want to point out that by forbidding the input polygon P to have holes we aim at keeping the definition of the problem and the presentation of the algorithms simple. However, we could easily deal with holes in the input polygon by simplifying the polygonal rings of P independently of each other.

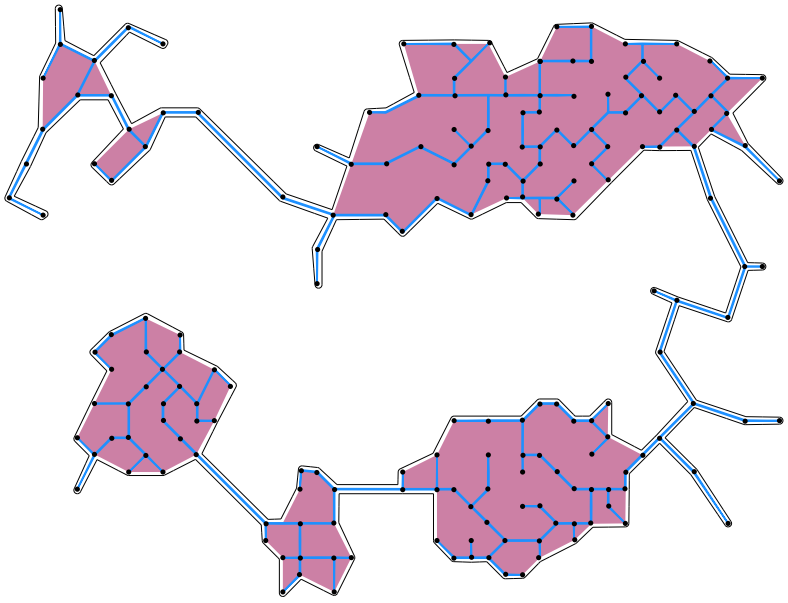
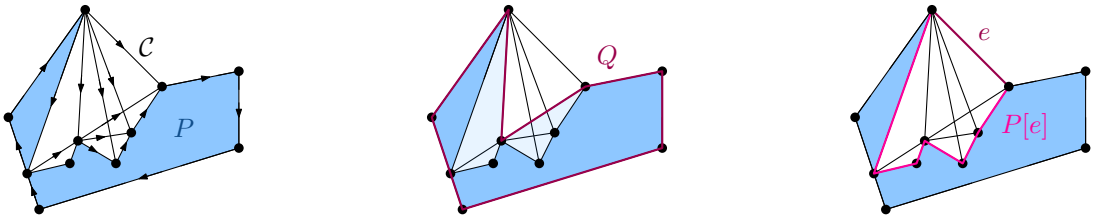


Figure 11.3: Shortcut hull Q of a minimum spanning tree T (blue edges). The boundary of Q is sketched with some offset (black edges). Edges of T that occur twice on the boundary of Q are also accounted twice for the perimeter of Q.



(a) input P and $\mathscr{C}$ (b) $\mathscr{C}$-hull Q (c) pocket $P[e]+e$

Figure 11.4: The input, a solution, and a subinstance for an instance of the problem.

Our Contribution We first discuss how to construct an optimal $\mathscr{C}$-hull in $O(|\mathscr{C}|)$ time for the special case in which the computed $\mathscr{C}$-hull is required not to have holes (Section 11.3). Afterward, we turn to the general case in which the $\mathscr{C}$-hull may have holes (Sections 11.4–11.6). In particular, we show that finding an optimal $\mathscr{C}$-hull Q of P is closely related to finding a triangulation T of the exterior of P and assigning each triangle $\Delta \in T$ either to the interior or to the exterior of Q; see Figure 11.5a. We present an algorithm that solves ShortcutHull in $O(n^2)$ time if we forbid holes and in $O(n^3)$ time in the general case. Moreover, in the case that the edges of $\mathscr{C}$ do not cross each other, it runs in $O(n)$ time. More generally, we analyze the running time based on the structure of $\mathscr{C}$. Let S be the region between P and the convex hull of P. Let G be the *crossing graph* of $\mathscr{C}$, i.e., each node of G corresponds to an edge in $\mathscr{C}$ and two nodes of G are adjacent if the corresponding edges in $\mathscr{C}$ cross each other. The *spatial complexity* of $\mathscr{C}$ is the smallest number $\chi \in \mathbb{N}$ such that for every connected component of G the corresponding edges in $\mathscr{C}$ can be enclosed by a polygon with χ edges whose interior is disjoint with P and only consists of vertices from P; see Figure 11.5. We call the number h of connected components in G the *number of crossing components*. For the special case that no two edges of $\mathscr{C}$ cross each other, i.e., each connected component is a single node, we define $\chi = 1$ and $h = 1$. We show that the proposed algorithm runs in $O(h\chi^3 + n\chi)$ time. We emphasize that $\chi \in O(n)$. Moreover, we present two variants of $\mathscr{C}$-hulls that restrict the number of permitted edges or bends. We further discuss relations of shortcut hulls with respect to problems from applications in cartography and computational geometry (Section 11.7).

11.2 Related Work

In Chapter 9, we discussed related work on simplification and schematization. In the following, we want to add to this related work on triangulations of polygons. Triangulating a polygon is widely studied in computational

(a) $\chi = 1$ (b) $\chi = 7, h = 3$

Figure 11.5: Two examples of the set $\mathscr{C}$ (gray edges) with different spatial complexities χ. (a) The set $\mathscr{C}$ forms a $\mathscr{C}$-triangulation and an arbitrary $\mathscr{C}$-hull (lilac) is shown. Each connected component of the crossing graph of $\mathscr{C}$ is a single node. (b) The crossing graph of $\mathscr{C}$ has three connected components (enclosed by a blue, yellow, and lilac polygon) that are not single nodes.

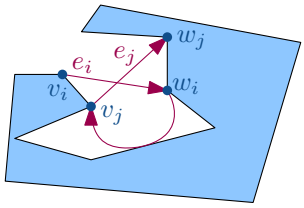


Figure 11.6: Illustration of proof for Theorem 1. Due to the order of the vertices of P the edges e_i and e_j cannot be both part of S.

geometry. Triangulation of a simple polygon can be done in worst-case linear time [Chazelle, 1991]. A polygon with h holes, having in total n vertices, can be triangulated in $O(n \log n)$ time [Garey et al., 1978] or even $O(n + h \log^{1+\varepsilon} h)$ time [Bar-Yehuda and Chazelle, 1994]. Our approach is particularly related to minimum-weight triangulations [Shamos and Hoey, 1975] and constrained triangulations [Chew, 1989, Chin and Wang, 1998, Kao and Mount, 1992, Lee and Lin, 1986, Shewchuk and Brown, 2015]. To summarize, a large number of triangulation-based methods for simplification tasks exist, but these usually keep a given triangulation fixed. In contrast, we present a method that yields a simplified representation of a polygon by optimizing over all triangulations of a polygon's exterior.

11.3 Shortcut Hulls without Holes

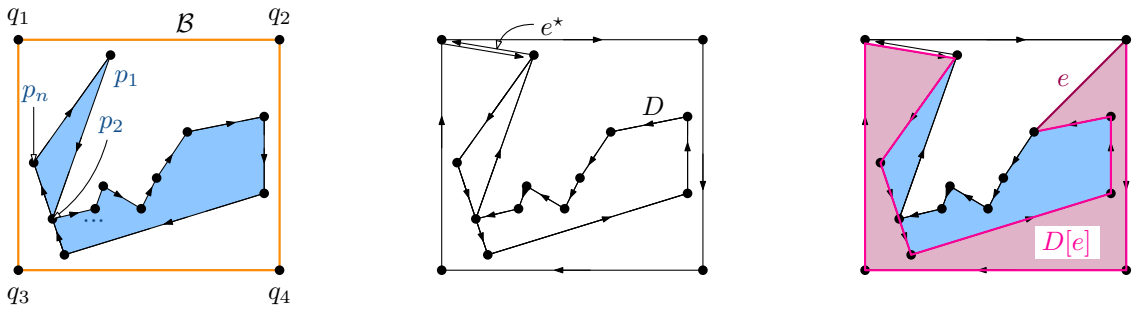
Let $G_{\mathscr{C}}$ be the directed graph induced by the directed edges in $\mathscr{C}$. We call $G_{\mathscr{C}}$ the *geometric graph* of $\mathscr{C}$. If we do not allow the shortcut hull to have holes, we can compute an optimal $\mathscr{C}$-hull Q based on a cost-minimal path in $G_{\mathscr{C}}$; see Figure 11.4b. For each edge $e \in \mathscr{C}$, let $P[e]$ be the section of the outer ring of P between the starting point and the endpoint of e. We call the polygon describing the area enclosed by e and $P[e]$ the *pocket* of e; see Figure 11.4c. For each edge e we introduce costs that rate the length $c_{1\mathrm{d}}(e)$ of e as well as the area $c_{2\mathrm{d}}(P[e])$ of the pocket of e with respect to λ, i.e. $c(e) = \lambda \cdot c_{1\mathrm{d}}(e) + (1-\lambda) \cdot c_{2\mathrm{d}}(P[e])$.

Observation 1. *The vertices of the convex hull of P are part of the boundary of any shortcut hull of P.*

Due to Observation 1, any $\mathscr{C}$-hull of P contains the topmost vertex v_{top} of P. Hence, $G_{\mathscr{C}}$ does not contain any edge e that contains v_{top} in its pocket, and when removing v_{top} from $G_{\mathscr{C}}$ we obtain a directed acyclic graph. In Theorem 1 we use this property to prove that a cost-minimal path in $G_{\mathscr{C}}$ corresponds to an optimal $\mathscr{C}$-hull.

Theorem 1. *The problem* SHORTCUTHULL *without holes can be solved in $O(|\mathscr{C}|)$ time. In particular, in the case that the edges in $\mathscr{C}$ do not cross each other it can be solved in $O(n)$ time and in $O(n^2)$ time otherwise.*

Proof. Let v_{out} be a copy of the topmost vertex v_{top} of $G_{\mathscr{C}}$ that is only incident to all outgoing edges and analogously let v_{in} be a copy v_{top} that is incident to all incoming edges. Let $S = (e_1, \ldots, e_l)$ be the sequence of edges of the shortest path in $G_{\mathscr{C}}$ starting at v_{out} and ending at v_{in}. Let Q be the polygon that we obtain by interpreting S as a polygon. We show that Q is an optimal $\mathscr{C}$-hull. In particular, we need to show that Q is crossing-free. Due to the definition of $G_{\mathscr{C}}$, the following two properties hold: (i) each edge $e = vw$ of $G_{\mathscr{C}}$ starts and ends on the boundary of P and (ii) e is directed such the starting point of e is the starting point of $P[e]$ and the endpoint of e is the endpoint of $P[e]$. Hence, the vertex v appears before w on the boundary of P when going along P starting at its topmost point. Assume that the edges $e_i = v_i w_i$ and $e_j = v_j w_j$ with $1 \leq i < j \leq l$ cross; Figure 11.6.



(a) $\mathscr{B}$ (orange polyline) (b) sliced donut D (boundary in black) (c) $D[e]$ (light pink polyline) and pocket of e (dark pink polygon)

Figure 11.7: Containing box $\mathscr{B}$ and sliced donut D of P.

Since $i < j$, the starting points and endpoints of e_i and e_j appear in the order $v_i w_i v_j w_j$ on S. Due to properties (i) and (ii), v_j lies in the pocket of e_i, and hence, they appear in the order $v_i v_j w_i w_j$ on P. However, this implies that the sequence $S_{i,j} = (e_{i+1}, \ldots, e_{j-1})$ contains an edge e whose starting point is the endpoint of $P[e]$ and whose endpoint is the starting point of $P[e]$, which contradicts Property (ii).

The computation of the shortest path in a directed acyclic graph with $|\mathscr{C}|$ vertices and edges takes $O(|\mathscr{C}|)$ time [Cormen et al., 2009]. In particular, when no two edges of $\mathscr{C}$ cross, we obtain $O(n)$ running time and otherwise $O(n^2)$.

□

If we allow Q to have holes, we cannot rate the costs for the area of a pocket in advance.

11.4 Structural Results for Shortcut Hulls with Holes

In this section, we present structural results for SHORTCUTHULL, which we utilize for an algorithm in Section 11.5. We allow the shortcut hull to have holes.

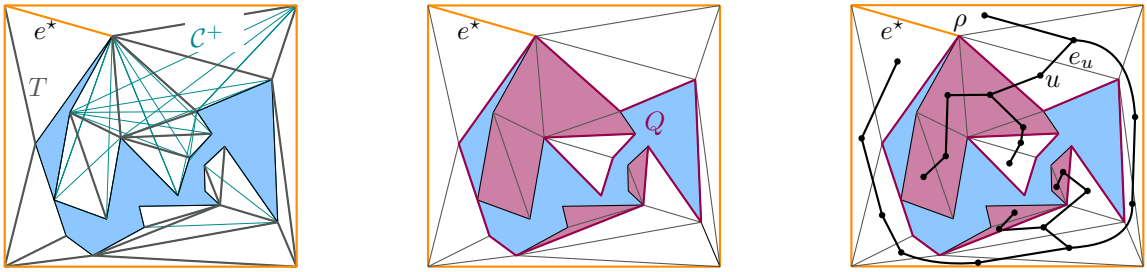
11.4.1 Basic Concepts

Let P be a weakly-simple polygon. Let $p_1, \ldots, p_n$ be the vertices of P; see Figure 11.7a. We assume that the topmost vertex of P is uniquely defined; we always can rotate P such that this is the case. We denote that vertex by p_1 and assume that P is clockwise oriented. Further, let $\mathscr{C}$ be a set of shortcuts of P and $\lambda \in [0,1]$; see Figure 11.4a. Due to Observation 1, every $\mathscr{C}$-hull of P has p_1 as a vertex.

First, we introduce concepts for the description of the structural results and the algorithm. Let $\mathscr{B}$ be an axis-aligned rectangle such that it is slightly larger than the bounding box of P; see Figure 11.7a. Let $q_1, \ldots, q_4$ be the vertices of $\mathscr{B}$ in clockwise order such that q_1 is the top-left corner of $\mathscr{B}$. We require that the diagonal edges q_1q_3 and q_2q_4 intersect P, which is always possible. We call $\mathscr{B}$ a *containing box* of P. Let D be the polygonal ring $q_1q_2q_3q_4q_1p_1p_n \ldots p_1q_1$. We call D a *sliced donut* of P; see Figure 11.7b. We observe that D is the boundary of a bounded face of a planar straight-line graph. Further, we call $e^\star = p_1q_1$ the *cut edge* of D. For an edge e that lies in the interior of the face bounded by D and connects two vertices of D, let $D[e]$ be the subchain of D that starts at the start vertex of e and ends at the end vertex of e such that $e^\star$ is not part of that subchain; see Figure 11.7c. Let $D[e] + e$ be the polygonal ring that we obtain by concatenating $D[e]$ and e. Note that if $e \in \mathscr{C}$ then $D[e] = P[e]$. We call the area enclosed by $D[e] + e$ the *pocket* of e. In particular, we define the area enclosed by D to be the pocket of $e^\star$.

Observation 2. *The edges of a $\mathscr{C}$-hull of P are contained in the sliced donut D.*

In the following, we define a set $\mathscr{C}^+$ of edges in D with $\mathscr{C} \subseteq \mathscr{C}^+$ that we use for constructing triangulations of D, which encode the shortcut hulls. Generally, a *triangulation* of a polygon H is a superset of the edges of H such that they partition the interior of H into triangles. Further, for a given set E of edges an *E-triangulation* of H is a triangulation of H that only consists of edges from E. Moreover, we say that a set E of edges is *part of* a triangulation T if E is a subset of the edges of T. Note that the edges of H are part of any E-triangulation of H. Conversely, we also say that T *contains* E if E is part of T.



(a) $\mathscr{C}^+$-triangulation T (b) the $\mathscr{C}^+$-hull Q of P is part of T and labels of triangles of T (c) dual graph $G^\star$

Figure 11.8: $\mathscr{C}^+$-triangulation T. (a) All depicted edges belong to $\mathscr{C}^+$, while only green edges do not belong to T. (b) The red triangles are active, while all other triangles are inactive. (c) The dual graph $G^\star$ of T forms a tree with root ρ.

We call a set $\mathscr{C}^+$ of edges with $\mathscr{C} \subseteq \mathscr{C}^+$ an *enrichment* of the shortcuts $\mathscr{C}$ and the sliced donut D if (1) every edge of $\mathscr{C}^+$ is contained in the face bounded by D, (2) every edge of $\mathscr{C}^+$ starts and ends at vertices of D, and (3) for every set $\mathscr{C}' \subseteq \mathscr{C}$ of pair-wisely non-crossing edges the set $D \cup \mathscr{C}'$ can be augmented by adding edges from $\mathscr{C}^+$ to form a triangulation of D. First, we observe that $\mathscr{C}^+$ is well-defined as every edge in $\mathscr{C}$ satisfies the first two properties. Further, by definition for any $\mathscr{C}$-hull Q there is a $\mathscr{C}^+$-triangulation T of D that contains Q. Hence, as an intermediate step, our algorithm for computing an optimal $\mathscr{C}$-hull Q creates an enrichment of $\mathscr{C}$ and D and then constructs a $\mathscr{C}^+$-triangulation that contains Q. In Section 11.4.2 we discuss the structural correspondences between $\mathscr{C}^+$-triangulations of D and (optimal) $\mathscr{C}$-hulls. In Section 11.4.3 we then show how to construct $\mathscr{C}^+$. For example, a simple approach would be to define an enrichment of $\mathscr{C}$ by including all possible shortcuts in D. We observe that any enrichment $\mathscr{C}^+$ of $\mathscr{C}$ has $O(n^2)$ edges. In general, the size of $\mathscr{C}^+$ can be described by the spatial complexity of $\mathscr{C}$, which impacts the running time of our algorithm (Section 11.5).

11.4.2 From $\mathscr{C}^+$-Triangulations to $\mathscr{C}$-Hulls

In this section, we assume that we are given an enrichment $\mathscr{C}^+$ for the set of shortcuts $\mathscr{C}$ and a sliced donut D. Let T be a $\mathscr{C}^+$-triangulation of D; see Figure 11.8.

Observation 3. *For each enrichment $\mathscr{C}^+$ of $\mathscr{C}$ and each $\mathscr{C}$-hull Q there exists a $\mathscr{C}^+$-triangulation T of the sliced donut D such that Q is part of T.*

Let T be a $\mathscr{C}^+$-triangulation of D such that the $\mathscr{C}$-hull Q is part of T; see Figure 11.8a. We can partition the set of triangles of T in those that are contained in the interior of Q and those that are contained in the exterior of Q. We call the former ones *active* and the latter ones *inactive*; see Figure 11.8b. Further, we call an edge e of T a *separator* if (1) it is a part of P and adjacent to an inactive triangle, or (2) it is adjacent to both an active and an inactive triangle. Conversely, let $\ell\colon T \to \{0,1\}$ be a labeling of T that assigns to each triangle Δ of T whether it is active ($\ell(\Delta) = 1$) or inactive ($\ell(\Delta) = 0$). We call the pair $\mathbf{T} = (T, \ell)$ a *labeled* $\mathscr{C}^+$-triangulation. From Observation 3 we obtain the next observation.

Observation 4. *For each enrichment $\mathscr{C}^+$ of $\mathscr{C}$ and each $\mathscr{C}$-hull Q there exists a labeled $\mathscr{C}^+$-triangulation such that its separators stem from $\mathscr{C}$ and form Q.*

Let $\mathbf{T} = (T, \ell)$ be a labeled $\mathscr{C}^+$-triangulation of the interior of a polygon H. We denote the set of separators of $\mathbf{T}$ by $S_{\mathbf{T}}$. We define

$$c_{1\mathrm{d}}(S_{\mathbf{T}}) = \sum_{e \in S_{\mathbf{T}}} c_{1\mathrm{d}}(e) \text{ and } c_{2\mathrm{d}}(T) = \sum_{\substack{\Delta \in T, \\ \ell(\Delta)=1}} c_{2\mathrm{d}}(\Delta),$$

where $c_{1\mathrm{d}}(e)$ denotes the length of e and $c_{2\mathrm{d}}(\Delta)$ denotes the area of Δ. The *costs* of $\mathbf{T}$ are then defined as

$$c(\mathbf{T}) = \lambda \cdot c_{1\mathrm{d}}(S_{\mathbf{T}}) + (1-\lambda) \cdot c_{2\mathrm{d}}(T).$$

For any $e \in \mathscr{C}^+ \setminus \mathscr{C}$ we define $c_{1\mathrm{d}}(e) = \infty$. Thus, we have $c(\mathbf{T}) < \infty$ if and only if $S_{\mathbf{T}} \subseteq \mathscr{C}$. We call a labeled $\mathscr{C}^+$-triangulation $\mathbf{T}$ of H *optimal* if there is no other labeled $\mathscr{C}^+$-triangulation $\mathbf{T}'$ of H with $c(\mathbf{T}') < c(\mathbf{T})$.

Next, we show that a labeled $\mathscr{C}^+$-triangulation $\mathbf{T} = (T, \ell)$ that is optimal can be recursively constructed based on optimal sub-triangulations. Let $G^\star$ be the dual graph of T, i.e., for each triangle, $G^\star$ has a node and two nodes are adjacent iff the corresponding triangles are adjacent in T; see Figure 11.8c.

Lemma 1. *The dual graph $G^\star$ of a $\mathscr{C}^+$-triangulation T of D is a binary tree.*

Proof. As each edge of T starts and ends at the boundary of D, each edge of T splits D into two disjoint regions. Hence, $G^\star$ is a tree. Further, since each node of $G^\star$ corresponds to a triangle of T, each node of $G^\star$ has at most two child nodes. □

We call $G^\star$ a *decomposition tree* of D. Let ρ be the node of $G^\star$ that corresponds to the triangle of T that is adjacent to the cut edge $e^\star$ of D; as $e^\star$ is a boundary edge of D, this triangle is uniquely defined. We assume that ρ is the root of $G^\star$; see Figure 11.8c. Let $G^\star_u$ be an arbitrary sub-tree of $G^\star$ that is rooted at a node u of $G^\star$. Further, let e_u be the edge of the triangle Δ_u of u that is not adjacent to the triangles of the child nodes of u; we call e_u the *base edge* of Δ_u. The triangles of the nodes of $G^\star_u$ form a $\mathscr{C}^+$-triangulation T_u of the pocket $A_u = D[e_u] + e_u$ of e_u. Thus, $G^\star_u$ is a decomposition tree of A_u. A labeled $\mathscr{C}^+$-sub-triangulation $\mathbf{T}_u = (T_u, \ell_u)$ consists of the $\mathscr{C}^+$-triangulation T_u of A_u with $T_u \subseteq T$ and the labeling ℓ_u with $\ell_u(\Delta) = \ell(\Delta)$ for every $\Delta \in T_u$.

Lemma 2. *Let $\mathbf{T} = (T, \ell)$ be a labeled $\mathscr{C}^+$-triangulation of D that is optimal. Let $\mathbf{T}_u = (T_u, \ell_u)$ be the labeled $\mathscr{C}^+$-sub-triangulation of $\mathbf{T}$ rooted at the node u and let $\mathbf{T}'_u = (T'_u, \ell'_u)$ be an arbitrary labeled $\mathscr{C}^+$-triangulation of the same region. We denote the triangles of $\mathbf{T}_u$ and $\mathbf{T}'_u$ adjacent to e_u by Δ_u and Δ'_u, respectively. If Δ_u and Δ'_u have the same labels, i.e., $\ell_u(\Delta_u) = \ell'_u(\Delta'_u)$, then $c(\mathbf{T}_u) \leq c(\mathbf{T}'_u)$.*

Proof. For the proof, we use a simple exchange argument. Assume that there is a labeled $\mathscr{C}^+$-triangulation $\mathbf{T}'_u$ of the pocket $D[e_u] + e_u$ with $\ell_u(\Delta_u) = \ell'_u(\Delta'_u)$ and $c(\mathbf{T}'_u) < c(\mathbf{T}_u)$. As both $\mathbf{T}_u$ and $\mathbf{T}'_u$ are triangulations of the pocket $D[e_u] + e_u$, we can replace the triangles of T_u with the triangles of T'_u in T obtaining a new triangulation $\overline{T}$ of D. Further, we define a new labeling $\overline{\ell}$ such that $\overline{\ell}(\Delta) = \ell(\Delta)$ for every $\Delta \in T \setminus T_u$ and $\overline{\ell}(\Delta) = \ell'_u(\Delta)$ for every $\Delta \in T'_u$. Let $\overline{\mathbf{T}} = (\overline{T}, \overline{\ell})$ be the corresponding labeled $\mathscr{C}^+$-triangulation of D. The following calculation shows $c(\overline{\mathbf{T}}) < c(\mathbf{T})$, which contradicts the optimality of $\mathbf{T}$.

$$
\begin{aligned}
c(\overline{\mathbf{T}}) &= \lambda \cdot \big(c_{1\mathrm{d}}(S_{\overline{\mathbf{T}}} \setminus S_{\mathbf{T}'_u}) + c_{1\mathrm{d}}(S_{\mathbf{T}'_u})\big) + \\
&\quad (1-\lambda) \cdot \big(c_{2\mathrm{d}}(\overline{T} \setminus T'_u) + c_{2\mathrm{d}}(T'_u)\big) \\
&= \lambda \cdot c_{1\mathrm{d}}(S_{\mathbf{T}} \setminus S_{\mathbf{T}'_u}) + (1-\lambda) \cdot c_{2\mathrm{d}}(\overline{T} \setminus T'_u) + \\
&\quad \lambda \cdot c_{1\mathrm{d}}(S_{\mathbf{T}'_u}) + (1-\lambda) \cdot c_{2\mathrm{d}}(T'_u) \\
&= \lambda \cdot c_{1\mathrm{d}}(S_{\mathbf{T}} \setminus S_{\mathbf{T}_u}) + (1-\lambda) \cdot c_{2\mathrm{d}}(\overline{T} \setminus T'_u) + c(\mathbf{T}'_u) \\
&< \lambda \cdot c_{1\mathrm{d}}(S_{\mathbf{T}} \setminus S_{\mathbf{T}_u}) + (1-\lambda) \cdot c_{2\mathrm{d}}(\overline{T} \setminus T'_u) + c(\mathbf{T}_u) = c(\mathbf{T})
\end{aligned}
$$

Altogether, we obtain the statement of the lemma. □

We use Lemma 2 for a dynamic programming approach that yields a labeled $\mathscr{C}^+$-triangulation $\mathbf{T}$ of D that is optimal. In the following, we show that there exists such a triangulation for a given $\mathscr{C}^+$ and D.

Lemma 3. *Let $\mathscr{C}^+$ be an enrichment of $\mathscr{C}$ and D a sliced donut of P. There exists a labeled $\mathscr{C}^+$-triangulation $\mathbf{T}$ of D that is optimal and has cost $c(\mathbf{T}) < \infty$. The separators of $\mathbf{T}$ form an optimal $\mathscr{C}$-hull of P.*

Proof. We show the following two claims, which prove the lemma. (1) For every $\mathscr{C}$-hull Q of P there is a labeled $\mathscr{C}^+$-triangulation $\mathbf{T}$ of D such that the separators of $\mathbf{T}$ form Q and $c(\mathbf{T}) = c(Q)$. (2) For every labeled $\mathscr{C}^+$-triangulation $\mathbf{T}$ of D with $c(\mathbf{T}) < \infty$ the separators of $\mathbf{T}$ form a $\mathscr{C}$-hull Q with $c(\mathbf{T}) = c(Q)$. *Claim 1.* Let Q be a $\mathscr{C}$-hull of P. By the definition of $\mathscr{C}^+$, there is a $\mathscr{C}^+$-triangulation T of D such that Q is part of T. We define the labeling ℓ such that $\ell(\Delta) = 1$ for every triangle $\Delta \in T$ that is contained in the interior of Q and $\ell(\Delta) = 0$ for every other triangle $\Delta \in T$. Hence, the separators of the labeled $\mathscr{C}^+$-triangulation $\mathbf{T} = (T, \ell)$ are the edges of Q. Further, by the construction of $\mathbf{T}$ we have $c(\mathbf{T}) = c(Q)$. This proves Claim 1.

Claim 2. Let $\mathbf{T} = (T, \ell)$ be a $\mathscr{C}^+$-triangulation of D with $c(\mathbf{T}) < \infty$ and let $S_{\mathbf{T}}$ be the separators of $\mathbf{T}$. By the definition of the costs of $\mathbf{T}$, we have $S_{\mathbf{T}} \subseteq \mathscr{C}$. Moreover, as T is a triangulation, the edges in $S_{\mathbf{T}}$ do not cross each other. We show that the edges in $S_{\mathbf{T}}$ form a $\mathscr{C}$-hull Q with $c(Q) = c(\mathbf{T})$. Let $G^\star$ be the dual graph of T. As the diagonal edges of the containing box $\mathscr{B}$ intersect P, each triangle of T that is incident to one of the vertices of $\mathscr{B}$ is also incident to a vertex of P; see Figure 11.9a. The vertices of the triangles incident to the vertices of $\mathscr{B}$ form a path $v_1, \ldots, v_k$ in $G^\star$ such v_1 is the root of $G^\star$ and v_k is a leaf. We denote the triangles represented by this path by $\Delta_1, \ldots, \Delta_k$, respectively.

Let $p_1, \ldots, p_l$ be the vertices of P in the order as they are incident to the triangles $\Delta_1, \ldots, \Delta_k$ in clockwise order; see Figure 11.9b. We define $p_{l+1} = p_1$. The vertices $p_1, \ldots, p_l$ form a weakly-simple polygon Q' that contains P; if P crossed Q', this would contradict that the vertices are incident to the disjoint triangles $\Delta_1, \ldots, \Delta_k$. We observe that Q' is a $\mathscr{C}^+$-hull of P without holes. Let $T' \subseteq T$ be the set of triangles that are contained in Q' and let E' be

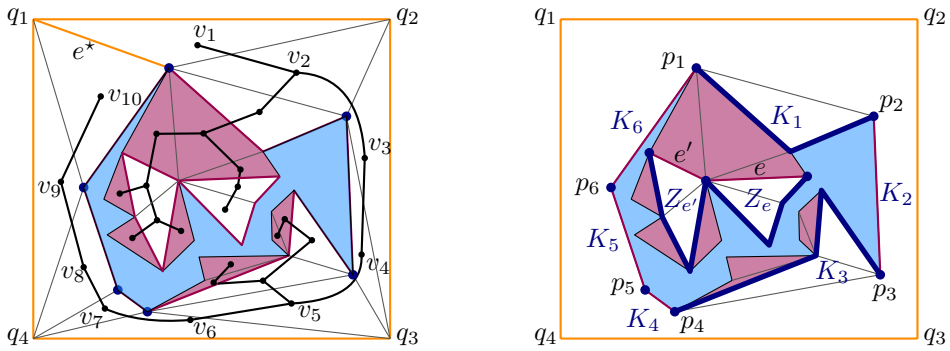


Figure 11.9: Proof of Lemma 3. (a) The triangles incident to the vertices q_1, q_2, q_3 and q_4 form a path in the dual graph of the labeled triangulation $\mathbf{T}$. (b) The vertices $p_1, \ldots, p_5$ form a $\mathscr{C}^+$-hull of P containing all active triangles (red) of $\mathbf{T}$.

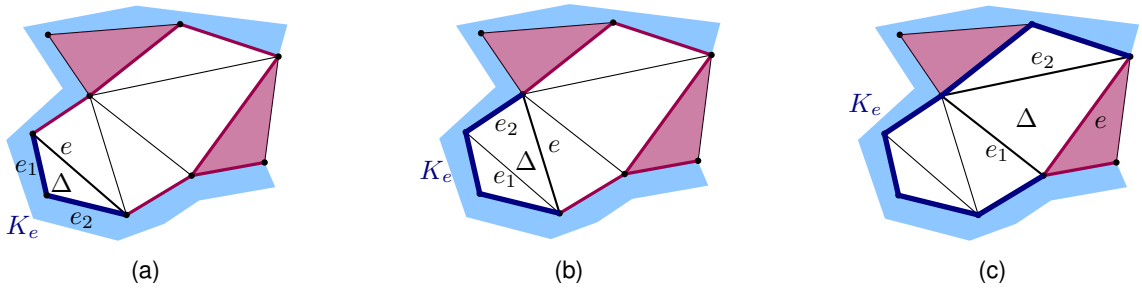


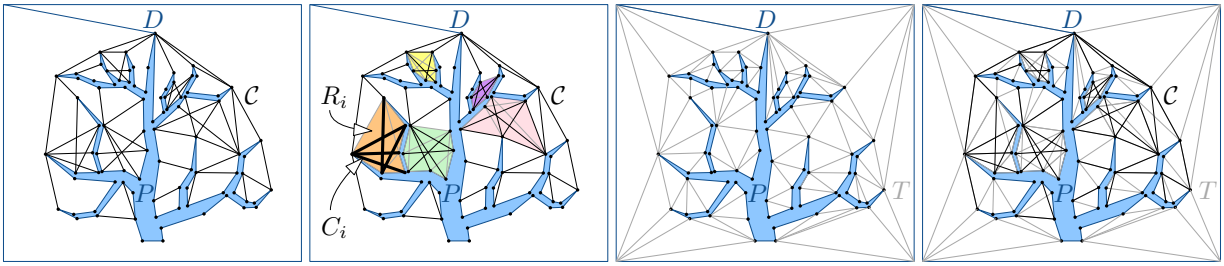
Figure 11.10: Inductive construction of the boundary path K_e of an edge e that is a base edge of an inactive triangle Δ. (a) Base case. (b) e_1 is a base edge of an inactive triangle, and e_2 is a separator. (c) Both e_1 and e_2 are base edges of inactive triangles.

the edges of these triangles. We first show that for each edge $e \in E'$ that is a base edge of an inactive triangle in $\mathbf{T}$ there is a path K_e in the pocket of e such that (1) K_e only consists of edges from $S_{\mathbf{T}}$, (2) K_e connects the endpoints of e, and (3) the polygon $K_e + e$ only contains inactive triangles of $\mathbf{T}$. We call K_e the *boundary path* of e; see Figure 11.10. Later, we use these boundary paths to assemble Q.

Let Δ be the inactive triangle of which e is the base edge and let e_1 and e_2 be the other two edges of Δ. We do an induction over the number of triangles of $\mathbf{T}$ that are contained in the pocket of e. If the pocket of e only contains Δ, both edges e_1 and e_2 are edges of P; see Figure 11.10a. Hence, by definition they are separators. We define K_e as the path $e_1 + e_2$, which satisfies the three requirements above. So assume that the pocket of e contains more than one triangle; see Figure 11.10b–c. If e_1 is not a separator, then it is the base edge of an inactive triangle. Hence, by induction, there is a path K_{e_1} that satisfies the requirements above. If e_1 is a separator, we define $K_{e_1} = e_1$. In the same way, we define a path K_{e_2} for the edge e_2. The concatenation $K_{e_1} + K_{e_2}$ forms a path that satisfies the requirements above, which proves the existence of the boundary path for an edge $e \in E'$.

We now describe the construction of the boundary of Q. For a pair p_i, p_{i+1} with $1 \leq i < l$, the adjacent triangle incident to one of the vertices of $\mathscr{B}$ is inactive. Let $K_i = p_i p_{i+1}$ if $p_i p_{i+1}$ is a separator. Otherwise, $p_i p_{i+1}$ is the base edge of an inactive triangle in $\mathbf{T}$. Thus, it has a boundary path $K_{p_i p_{i+1}}$ and we define K_i as K_e. The concatenation $K_1 + \cdots + K_l$ forms the boundary B of a weakly-simple polygon Q that encloses P; see Figure 11.9b. By construction, it consists of edges from $\mathscr{C}$.

Finally, we show how to construct the holes of Q. Let $e \in S_{\mathbf{T}}$ be a separator that is contained in the interior of B and that is a base edge of an inactive triangle; see e and e' in Figure 11.9b. The polygon Z_e that consists of e and the boundary path K_e only contains inactive triangles of $\mathbf{T}$ and is entirely contained in B. Further, for any pair e and e' of such separators in the interior of B the interiors of the polygons Z_e and $Z_{e'}$ are disjoint. Hence, we set these polygons to be the holes of Q. Thus, we obtain a $\mathscr{C}$-hull Q of P with holes such that the inactive triangles of $\mathbf{T}$ lie in the exterior of Q, while all active triangles lie in the interior of Q. This implies that $c(Q) = c(\mathbf{T})$, which concludes the proof of Claim 2. □



(a) P, $\mathscr{C}$, and D (b) crossing components (c) edges E_T (d) $\mathscr{C}^+$

Figure 11.11: Obtaining the enrichment $\mathscr{C}^+$ from $\mathscr{C}$.

11.4.3 From $\mathscr{C}$ to $\mathscr{C}^+$

The performance of our algorithm solving ShortcutHull relies on the considered enrichment $\mathscr{C}^+$. For an edge $e \in \mathscr{C}^+$, let δ_e be the number of triangles that can be formed by e and two other edges from $\mathscr{C}^+$, and let $\delta(\mathscr{C}^+)$ be the maximum δ_e over all edges e in $\mathscr{C}^+$. In Section 11.5 we show that the problem can be solved in $O(|\mathscr{C}^+| \cdot \delta(\mathscr{C}^+))$ time. In this part, we shortly discuss how to choose $\mathscr{C}^+$.

A simple choice for $\mathscr{C}^+$ is the set of all edges that lie in D and connect vertices of D. It is an enrichment of $\mathscr{C}$ as it contains any choice of $\mathscr{C}$ and any triangulation of D that is based on the vertices of D is a subset of $\mathscr{C}^+$.

Observation 5. *There exists an enrichment $\mathscr{C}^+$ of $\mathscr{C}$ with $|\mathscr{C}^+| \in O(n^2)$ and $\delta(\mathscr{C}^+) \in O(n)$.*

If $\mathscr{C}$ has no crossings, we can do much better. We first observe that the edges of any triangulation T of the sliced donut D are an enrichment of $\mathscr{C}$ and D if $\mathscr{C}$ is a subset of these edges. Hence, we can define an enrichment as the set of edges of a triangulation T of D such that the edges of $\mathscr{C}$ are part of T; for this purpose we can for example utilize constrained Delaunay triangulations, but also other triangulation techniques are possible.

Observation 6. *If the edges in $\mathscr{C}$ do not cross, $\mathscr{C}$ has an enrichment $\mathscr{C}^+$ with $|\mathscr{C}^+| \in O(n)$ and $\delta(\mathscr{C}^+) \in O(1)$.*

In the following, we generalize both constructions of $\mathscr{C}^+$ and relate $|\mathscr{C}^+|$ and $\delta(\mathscr{C}^+)$ to the number n of vertices of P, the number of crossing components h, and the spatial complexity χ of $\mathscr{C}$.

Theorem 2. *There is an enrichment $\mathscr{C}^+$ of $\mathscr{C}$ with $|\mathscr{C}^+| \in O(h\chi^2 + n)$ and $\delta(\mathscr{C}^+) \in O(\chi)$, where h is the number of crossing components, χ is the spatial complexity.*

Proof. Let $\mathscr{C}_1, \ldots, \mathscr{C}_h$ be subsets of $\mathscr{C}$ such that two edges $e \in \mathscr{C}_i$ and $e' \in \mathscr{C}_j$ with $1 \leq i, j \leq h$ cross each other if and only if $i = j$; see Figure 11.11. We call $\mathscr{C}_i$ a *crossing component* of $\mathscr{C}$. Let R_i be the polygon in D with the fewest edges, that is defined by vertices of P and contains C_i. We call R_i the *region* of C_i. Observe that the regions $R_1, \ldots, R_h$ of crossing components induce a partition $\mathbf{R}$ of D that consists of $R_1, \ldots, R_h$ and regions $R'_1, \ldots, R'_g$ defining $D \setminus \bigcup_{i=1}^{h} R_i$. For every region R'_i let T'_i be an arbitrary triangulation.

Let $\mathscr{C}^+$ be the set of edges that contains (i) all edges of $\mathscr{C}$, (ii) for each $1 \leq i \leq g$ the edges of T'_i, and (iii) for each $1 \leq i \leq h$ the set of all possible shortcuts of the region R_i such that these start and end at vertices of R_i and are contained in D. The set $\mathscr{C}^+$ has size $O(h\chi^2 + n)$ as each region R_i has at most χ vertices and the triangulations of $R'_1, \ldots, R'_g$ have $O(n)$ edges in total.

We show that $\mathscr{C}^+$ is an enrichment, by proving that for each set $\mathscr{C}' \subseteq \mathscr{C}$ of pair-wisely non-crossing edges there is a $\mathscr{C}^+$-triangulation T of D such that $\mathscr{C}'$ is part of T. Since an edge $e \in \mathscr{C}^+$ cannot cross the boundary of two regions $R, R' \in \mathbf{R}$, we can compose T by triangulations of the regions in $\mathbf{R}$ as follows.

Let E be the edges of $\mathscr{C}'$ that are contained in the region $R \in \mathbf{R}$. If R is a region of a crossing component, $\mathscr{C}^+$ contains all shortcuts in this region. Since the edges of E are crossing-free, there exists a $\mathscr{C}^+$-triangulation of R that is constrained to E. Thus, the edges of E are part of a $\mathscr{C}^+$-triangulation of R. If R is not a region of a crossing component, the edges of $\mathscr{C}^+$ that lie in R form by the construction of $\mathscr{C}^+$ a triangulation of R. By joining those triangulations of the regions in $\mathbf{R}$, we obtain a $\mathscr{C}^+$-triangulation of D such that $\mathscr{C}'$ is part of it. □

11.5 Computing Optimal Shortcut Hulls with Holes

The core of our algorithm is a dynamic programming approach that recursively builds the decomposition tree of T as well as the labeling ℓ using the sliced donut D of the input polygon P and the input set of shortcuts $\mathscr{C}$ as guidance utilizing Lemma 2. The algorithm consists of the following steps.

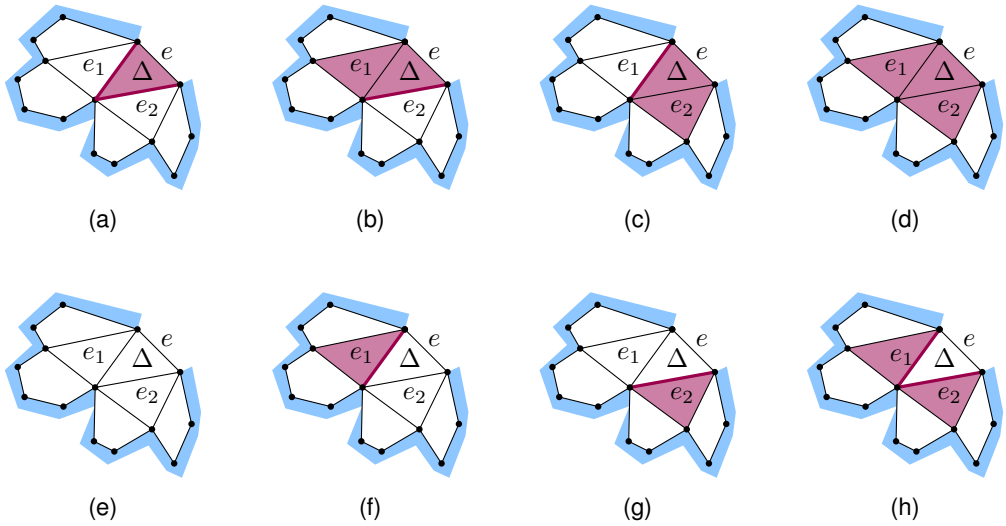


Figure 11.12: The possible cases for the (a)–(d) active (red) and (e)–(h) inactive cost of a triangle Δ.

1. Create a containing box $\mathscr{B}$ and the sliced donut D of P and $\mathscr{B}$. Let $e^\star$ be the cut edge of D.
2. Create an enrichment $\mathscr{C}^+$ of $\mathscr{C}$ and D.
3. Create the directed graph $G_{\mathscr{C}^+}$ induced by the edges in $\mathscr{C}^+$, i.e., there is a one-to-one correspondence between the edges of $\mathscr{C}^+$ and the edges of $G_{\mathscr{C}^+}$. Let $\mathscr{T}$ be the set of triangles in $G_{\mathscr{C}^+}$.
4. Determine for each edge e of $G_{\mathscr{C}^+}$ the set $\mathscr{T}_e \subseteq \mathscr{T}$ of all triangles (e,e_1,e_2) in $G_{\mathscr{C}^+}$ such that e_1 and e_2 lie in the pocket of e.
5. Create two tables A and I such that they have an entry for each edge e of $G_{\mathscr{C}^+}$.
   - $\mathrm{A}[e]$: minimal cost of a labeled $\mathscr{C}^+$-triangulation $\mathbf{T}$ of the pocket $D[e]$+e such that the triangle adjacent to e is active.
   - $\mathrm{I}[e]$: minimal cost of a labeled $\mathscr{C}^+$-triangulation $\mathbf{T}$ of the pocket $D[e]$+e such that the triangle adjacent to e is inactive.
6. Starting at $\mathrm{I}[e^\star]$ apply a backtracking procedure to create a $\mathscr{C}^+$-triangulation $\mathbf{T}$ of D that is optimal. Return $\mathbf{T}$ and the corresponding optimal $\mathscr{C}$-hull Q of $\mathbf{T}$ (see proof of Lemma 3 for construction of Q).

We now explain Step 5 and Step 6 in greater detail.

Step 5 Let e be the currently considered edge of $G_{\mathscr{C}^+}$. For a triangle $\Delta = (e,e_1,e_2) \in \mathscr{T}_e$ of e we define its *active cost* x_Δ as

$$x_\Delta = \sum_{i\in\{1,2\}} \min\{\mathrm{A}[e_i], I[e_i] + \lambda \cdot c_{1\mathrm{d}}(e_i)\}.$$

Hence, x_Δ is the cost of a labeled $\mathscr{C}^+$-triangulation $\mathbf{T}_e$ of the pocket $D[e]+e$ such that Δ is active and the sub-triangulations of $\mathbf{T}_e$ restricted to the pockets $D[e_1]+e_1$ and $D[e_2]+e_2$ are optimal, respectively; see Figure 11.12 for the four possible cases.

$$\mathrm{A}[e] = \begin{cases} \infty & e \notin \mathscr{C} \\ (1-\lambda)\cdot c_{2\mathrm{d}}(e) & e \in \mathscr{C},\ \mathscr{T}_e = \emptyset, \\ \min\{x_\Delta \mid \Delta \in \mathscr{T}_e\} + (1-\lambda)\cdot c_{2\mathrm{d}}(e) & e \in \mathscr{C},\ \mathscr{T}_e \neq \emptyset. \end{cases}$$

Analogously, we define for Δ its *inactive cost* y_Δ as

$$y_\Delta = \sum_{i\in\{1,2\}} \min\{\mathrm{A}[e_i] + \lambda \cdot c_{1\mathrm{d}}(e_i), I[e_i]\}.$$

Hence, y_Δ is the cost of a labeled $\mathscr{C}^+$-triangulation $\mathbf{T}_e$ of the pocket $D[e]+e$ such that Δ is inactive and the sub-triangulations of $\mathbf{T}_e$ restricted to the pockets $D[e_1]+e_1$ and $D[e_2]+e_2$ are optimal, respectively. We compute the entry $\mathrm{I}[e]$ as follows.

$$\mathrm{I}[e] = \begin{cases} \infty & e \in \mathscr{C} \text{ and } \mathscr{T}_e = \emptyset, \\ \min\{y_\Delta \mid \Delta \in \mathscr{T}_e\} & \text{otherwise.} \end{cases}$$

By the definition of the tables A and I and Lemma 2, it directly follows that $\mathrm{I}[e^\star]$ is the cost of a labeled $\mathscr{C}^+$-triangulation of D that is optimal. In particular, by Lemma 3 the entry $\mathrm{I}[e^\star]$ is the cost of an optimal $\mathscr{C}$-hull. Note that we compute the table entries of A and I in increasing order of the areas of the edges' pockets. With this, we can ensure that $A[e_1]$,$A[e_2]$,$I[e_1]$, and $I[e_2]$ are already computed when considering e, as the areas of the pockets of e_1 and e_2 are strictly smaller than the area of the pocket of e.

Step 6 When filling both tables, we further store for each entry $\mathrm{A}[e]$ the triangle $(e,e_1,e_2) \in \mathscr{T}_e$ with minimum active cost. In particular, for the edge e_i (with $i \in \{1,2\}$) we store a pointer to the entry $\mathrm{A}[e_i]$ if $\mathrm{A}[e_i] < \mathrm{I}[e_i] + \lambda \cdot c_{1\mathrm{d}}(e_i)$ and a pointer to the entry $\mathrm{I}[e_i]$ otherwise. Similarly, we store for each entry $\mathrm{I}[e]$ the triangle $(e,e_1,e_2) \in T_e$ with minimum inactive cost. In particular, for the edge e_i (with $i \in \{1,2\}$) we store a pointer to the entry $\mathrm{I}[e_i]$ if $\mathrm{I}[e_i] < \mathrm{A}[e_i] + \lambda \cdot c_{1\mathrm{d}}(e_i)$ and a pointer to the entry $\mathrm{A}[e_i]$ otherwise. Starting at the entry $\mathrm{I}[e^\star]$, we follow the pointers and collect for each encountered entry its triangle —if such a triangle does not exist, we terminate the traversal. If the entry belongs to A we label Δ active and if it belongs to I, we label Δ inactive. The set T of collected triangles forms a labeled $\mathscr{C}^+$-triangulation $\mathbf{T}$ of D that is optimal. By Lemma 3 the separators of $\mathbf{T}$ form an optimal $\mathscr{C}$-hull.

Running Time For the running time analysis, we assume that the crossing components and their boundaries are pre-computed. The first step of our algorithmclearly runs in $O(n)$ time. By Theorem 2 there is an enrichment $\mathscr{C}^+$ of $\mathscr{C}$ and D that has size $O(h\chi^2+n)$. It can be easily constructed in $O(h\chi^3+\chi n)$ time, which dominates the running times of Step 2. For each edge e of $G_{\mathscr{C}^+}$ the set $\mathscr{T}_e$ contains $\delta(\mathscr{C}^+)$ triangles. Hence, the size of $\mathscr{T}$ is in $O(|\mathscr{C}^+| \cdot \delta(\mathscr{C}^+))$ which dominates the running times of Step 3, Step 4, and Step 5. Hence, by Theorem 2 we obtain $O(h\chi^3+\chi n)$ running time. The backtracking takes linear time as it only traverses the decomposition tree that is implicitly given by the pointers between the table entries.

Theorem 3. *Given crossing components with their boundaries for the shortcuts, the problem* SHORTCUTHULL *can be solved in $O(h\chi^3+\chi n)$ time. In particular, it is solvable in $O(n^3)$ time in general and in $O(n)$ time if the edges in $\mathscr{C}$ do not cross.*

11.6 Restricted Shortcut Hulls

In this section, we discuss two variants of SHORTCUTHULL in which we restrict the number of edges and bends of the computed shortcut hull. These restrictions are particularly interesting for the simplification of geometric objects as they additionally allow us to easily control the complexity of the simplification.

11.6.1 Restricted $\mathscr{C}$-Hull: Number of Edges

We show how to find a $\mathscr{C}$-hull Q that balances its enclosed area and perimeter under the restriction that it consists of at most k edges. We say that Q is optimal *restricted to at most k edges*, if there is no other $\mathscr{C}$-hull Q' with at most k edges and $c(Q') < c(Q)$.

k-EDGESHORTCUTHULL.

input: A weakly-simple region P with n vertices,
A set $\mathscr{C}$ of shortcuts of P, $\lambda \in [0,1]$, and $k \in \mathbb{N}$
output: An optimal $\mathscr{C}$-hull Q of P (if it exists) restricted to at most k edges.

To solve k-EDGESHORTCUTHULL we adapt Step 5 of the algorithm presented in Section 11.5. We extend the tables A and I by an additional dimension of size k modeling the budget of edges that we have left for the particular instance. For a shortcut $e \in \mathscr{C}^+$ and a *budget* b we interpret the table entries as follows.

- $\mathrm{A}[e][b]$: cost of labeled $\mathscr{C}^+$-triangulation $\mathbf{T}$ of the pocket of e such that $\mathbf{T}$ is optimal, the triangle adjacent to e is active and $\mathbf{T}$ contains at most b separators.
- $\mathrm{I}[e][b]$: cost of labeled $\mathscr{C}^+$-triangulation $\mathbf{T}$ of the pocket of e such that $\mathbf{T}$ is optimal, the triangle adjacent to e is inactive and $\mathbf{T}$ contains at most b separators.

Let e be the currently considered edge of $G_{\mathscr{C}^+}$ when filling the tables. For a triangle $\Delta = (e,e_1,e_2) \in \mathscr{T}_e$ of e its active and inactive costs depend on the given budgets b_1 and b_2 with $1 \leq b_1,b_2 \leq k$ that we intend to use for the sub-instances attached to e_1 and e_2.

$$x_{\Delta,b_1,b_2} = \sum_{i\in\{1,2\}} \min\{\mathrm{A}[e_i][b_i], I[e_i][b_i-1] + \lambda \cdot c_{1\mathrm{d}}(e_i)\}$$

$$y_{\Delta,b_1,b_2} = \sum_{i\in\{1,2\}} \min\{\mathrm{A}[e_i][b_i-1] + \lambda \cdot c_{1\mathrm{d}}(e_i), I[e_i][b_i]\}$$

Hence, for the case that $e \in \mathscr{C}$ and $\mathscr{T}_e \neq \emptyset$ we define

$$\mathrm{A}[e][b] = \min\{x_{\Delta,b_1,b_2} \mid \Delta \in \mathscr{T}_e, b_1 + b_2 = b\} + (1-\lambda) \cdot c_{2\mathrm{d}}(e).$$

There are b possible choices of b_1 and b_2 that satisfy $b_1 + b_2 = b$. Thus, we can compute $A[e][b]$ in $O(b)$ time. For the remaining cases, we define

$$\mathrm{A}[e][b] = \begin{cases} \infty & e \notin \mathscr{C} \\ (1-\lambda) \cdot c_{2\mathrm{d}}(e) & e \in \mathscr{C},\ \mathscr{T}_e = \emptyset, \end{cases}$$

which can be computed in $O(1)$ time. Moreover, for the case that $e \notin \mathscr{C}$ or $\mathscr{T}_e \neq \emptyset$ we define

$$\mathrm{I}[e][b] = \min\{y_{\Delta,b_1,b_2} \mid \Delta \in \mathscr{T}_e, b_1 + b_2 = b\}.$$

For the same reasons as before we can compute $\mathrm{I}[e][b]$ in $O(1)$ time. For $e \in \mathscr{C}$ or $\mathscr{T}_e \neq \emptyset$ we define $\mathrm{I}[e][b] = \infty$. Finally, to cover border cases we set $\mathrm{A}[e][0] = \infty$ and $\mathrm{I}[e][0] = \infty$. Altogether, the entry $\mathrm{I}[e^\star][k]$ contains the cost of an optimal $\mathscr{C}$-hull that is restricted to k edges. Apart from minor changes in Step 6, the other parts of the algorithm remain unchanged.

Running time Compared to the algorithm of Section 11.5 the running time of computing a single entry increases by a factor of $O(k)$. Further, there are $O(k)$ times more entries to be computed, which yields that the running time increases by a factor of $O(k^2)$.

Theorem 4. *The problem k-EDGESHORTCUTHULL can be solved in $O(k^2(h\chi^3 + n\chi))$ time. In particular, it can be solved in $O(k^2n^3)$ time in general and in $O(k^2n)$ time if the edges in $\mathscr{C}$ do not cross.*

11.6.2 Restricted $\mathscr{C}$-Hull: Number of Bends

A slightly stronger constraint than restricting the number of edges is restricting the number of bends of a $\mathscr{C}$-hull. Formally, we call two consecutive edges of a weakly-simple polygon a *bend* if the enclosed angle is not $180°$. We say that Q is *optimal restricted to at most k bends* if there is no other $\mathscr{C}$-hull Q' with at most k bends and $c(Q') < c(Q)$.

k-BENDSHORTCUTHULL.

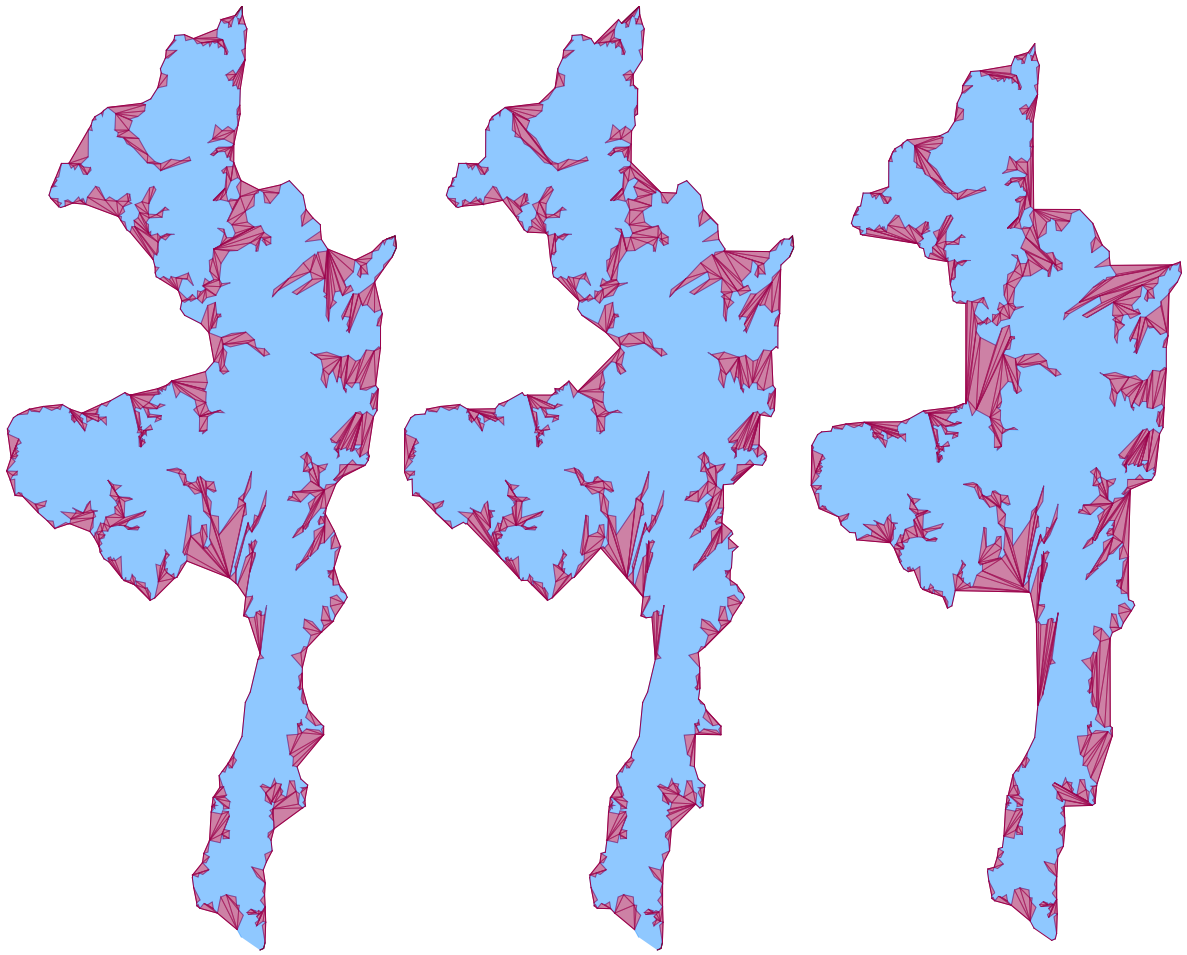
input: A weakly-simple region P and with n vertices,
A set $\mathscr{C}$ of shortcuts of P, $\lambda \in [0,1]$, and $k \in \mathbb{N}$
output: An optimal $\mathscr{C}$-hull Q of P (if it exists) that is restricted to at most k bends.

If the vertices of P are in general position, i.e., no three vertices lie on a common line, a $\mathscr{C}$-hull Q of P is optimal restricted to at most k bends if and only if it is optimal restricted to k edges. Hence, in that case, we can solve k-BENDSHORTCUTHULL using the algorithm presented in Section 11.6.1. In applications, the case that the vertices of P are not in general position occurs likely when the input polygon is, e.g., a schematic polygon or a polygon whose vertices lie on a grid. In that case, we add an edge p_1p_h to $\mathscr{C}$ for each sequence $p_1, \ldots, p_h$ of at least three vertices of P that lie on a common line; we add p_1p_h only if it lies in the exterior of P. The newly obtained set $\mathscr{C}'$ has $O(n^2)$ edges. Hence, compared to $\mathscr{C}$ it possibly has an increased spatial complexity with $\chi \in O(n)$. From Theorem 4 we obtain the next result.

Theorem 5. *The problem k-BENDSHORTCUTHULL can be solved in $O(k^2n^3)$ time.*

11.7 Experiments

We have implemented the algorithm presented in Section 11.5. For example, computing a shortcut hull for the instance shown in Figure 11.2 with 1836 vertices one run of the dynamic programming approach (Step 5) took 400ms on average using an ordinary laptop. This suggests that despite its cubic worst-case running time our algorithm is efficient enough for real-world applications. However, more experiments are needed to substantiate this finding.



(a) (b) octilinear (c) rectilinear

Figure 11.13: Simplification (a) and schematization (b)–(c) of the main island of Shetland.

Balancing the Costs of Area and Perimeter In Figure 11.1 we display a series of optimal $\mathscr{C}$-hulls[1]. We use the same polygon as input and define all possible shortcuts as admissible shortcuts while increasing the parameter λ of the cost function. To find relevant values of λ we implemented a systematic search in the range $[0,1]$. It uses the simple observation that with monotonically increasing λ the amount of area enclosed by an optimal shortcut hull increases monotonically. More in detail, we compute the optimal shortcut hull for $\lambda = 0$ and $\lambda = 1$. If the area cost c_A of these shortcut hulls differ, we recursively consider the intervals $[0,0.5]$ and $[0.5,1]$ for the choice of λ similar to a binary search. Otherwise, we stop the search.

As presented in Equation 11.1, we consider costs for the area and perimeter in SHORTCUTHULL. The second column of Figure 11.1 shows a result for a small value of λ, i.e., the costs for the area are weighted higher. As expected the resulting optimal $\mathscr{C}$-hull is rather close to the input polygon. In contrast, the last column of Figure 11.1 shows the optimal $\mathscr{C}$-hull for a larger value of λ. We particularly obtain holes that represent large areas enclosed by the polygon, while small gaps are filled.

Simplification and Schematization of Simple Polygons In the following, we discuss how our approach relates to typical measures for simplification and schematization. These are the number of edges, the number of bends [Douglas and Peucker, 1973] or the perimeter [Tufte, 1985], which are implemented by shortcut hulls; e.g., Figure 11.13a shows the simplification of the border of the main island of Shetland by a $\mathscr{C}$-hull as defined in SHORTCUTHULL. The schematization of a polygon is frequently implemented as a hard constraint with respect to a given set O of edge orientations. For schematizing a polygon with $\mathscr{C}$-hulls, we outline two possibilities: a non-strict and a strict schematization. For the non-strict schematization, we adapt the cost function of the shortcuts such that edges with an orientation similar to an orientation of O are cheap while the others are expensive; see Figure 11.13b for O consisting of horizontal, vertical, and diagonal orientations and Figure 11.13c for O

[1] Figure 11.1b: $\lambda = 0.906$; Figure 11.1c: $\lambda = 0.995$; Figure 11.1e: $\lambda = 0.914$; Figure 11.1f: $\lambda = 0.975$

consisting of the horizontal and vertical orientations. The strict schematization restricts the set $\mathscr{C}$ of shortcuts, such that each edge's orientation is from O. For example, one can define $\mathscr{C}$ based on an underlying grid that only uses orientations from O. We then need to take special care about the connectivity of $\mathscr{C}$, e.g., by also having all edges of the input polygon in $\mathscr{C}$.

11.8 Conclusion

We introduced a simplification technique for polygons that yields shortcut hulls, i.e., weakly-simple polygons that are described by shortcuts and that enclose the input polygon. In contrast to other work, we consider the admissible shortcuts as input. We introduced a cost function of a shortcut hull that is a linear combination of the covered area and the perimeter. Computing an optimal shortcut hull without holes takes $O(n^2)$ time. For the case that we permit holes, we presented an algorithm based on dynamic programming that runs in $O(n^3)$ time. If the input shortcuts do not cross, it runs in $O(n)$ time.

12 Part II: Conclusion and Outlook

Conclusion

The goal of the second part of this thesis was the formalization of optimal hulls of polygons and the development of algorithms for the computation. The hulls should aim for a simplification and/or schematization of the input polygon. We introduced a general framework of $\mathscr{C}$-hulls of polygons, i.e., hulls that contain the input polygon and consist of segments from a pre-defined set of segments $\mathscr{C}$. With an appropriate choice of $\mathscr{C}$, e.g., the direction of all segments in $\mathscr{C}$ stem from a pre-defined set, the $\mathscr{C}$-hull schematizes the input polygon. To achieve that the hull is a simplification of the input polygon, we introduce two different types of cost functions: costs that depend on the area of the hull, and costs that depend on the boundary of the hull. By balancing these costs, we achieve hulls that mimic the shape of the input while being clearer.

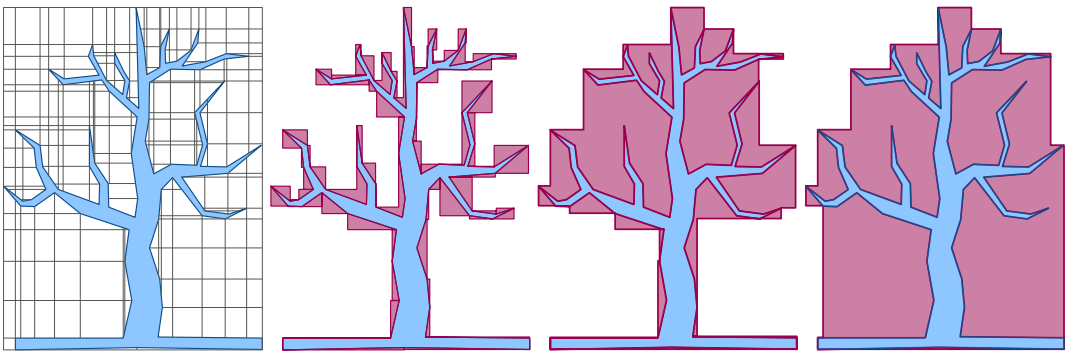
We discussed shortcut hulls in detail, where the set $\mathscr{C}$ of segments consists of segments between vertices of the input polygon. We proved that shortcut hulls can be computed efficiently with a running time of $O(n^3)$ in the worst case. We give a more detailed analysis of the running time where we also discuss better bounds for special cases depending on the input set $\mathscr{C}$, e.g., if the segments in $\mathscr{C}$ do not cross, we obtain $O(n)$ running time.

On-Going Research on Optimal Tight Hulls

In the following, we want to give an overview of a generalized variant of shortcut hulls that we started to study. We published preliminary work at the EuroCG'20 workshop [Bonerath et al., 2020a]. In the following, we give an overview of the problem formulation.

A *tight hull* of a crossing-free polygon P is another crossing-free polygon Q such that it contains P, each of its segments touches P, and its exterior is a connected region. We investigate the construction of tight $\mathscr{C}$-hulls, i.e., tight hulls that are restricted to a pre-defined grid $\mathscr{C}$ in the exterior of P. As one example among many, the grid $\mathscr{C}$ is based on horizontal and vertical rays emanating from the vertices of P; see Figure 12.1. This special case of tight $\mathscr{C}$-hulls, which we call *tight rectilinear hulls*, finds application in the schematization of polygons. We present a polynomial-time algorithm that constructs tight $\mathscr{C}$-hulls that are optimal with respect to an objective linearly balancing a cost charged for the segments (e.g., their total length) and the area enclosed by the tight hull.

We suggest an algorithm for finding a tight $\mathscr{C}$-hull Q of P based on a refinement process. We start with another tight $\mathscr{C}$-hull $\mathscr{A}$ of P that can be precomputed and successively refine $\mathscr{A}$; see Figure 12.2. A refinement step can be imagined as carving off pieces from the precomputed tight $\mathscr{C}$-hull. Using dynamic programming, we aim at computing the optimal tight $\mathscr{C}$-hull Q by exploring all refinements. More in detail, in the refinement step of $\mathscr{A}$ we partition the boundary of $\mathscr{A}$ at every shared vertex with P and refine these polylines independently; see Figure 12.3. It is particularly challenging to argue that there always exists a refinement of such a polyline that



(a) (b) (c) (d)

Figure 12.1: Tight $\mathscr{C}$-hulls. (a) Input polygon P and grid $\mathscr{C}$ based on the segments of P and vertical and horizontal rays emanating from the vertices of P. (b)–(d) Different balances between the perimeter and area of the simplification.

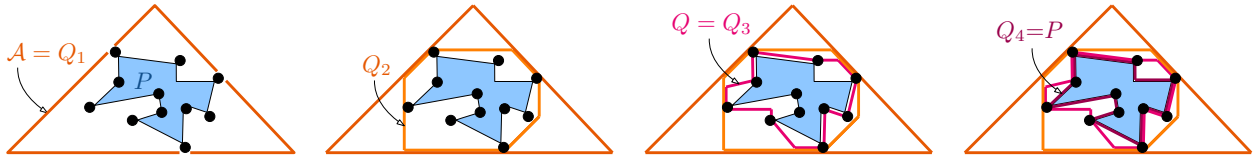


Figure 12.2: Refinement process. (a) Precomputed tight $\mathscr{C}$-hull $\mathscr{A}$ of the input polygon P. (b) Refinement Q_2 of Q_1. (c) Refinement of Q_3 of Q_2. (d) Refinement Q_4 that corresponds to the optimal tight $\mathscr{C}$-hull of P.

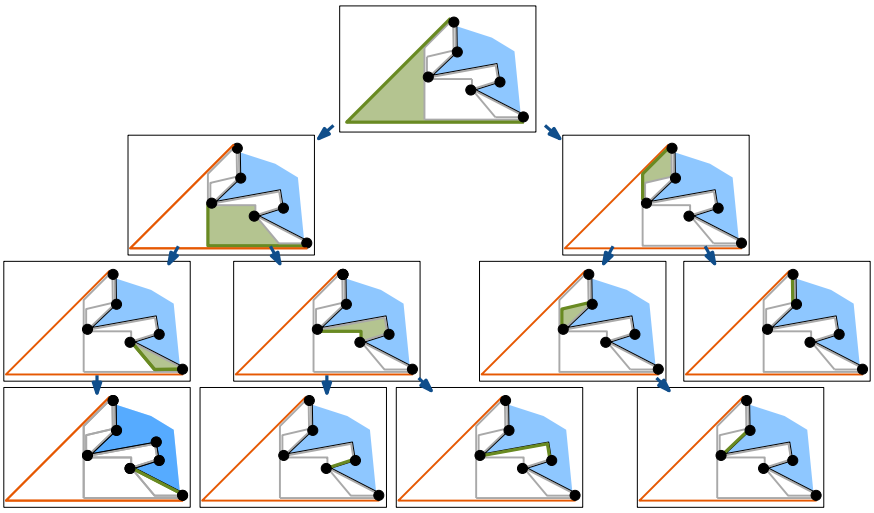


Figure 12.3: The refinement tree of a part of the polygon illustrated in Figure 12.2.

fulfills all requirements.

At EuroCG'20 workshop [Bonerath et al., 2020a], we presented results for the restricted case of tight rectilinear hulls of rectilinear input polygons. In this version, we even considered the number of bends in the cost function. With a dynamic program based on the sketched refinement process, we showed how to compute optimal tight rectilinear hulls in polynomial time ($O(n^4)$ time where n is the number of vertices of the input polygon).

Outlook

In the following, we provide ideas and open problems that are related to our presented research.

- Until now, we have not considered the **bends** of the hull in the optimization of shortcut hulls. We deem that users perceive a hull differently depending on whether it contains a number of sharp corners or not. For many applications, e.g., representation of one homogeneous object we think it to be desirable to avoid these sharp corners such that the hull looks homogeneous. We deem that both, corners with very small angles, and corners with very large angles, are sharp corners and have similar effects. One could formalize this property in the objective by introducing costs for bends. The costs can be defined as the number of bends that are considered sharp or the sum of costs that depend on the sharpness of the bend.
- In the presented work, we mentioned that one can apply shortcut hulls also for spanning trees of point sets as input. However, it was out of scope to experimentally and theoretically investigate different **spanning trees** of point sets. It is not clear whether a particular spanning tree, e.g., the minimum spanning tree, leads to an optimal solution with respect to the objective of a small covered area and small perimeter. By experimentally evaluating these hulls and comparing them to other polygonal hulls of point sets, we can get insights into the technique.
- Until now, we presented work on an optimal hull of one polygon. Besides our work, Rottmann et al. [2021] presented a formalization of an optimal hull of a set of polygons with the constraint that the hull consists of a subset of triangles of an input triangulation. We deem that investigating optimal hulls for **a set of (possibly heterogeneous) geometries**, e.g., point sets, sets of edges, sets of polygons, sets of heterogeneous geometries, or sets of three- or more-dimensional, is a promising research field. This opens up a series of new challenges. For example, for a point set as input the objective that consists of

costs for the covered area and the perimeter length is always optimal for a hull that consists only of the point set. Hence, here, one challenge is to define a reasonable mathematical model. Also, it is not clear, from a theoretical point of view whether no two hulls of the optimal set of hulls intersect, in particular, if one uses objectives that are complex.

- Our approach relies on hulls that are formed by straight-line segments which is a reasonable restriction since it is a common approach, allows a clear visualization, and comprises the information well. However, allowing **curved segments for the hull** might result in hulls that perform even better with respect to certain objectives and human perception. Also, such hulls can give insights into the objective and allow us to understand their strengths and weaknesses with respect to visualization. Hence, we suggest to investigate models of hulls with as few constraints as possible.
- Also, we suggest incorporating more **cartographic constraints and objective** for optimal hulls of geospatial objects. For example, the hull of a building footprint may not overlap with a street that lies next to it. Another example is that certain building footprints have recognizable characteristics that should be preserved in the hull. Introducing these real-world constraints and providing implementations of the algorithm can increase the applicability and visibility, e.g., in the cartographic community.
- In order to develop good models of hulls it is important to consider human perception. In the previously discussed research directions, we mentioned a series of changes to the input data, the constraints and objectives of the hull and application scenarios. We suggest evaluating these properties of hulls in **user studies**. We distinguish between (i) user studies that evaluate user performance with respect to a particular task, (ii) user studies that focus on aesthetic perception in a certain application scenario, and (iii) qualitative interviews with experts of different application domains.

  For (i), it is not straightforward how to set up the tasks. The idea of shortcut hulls is to give a clear visualization that allows a quick assessment of the rough structure. To test this, one could display a map for only a few seconds where some of the geospatial objects are replaced by their shortcut hulls. Then, the user is asked to redraw the shape of the geospatial objects in the map and one can measure the similarity to the original object.

  For (ii), it would be interesting to develop an interface where one visualizes the input geometries, an optimal hull of these geometries, and a panel where the user can interactively change the properties of the hull (e.g., weighting factors for costs of covered area and perimeter length). The task for the user is to optimize these properties such that the hull represents the data for a given application scenario best. One application scenario can be the hull of a building footprint for a certain zoom level, or the hulls of a set of animal observations that represent their geospatial habitats.

  For (iii), in our opinion, it is important to contact experts from different application fields since the requirements might differ tremendously. We suggest performing qualitative interviews for example on the aesthetic perception, the clearness of the visualization, and the perceived accuracy. Also, we propose to ask for special properties of the hulls that are motivated by the particular application field.

  With these experiments, we suggest to derive insights and maybe guidelines for desirable properties of hulls. Using this knowledge, one can iteratively improve the model and optimization of hulls.

Bibliography

M. A. Abam, M. de Berg, P. Hachenberger, and A. Zarei. Streaming algorithms for line simplification. *Discrete and Computational Geometry*, 43(3):497–515, 2010. doi: 10.1007/s00454-008-9132-4.

J. Adegeest, M.H. Overmars, and J. Snoeyink. Minimum-link c-oriented paths: Single-source queries. *Int. J. Computational Geometry and Applications*, 4(1):39–51, 1994. doi: 10.1142/S0218195994000045.

G. M. Adelson-Velskii and E. M. Landis. An algorithm for organization of information. In *Doklady Akademii Nauk*, volume 146, pages 263–266. Russian Academy of Sciences, 1962.

P. K. Agarwal, M. J. van Kreveld, and S. Suri. Label placement by maximum independent set in rectangles. *J. of Computational Geometry*, 11(3-4):209–218, 1998. doi: 10.1016/S0925-7721(98)00028-5.

C. Ahlberg and B. Shneiderman. Visual information seeking: Tight coupling of dynamic query filters with starfield displays. In *The Craft of Information Visualization*, Interactive Technologies, pages 7–13. Morgan Kaufmann, 2003. doi: 10.1016/B978-155860915-0/50004-4.

C. Ahlberg, C. Williamson, and B. Shneiderman. Dynamic queries for information exploration: An implementation and evaluation. In *Proc. Conf. Human Factors in Computing Systems (CHI'92)*, pages 619–626. ACM, 1992. doi: 10.1145/142750.143054.

T. Ai, Yaolin L., and Jun Chen. The hierarchical watershed partitioning and data simplification of river network. In *Progress in spatial data handling*, pages 617–632. Springer, 2006. doi: 10.1007/3-540-35589-8_39.

T. Ai, Q. Zhou, X. Zhang, Y. Huang, and M. Zhou. A simplification of ria coastline with geomorphologic characteristics preserved. *Marine Geodesy*, 37(2):167–186, 2014. doi: 10.1080/01490419.2014.903215.

T. Ai, S. Ke, M. Yang, and J. Li. Envelope generation and simplification of polylines using Delaunay triangulation. *Int. J. Geographic Information Science*, 31(2):297–319, 2017. doi: 10.1080/13658816.2016.1197399.

C. Alegría, D. Orden, C. Seara, and J. Urrutia. On the o-hull of planar point sets. In *Proc. of Europ. Workshop on Computational Geometry (EuroCG'14)*. Citeseer, 2014.

C. Alegría, D. Orden, C. Seara, and J. Urrutia. Efficient computation of minimum-area rectilinear convex hull under rotation and generalizations. *J. Glob. Optim.*, 79(3):687–714, 2021. doi: 10.1007/s10898-020-00953-5.

G. L. Andrienko and N. V. Andrienko. Interactive maps for visual data exploration. *Int. J. of Geographical Information Science*, 13(4): 355–374, 1999. doi: 10.1080/136588199241247.

G. L. Andrienko, N. V. Andrienko, P. Jankowski, D. A. Keim, M.-J. Kraak, A. M. MacEachren, and S. Wrobel. Geovisual analytics for spatial decision support: Setting the research agenda. *Int. J. of Geographical Information Science*, 21(8):839–857, 2007. doi: 10.1080/13658810701349011.

G. L. Andrienko, N. V. Andrienko, U. Demsar, D. Dransch, J. Dykes, S. I. Fabrikant, M. Jern, M.-J. Kraak, H. Schumann, and C. Tominski. Space, time and visual analytics. *Int. J. Geographical Information Science*, 24(10):1577–1600, 2010. doi: 10.1080/13658816.2010.508043.

G. L. Andrienko, N. V. Andrienko, C. Hurter, S. Rinzivillo, and S. Wrobel. Scalable analysis of movement data for extracting and exploring significant places. *IEEE Trans. on Visualization and Computer Graphics*, 19(7):1078–1094, 2013. doi: 10.1109/tvcg.2012.311.

G. L. Andrienko, N. V. Andrienko, H. Schumann, and C. Tominski. *Visualization of Trajectory Attributes in Space–Time Cube and Trajectory Wall*, pages 157–163. Springer, Berlin, Heidelberg, 2014. doi: 10.1007/978-3-642-32618-9_11.

N. V. Andrienko, G. L. Andrienko, and P. Gatalsky. Exploratory spatio-temporal visualization: an analytical review. *J. of Visual Languages and Computing*, 14(6):503–541, 2003. doi: 10.1016/S1045-926X(03)00046-6.

D. Attali. r-regular shape reconstruction from unorganized points. *J. of Computational Geometry*, 10(4):239–247, 1998. doi: 10.1016/S0925-7721(98)00013-3.

D. Bahrdt, M. Becher, S. Funke, F. Krumpe, A. Nusser, M. Seybold, and S. Storandt. Growing balls in $\mathbb{R}^d$. In *Proc. of 19th Workshop on Algorithm Engineering and Experiments (ALENEX'17)*, pages 247–258. SIAM, 2017. doi: 10.1137/1.9781611974768.20.

M. J. Bannister, C. DuBois, D. Eppstein, and P. Smyth. Windows into relational events: Data structures for contiguous subsequences of edges. In *Symp. on Discrete Algorithms (SODA'13)*, pages 856–864. SIAM, 2013. doi: 10.1137/1.9781611973105.61.

M. J. Bannister, W. E. Devanny, M. T. Goodrich, J. A. Simons, and L. Trott. Windows into geometric events: Data structures for time-windowed querying of temporal point sets. In *Proc. Canadian Conf. on Computational Geometry (CCCG'14)*, 2014.

R. Bar-Yehuda and B. Chazelle. Triangulating disjoint jordan chains. *Int. J. Computational Geometry and Applications*, 4(4):475–481, 1994. doi: 10.1142/S0218195994000252.

T. Barkowsky, L. J. Latecki, and K.-F. Richter. Schematizing maps: Simplification of geographic shape by discrete curve evolution. In *Spatial Cognition II, Integrating Abstract Theories, Empirical Studies, Formal Methods, and Practical Applications*, volume 1849 of *Lecture Notes in Computer Science*, pages 41–53. Springer, 2000. doi: 10.1007/3-540-45460-8_4.

L. Barth, B. Niedermann, M. Nöllenburg, and D. Strash. Temporal Map Labeling: A New Unified Framework with Experiments. In *Advances in Geographic Information Systems (ACM SIGSPATIAL'16)*, pages 23:1–23:10. ACM, 2016. doi: 10.1145/2996913.2996957.

S. E. Battersby, D. Strebe, and M. P. Finn. Shapes on a plane: evaluating the impact of projection distortion on spatial binning. *Cartography and Geographic Information Science*, 44(5):410–421, 2017. doi: 10.1080/15230406.2016.1180263.

L. Battle, R. Chang, and M. Stonebraker. Dynamic prefetching of data tiles for interactive visualization. In Fatma Özcan, Georgia Koutrika, and Sam Madden, editors, *Proc. of Int. Conf. Management of Data, SIGMOD'16*, pages 1363–1375. ACM, 2016. doi: 10.1145/2882903.2882919.

M. Baum, T. Bläsius, A. Gemsa, I. Rutter, and F. Wegner. Scalable exact visualization of isocontours in road networks via minimum-link paths. *J. Computational Geometry*, 9(1):27–73, 2018. doi: 10.20382/jocg.v9i1a2.

K. Been, E. Daiches, and C. Yap. Dynamic map labeling. *IEEE Trans. on Visualization and Computer Graphics*, 12(5):773–780, 2006. doi: 10.1109/TVCG.2006.136.

K. Been, M. Nöllenburg, S.-H. Poon, and A. Wolff. Optimizing active ranges for consistent dynamic map labeling. *J. of Computational Geometry*, 43(3):312–328, 2010. doi: 10.1016/j.comgeo.2009.03.006. Special Issue on Symp. on Computational Geometry (SoCG'08).

J. L. Bentley. Decomposable searching problems. Technical report, Carnegie-Mellon University, Pittsburg, United States, Departement of Computer Science, 1978.

F. Bernardini and C. L. Bajaj. Sampling and reconstructing manifolds using alpha-shapes. In *Proc. Canadian Conf. on Computational Geometry (CCCG'97)*, 1997.

P. Bobák, L. Cmolík, and M. Cadík. Temporally stable boundary labeling for interactive and non-interactive dynamic scenes. *Computers & Graphics*, 91:265–278, 2020. doi: 10.1016/j.cag.2020.08.005.

D. Bokal, S. Cabello, and D. Eppstein. Finding All Maximal Subsequences with Hereditary Properties. In *Proc. of Symp. on Computational Geometry (SoCG'15)*, volume 34 of *LIPIcs*, pages 240–254. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2015. doi: 10.4230/LIPICS.SOCG.2015.240.

A. Bonerath. Dynamische Aggregation von Geo-Objekten für die interaktive Exploration von Forschungsdaten, November 2018. M.Sc. thesis.

A. Bonerath, J.-H. Haunert, and B. Niedermann. Computing alpha-Shapes for Temporal Range Queries on Point Sets. In *Proc. of Europ. Workshop on Computational Geometry (EuroCG'19)*, 2019. Preprint.

A. Bonerath, B. Niedermann, and J.-H. Haunert. Retrieving alpha-Shapes and Schematic Polygonal Approximations for Sets of Points within Queried Temporal Ranges. In *Advances in Geographic Information Systems (ACM SIGSPATIAL'19)*, pages 249–258. ACM, 2019. doi: 10.1145/3347146.3359087.

A. Bonerath, J.-H. Haunert, and B. Niedermann. Tight Rectilinear Hulls of Simple Polygons. In *Proc. of Europ. Workshop on Computational Geometry (EuroCG'20)*, 2020a.

A. Bonerath, J.-H. Haunert, and B. Niedermann. Dynamic aggregation of geo-objects for the interactive exploration of research data. In *Publikationen der Deutschen Gesellschaft für Photogrammetrie, Fernerkundung und Geoinformation e.V., DGPF Jahrestagung 2020, Stuttgart, Germany*, volume 29, pages 488–496, 2020b.

A. Bonerath, B. Niedermann, J. Diederich, Y. Orgeig, J. Oehrlein, and J.-H. Haunert. A time-windowed data structure for spatial density maps. In *Advances in Geographic Information Systems (ACM SIGSPATIAL'20)*, pages 15–24. ACM, 2020c. doi: 10.1145/3397536.3422242.

A. Bonerath, Y. Dong, and J.-H. Haunert. An efficient data structure providing maps of the frequency of public transit service within user-specified time windows. *Advances in Cartography and GIScience of the ICA*, 4:1, 2023a. doi: 10.5194/ica-adv-4-1-2023. Special issue of 31st International Cartographic Conference (ICC'23).

A. Bonerath, J.-H. Haunert, J. S. B. Mitchell, and B. Niedermann. Shortcut hulls: Vertex-restricted outer simplifications of polygons. *Computational Geometry Theory and Applications*, page 101983, 2023b. doi: 10.1016/j.comgeo.2023.101983.

K. Buchin, B. Speckmann, and K. Verbeek. Flow map layout via spiral trees. *IEEE Trans. on Visualization and Computer Graphics*, 17(12):2536–2544, 2011. doi: 10.1109/tvcg.2011.202.

K. Buchin, W. Meulemans, A. van Renssen, and B. Speckmann. Area-preserving simplification and schematization of polygonal subdivisions. *ACM Trans. Spatial Algorithms Systems*, 2(1):2:1–2:36, 2016. doi: 10.1145/2818373.

D. Burghardt, S. Schmid, and J. Stoter. Investigations on cartographic constraint formalisation. In *10th ICA Workshop on Generalization and Multiple Representation*, volume 19, page 2, 2007.

S. Burigat and L. Chittaro. Visualizing the results of interactive queries for geographic data on mobile devices. In *Proc. of Int. Workshop Geographic Information Systems (GIS'05)*, pages 277–284. ACM, 2005. doi: 10.1145/1097064.1097103.

D. B. Carr, A. R. Olsen, and D. White. Hexagon Mosaic Maps for Display of Univariate and Bivariate Geographical Data. *Cartography and Geographic Information Systems*, 19(4):228–236, 1992. doi: 10.1559/152304092783721231.

J. W.-T. Chan, F. Y. L. Chin, X. Hong, and H.-F. Ting. Dynamic offline conflict-free coloring for unit disks. In *Proc. of Int. Workshop on Approximation and Online Algorithms (WAOA'08)*, volume 5426 of *Lecture Notes in Computer Science*, pages 241–252. Springer, 2008a. doi: 10.1007/978-3-540-93980-1_19.

S. Chan, L. Xiao, J. Gerth, and P. Hanrahan. Maintaining interactivity while exploring massive time series. In *3rd IEEE Symposium on Visual Analytics Science and Technology, IEEE VAST 2008, Columbus, OH, USA, October 19-24, 2008*, pages 59–66. IEEE Computer Society, 2008b. doi: 10.1109/VAST.2008.4677357.

T. M. Chan. Optimal output-sensitive convex hull algorithms in two and three dimensions. *Discrete and Computational Geometry*, 16(4): 361–368, 1996. doi: 10.1007/BF02712873.

T. M. Chan and S. Pratt. Time-windowed closest pair. In *Proc. Canadian Conf. on Computational Geometry (CCCG'15)*, 2015.

T. M. Chan and S. Pratt. Two approaches to building time-windowed geometric data structures. In *Proc. of Symp. on Computational Geometry (SoCG'16)*, volume 51 of *LIPIcs*, pages 28:1–28:15. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016. doi: 10.1007/s00453-019-00588-3.

F. Chanchary and A. Maheshwari. Time windowed data structures for graphs. *J. of Graph Algorithms and Applications*, 23(2):191–226, 2019. doi: 10.7155/jgaa.00489.

F. Chanchary, A. Maheshwari, and M. Smid. Window queries for problems on intersecting objects and maximal points*. In B. S. Panda and Partha P. Goswami, editors, *Proc. Algorithms and Discrete Applied Mathematics*, pages 199–213, Cham, 2018. Springer. doi: 10.1007/978-3-319-74180-2_17.

F. Chanchary, A. Maheshwari, and M. Smid. Querying relational event graphs using colored range searching data structures. *J. of Discrete Applied Mathematics*, 2019. doi: 10.1016/j.dam.2019.03.006.

A. R. Chaudhuri, B. B. Chaudhuri, and S. K. Parui. A novel approach to computation of the shape of a dot pattern and extraction of its perceptual border. *J. Computer Vision and Image Understanding*, 68(3):257–275, 1997. doi: 10.1006/cviu.1997.0550.

B. Chazelle. Filtering search: A new approach to query-answering. *SIAM J. Computing*, 15(3):703–724, 1986. doi: 10.1137/0215051.

B. Chazelle. Lower bounds for orthogonal range searching: I. the reporting case. *J. ACM*, 37(2):200–212, apr 1990. doi: 10.1145/77600.77614.

B. Chazelle. Triangulating a simple polygon in linear time. *Discrete and Computational Geometry*, 6:485–524, 1991. doi: 10.1007/BF02574703.

B. Chazelle and L. J. Guibas. Fractional cascading: I. A data structuring technique. *Algorithmica*, 1(1-4):133–162, 1986a. doi: 10.1007/bf01840440.

B. Chazelle and L. J. Guibas. Fractional cascading: II. applications. *Algorithmica*, 1(1-4):163–191, 1986b. doi: 10.1007/bf01840441.

L. P. Chew. Constrained Delaunay triangulations. *Algorithmica*, 4(1):97–108, 1989. doi: 10.1007/BF01553881.

M. Chimani, T. C. van Dijk, and J.-H. Haunert. How to eat a graph: Computing selection sequences for the continuous generalization of road networks. In *Advances in Geographic Information Systems (ACM SIGSPATIAL'14)*, page 243–252. ACM, 2014. doi: 10.1145/2666310.2666414.

F. Y. L. Chin and C. A. Wang. Finding the constrained Delaunay triangulation and constrained voronoi diagram of a simple polygon in linear time. *SIAM J. Computing*, 28(2):471–486, 1998. doi: 10.1137/S0097539795285916.

E. C Chukwuma, O. A. Nwoke, and D. O. Amaefule. Gross electrical energy production potential from agricultural waste: a gis-base assessment of anambra state of nigeria. In *Proc. of Int. Meeting (ASABE'19)*, page 1. ASABE, 2019. doi: 10.13031/aim.201900033.

K. A. Cook and J. J. Thomas. *Illuminating the Path: The Research and Development Agenda for Visual Analytics*. IEEE, 5 2005. ISBN 9780769523231.

T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT press, 2009. ISBN 9780262033848.

J. W. Crampton. Interactivity types in geographic visualization. *Cartography and Geographic Information Science*, 29(2):85–98, 2002. doi: 10.1559/152304002782053314.

J. J. Daymude, R. Gmyr, K. Hinnenthal, I. Kostitsyna, C. Scheideler, and A. W. Richa. Convex hull formation for programmable matter. In *Proc. of 21st Int. Conf. on Distributed Computing and Networking, ICDCN 2020*, pages 2:1–2:10. ACM, 2020. doi: 10.1145/3369740.3372916.

M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 3rd edition, 2008. ISBN 9783540779735.

M. de Berg, W. Meulemans, and B. Speckmann. Delineating imprecise regions via shortest-path graphs. In *Advances in Geographic Information Systems (ACM SIGSPATIAL'11)*, GIS '11, pages 271–280. ACM, 2011. doi: 10.1145/2093973.2094010.

C. A. de Lara Pahins, S. A. Stephens, C. Scheidegger, and J. L. D. Comba. Hashedcubes: Simple, low memory, real-time visual exploration of big data. *IEEE Trans. on Visualization and Computer Graphics*, 23(1):671–680, 2017. doi: 10.1109/TVCG.2016.2598624.

J. Diederich. Effiziente Generierung von Dichtekarten für temporale Bereichsanfragen auf grössen Mengen georeferenzierter Ereignisse, November 2019. M.Sc. thesis.

Y. Dong. A time-windowed data structure for the visualization of traffic in transportation graphs, May 2021. M.Sc. thesis.

D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica. Int. J. for Geographic Information and Geovisualization*, 10(2):112–122, 1973. doi: 10.1002/9780470669488.ch2.

A. Driemel and S. Har-Peled. Jaywalking your dog: Computing the fréchet distance with shortcuts. *SIAM J. Computing*, 42(5):1830–1866, 2013. doi: 10.1137/120865112.

M. Duckham, L. Kulik, M. F. Worboys, and A. Galton. Efficient generation of simple polygons for characterizing the shape of a set of points in the plane. *Pattern Recognition*, 41(10):3224–3236, 2008. doi: 10.1016/j.patcog.2008.03.023.

H. Edelsbrunner. A new approach to rectangle intersections part i. *Int. J. Computer Mathematics*, 13(3-4):209–219, 1983. doi: 10.1080/00207168308803364.

H. Edelsbrunner. Weighted alpha shapes. Technical Report UIUCDCS-R-92-1760, University of Illinois at Urbana-Champaign, Champaign, IL, USA, 1992.

H. Edelsbrunner. Alpha shapes – a survey. *Tessellations in the Sciences*, 27:1–25, 2010.

H. Edelsbrunner, H. A. Maurer, F. P. Preparata, A. L. Rosenberg, E. Welzl, and D. Wood. Stabbing line segments. *BIT Computer Science Section*, 22(3):274–281, 1982. doi: 10.1007/BF01934440.

H. Edelsbrunner, D. G. Kirkpatrick, and R. Seidel. On the shape of a set of points in the plane. *IEEE Trans. Information Theory*, 29(4):551–558, 1983. doi: 10.1109/TIT.1983.1056714.

D. Eppstein, M. J. van Kreveld, B. Speckmann, and F. Staals. Improved grid map layout by point set matching. *Int. J. Computational Geometry Appl.*, 25(2):101–122, 2015. doi: 10.1142/S0218195915500077.

R. Estkowski and J. S. B. Mitchell. Simplifying a polygonal subdivision while keeping it simple. In *Proc. of Symp. on Computational Geometry (SOCG'01)*, pages 40–49. ACM, 2001. doi: 10.1145/378583.378612.

N. Ferreira, L. Lins, D. Fink, S. Kelling, C. Wood, J. Freire, and C. Silva. Birdvis: Visualizing and understanding bird populations. *IEEE Trans. on Visualization and Computer Graphics*, 17:2374–83, 12 2011. doi: 10.1109/tvcg.2011.176.

A. Filtser and O. Filtser. Static and streaming data structures for fréchet distance queries. In *Proc. of Symp. on Discrete Algorithms (SODA'21)*, pages 1150–1170. SIAM, 2021. doi: 10.1137/1.9781611976465.71.

E. Fink and D. Wood. *Restricted-Orientation Convexity*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2004. ISBN 9783540668152.

Adrien Fonnet and Yannick Prié. Survey of immersive analytics. *IEEE Transactions on Visualization and Computer Graphics*, 27(3):2101–2122, 2021. doi: 10.1109/TVCG.2019.2929033.

A. Forsch, Y. Dehbi, B. Niedermann, J. Oehrlein, P. Rottmann, and J.-H. Haunert. Mult.dal travel-time maps with formally correct and schematic isochrones. *Trans. in GIS*, 25(6):3233–3256, 2021. doi: 10.1111/tgis.12821.

R. J. Fowler, M. Paterson, and S. L. Tanimoto. Optimal packing and covering in the plane are np-complete. *Information Processing Letters*, 12(3):133–137, 1981. doi: 10.1016/0020-0190(81)90111-3.

S. Funke, F. Krumpe, and S. Storandt. Crushing disks efficiently. In *Proc. of Int. Workshop Combinatorial Algorithms (IWOCA'16)*, volume 9843 of *Lecture Notes in Computer Science*, pages 43–54. Springer, 2016. doi: 10.1007/978-3-319-44543-4_4.

H. N. Gabow, J. L. Bentley, and R. E. Tarjan. Scaling and related techniques for geometry problems. In *Proc. of 16th Annual ACM Symposium on Theory of Computing (STOC'84)*, page 135–143. ACM, 1984. doi: 10.1145/800057.808675.

M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979. ISBN 9780716710448.

M. R. Garey, D. S. Johnson, F. P. Preparata, and R. Endre Tarjan. Triangulating a simple polygon. *Information Processing Letters*, 7(4):175–179, 1978. doi: 10.1016/0020-0190(78)90062-5.

A. Gemsa, M. Nöllenburg, and I. Rutter. Evaluation of labeling strategies for rotating maps. *ACM J. of Experimental Algorithmics*, 21(1):1.4:1–1.4:21, 2016a. doi: 10.1145/2851493.

A. Gemsa, M. Nöllenburg, and I. Rutter. Consistent labeling of rotating maps. *J. of Computational Geometry*, 7(1):308–331, 2016b. doi: 10.20382/jocg.v7i1a15.

A. Gemsa, B. Niedermann, and M. Nöllenburg. A unified model and algorithms for temporal map labeling. *Algorithmica*, 82(10):2709–2736, 2020. doi: 10.1007/s00453-020-00694-7.

J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub totals. *Data Min. Knowl. Discov.*, 1(1):29–53, 1997. doi: 10.1023/A:1009726021843.

A. Gupta and I. S. Mumick. *Materialized views: techniques, implementations, and applications*. MIT press, 1999. ISBN 9780262571227.

G. Hake, D. Grünreich, and L. Meng. *Kartographie: Visualisierung raum-zeitlicher Informationen*. Walter de Gruyter, 2002. ISBN 9783110164039.

J. Haslett, R. Bradley, P. Craig, A. Unwin, and G. Wills. Dynamic graphics for exploring spatial data with application to locating global and local anomalies. *The American Statistician*, 45(3):234–242, 1991. doi: 10.2307/2684298.

J.-H. Haunert and A. Wolff. Optimal and topologically safe simplification of building footprints. In *Advances in Geographic Information Systems (ACM SIGSPATIAL'10)*, pages 192–201. ACM, 2010. doi: 10.1145/1869790.1869819.

J.-H. Haunert and A. Wolff. Beyond maximum independent set: an extended integer programming formulation for point labeling. *ISPRS Int. J. of Geo-Information*, 6(11):342, 2017. doi: 10.3390/ijgi6110342.

J.-H. Haunert, A. Wolff, et al. Optimal simplification of building ground plans. In *Proc. of 21st ISPRS Congress*, pages 372–378, 2008.

H. Hochheiser and B. Shneiderman. Dynamic query tools for time series data sets: Timebox widgets for interactive exploration. *Information Sciences*, 3(1):1–18, 2004. doi: 10.1057/palgrave.ivs.9500061.

B. Jacobsen, M. Wallinger, S. G. Kobourov, and M. Nöllenburg. Metrosets: Visualizing sets as metro maps. *IEEE Trans. on Visualization and Computer Graphics*, 27(2):1257–1267, 2021. doi: 10.1109/TVCG.2020.3030475.

C. B. Jones, G. L. Bundy, and M. J. Ware. Map generalization with a triangulated data structure. *Cartography and Geographic Information Systems*, 22(4):317–331, 1995. doi: 10.1559/152304095782540221.

T. C. Kao and D. M. Mount. Incremental construction and dynamic maintenance of constrained Delaunay triangulations. In *Proc. Canadian Conf. on Computational Geometry (CCCG'92)*, pages 170–175, 1992.

T. Kapler and W. Wright. Geotime information visualization. *Information Visualization*, 4(2):136–146, 2005. doi: 10.1057/palgrave.ivs.9500097.

D. A. Keim, G. L. Andrienko, J.-D. Fekete, C. Görg, J. Kohlhammer, and G. Melançon. Visual analytics: Definition, process, and challenges. In *Information Visualization - Human-Centered Issues and Perspectives*, volume 4950 of *Lecture Notes in Computer Science*, pages 154–175. Springer, 2008. doi: 10.1007/978-3-540-70956-5_7.

K. Kiyosugi, C. B. Connor, D. Zhao, L. J. Connor, and K. Tanaka. Relationships between volcano distribution, crustal structure, and p-wave tomography: an example from the abu monogenetic volcano group, sw japan. *Bulletin of volcanology*, 72(3):331–340, 2010. doi: 10.1007/s00445-009-0316-4.

M. Konzack, P. Gijsbers, F. Timmers, E. van Loon, M. A Westenberg, and K. Buchin. Visual exploration of migration patterns in gull data. *Proc. of Symp. on Information Visualization (IEEE InfoVis'19)*, 18(1):138–152, 2019. doi: 10.1177/1473871617751245.

M.-J. Kraak. Geovisualization illustrated. *ISPRS J. of Photogrammetry and Remote Sensing*, 57(5-6):390–399, 2003. doi: 10.1016/s0924-2716(02)00167-3.

D. Krasnoshchekov and V. Polishchuk. Robust curve reconstruction with k-order alpha-shapes. In *Proc. of IEEE Int. Conf. Shape Modeling and Applications*, pages 279–280, 2008. doi: 10.1109/SMI.2008.4548006.

D. N. Krasnoshchekov and V. Polishchuk. Order-k alpha-hulls and alpha-shapes. *Information Processing Letters*, 114(1-2):76–83, 2014. doi: 10.1016/j.ipl.2013.07.023.

D.-T. Lee and A. K. Lin. Generalized Delaunay triangulation for planar graphs. *Discrete and Computational Geometry*, 1(3):201–217, 1986. doi: 10.1007/bf02187695.

D. T. Lee, C.-D. Yang, and C. K. Wong. Rectilinear paths among rectilinear obstacles. *Discret. Appl. Math.*, 70(3):185–215, 1996. doi: 10.1016/0166-218X(96)80467-7.

S. T. Leutenegger, M. A. Lopez, and J. Edgington. STR: a simple and efficient algorithm for R-tree packing. In *Proc. of Int. Conf. Data Engineering*, pages 497–506, 1997. doi: 10.1109/ICDE.1997.582015.

J. Li and T. Ai. A triangulated spatial model for detection of spatial characteristics of GIS data. In *Proc. of Int. Conf. on Progress in Informatics and Computing, PIC 2010*, volume 1, pages 155–159. IEEE, 2010. doi: 10.1109/PIC.2010.5687417.

J. Liang, H. Edelsbrunner, P. Fu, P. V. Sudhakar, and S. Subramaniam. Analytical shape computation of macromolecules: I. molecular area and volume through alpha shape. *Proteins: Structure, Function, and Bioinformatics*, 33(1):1–17, 1998. doi: 10.1002/(SICI)1097-0134(19981001)33:13.3.CO;2-9.

C.-S. Liao, C.-W. Liang, and S.-H. Poon. Approximation algorithms on consistent dynamic map labeling. *Theoretical Computer Science*, 640:84–93, 2016. doi: 10.1016/j.tcs.2016.06.006.

L. D. Lins, J. T. Klosowski, and C. E. Scheidegger. Nanocubes for real-time exploration of spatiotemporal datasets. *IEEE Trans. on Visualization and Computer Graphics*, 19(12):2456–2465, 2013. doi: 10.1109/TVCG.2013.179.

Z. Liu, B. Jiang, and J. Heer. *imMens*: Real-time visual querying of big data. *Computer Graphics Forum*, 32(3):421–430, 2013. doi: 10.1111/cgf.12129.

K. Marriott, F. Schreiber, T. Dwyer, K. Klein, N.H. Riche, T. Itoh, W. Stuerzlinger, and B.H. Thomas. *Immersive Analytics*. Lecture Notes in Computer Science. Springer International Publishing, 2018. ISBN 9783030013882.

M. Meijers, P. van Oosterom, M. Driel, and R. Šuba. Web-based dissemination of continuously generalized space-scale cube data for smooth user interaction. *Int. J. of Cartography*, 6(1):152–176, 2020. doi: 10.1080/23729333.2019.1705144.

T. Mendel. Area-preserving subdivision simplification with topology constraints: Exactly and in practice. In *Proc. of 20th Workshop on Algorithm Engineering and Experiments, ALENEX 2018*, pages 117–128. SIAM, 2018. doi: 10.1137/1.9781611975055.11.

W. Meulemans. *Similarity measures and algorithms for cartographic schematization*. PhD thesis, Mathematics and Computer Science, 2014.

W. Meulemans, A. van Renssen, and B. Speckmann. Area-preserving subdivision schematization. In *Proc. of 6th Int. Conf. on Geographic Information Science, GIScience 2010*, volume 6292 of *Lecture Notes in Computer Science*, pages 160–174. Springer, 2010. doi: 10.1007/978-3-642-15300-6_12.

F. Miranda, M. Lage, H. Doraiswamy, C. Mydlarz, J. Salamon, Y. Lockerman, J. Freire, and C. T. Silva. Time lattice: A data structure for the interactive visual analysis of large time series. *Computer Graphics Forum*, 37(3):23–35, 2018. doi: 10.1111/cgf.13398.

J. S. B. Mitchell, V. Polishchuk, and M. Sysikaski. Minimum-link paths revisited. *J. of Computational Geometry*, 47(6):651–667, 2014. doi: 10.1016/j.comgeo.2013.12.005.

G. Neyer. Line simplification with restricted orientations. In *Proc. of 6th Workshop on Algorithms and Data Structures, WADS '99*, volume 1663 of *Lecture Notes in Computer Science*, pages 13–24. Springer, 1999. doi: 10.1007/3-540-48447-7_2.

S. Nickel, M. Sondag, W. Meulemans, S. G. Kobourov, J. Peltonen, and M. Nöllenburg. Multicriteria optimization for dynamic demers cartograms. *IEEE Trans. on Visualization and Computer Graphics*, 28(6):2376–2387, 2022. doi: 10.1109/TVCG.2022.3151227.

M. Nöllenburg, V. Polishchuk, and M. Sysikaski. Dynamic one-sided boundary labeling. In D. Agrawal, P. Zhang, A. El Abbadi, and M. F. Mokbel, editors, *Advances in Geographic Information Systems (ACM SIGSPATIAL'10)*, pages 310–319. ACM, 2010. doi: 10.1145/1869790.1869834.

M. Nöllenburg. A survey on automated metro map layout methods. In *Schematic Mapping Workshop 2014*, 2014.

J. L. G. Pallero. Robust line simplification on the plane. *J. Computers and Geoscience*, 61:152–159, 2013. doi: 10.1016/j.cageo.2013.08.011.

D. Peng and A. Wolff. Watch your data structures! In *Proc. 22nd Annual GIS Research UK Conf. (GISRUK'14)*, GISRUK, 2014.

D. Peng, A. Wollf, and J.-H. Haunert. Finding optimal sequences for area aggregation: A* vs. integer linear programming. *ACM Trans. on Spatial Algorithms and Systems*, 7(1):4:1–4:40, 2020. doi: 10.1145/3409290.

D. Phan, L. Xiao, R. Yeh, P. Hanrahan, and T. Winograd. Flow map layout. In *Proc. of Information Visualisation (IEEE InfoVis'05)*, page 29. IEEE, 2005. doi: 10.1109/INFVIS.2005.1532150.

PostgreSQL Global Development Group. *PostgreSQL 11.0 Documentation*, 2018.

J. Rafael, J. Moreira, D. Mendes, M. Alves, and D. Gonçalves. Graceful Degradation for Real-time Visualization of Streaming Geospatial Data. In M.O. Agus, C. Garth, and A. Kerren, editors, *EuroVis 2021 - Short Papers*. The Eurographics Association, 2021. doi: 10.2312/evs.20211058.

G. J. E. Rawlins and D. Wood. Optimal computation of finitely oriented convex hulls. *Information and Computation*, 72(2):150–166, 1987. doi: 10.1016/0890-5401(87)90045-9.

A. C. Robinson, D. J. Peuquet, S. Pezanowski, F. A. Hardisty, and B. Swedberg. Design and evaluation of a geovisual analytics system for uncovering patterns in spatio-temporal event data. *J. of Cartography and Geographic Information Science*, 44(3):216–228, 2017. doi: 10.1080/15230406.2016.1139467.

R. Roth. Interactive maps: What we know and what we need to know. *J. of Spatial Information Science*, 6:59–115, 06 2013. doi: 10.5311/JOSIS.2013.6.105.

P. Rottmann, A. Driemel, H. J. Haverkort, Heiko Röglin, and J.-H. Haunert. Bicriteria aggregation of polygons via graph cuts. In *Proc. 11th Int. Conf. on Geographic Information Science, GIScience 2021*, volume 208 of *LIPIcs*, pages 6:1–6:16, 2021. doi: 10.4230/LIPIcs.GIScience.2021.II.6.

A. Sayidov and R. Weibel. Generalization of geological maps: Aggregation and typification of polygon groups. In *short papers, posters and poster abstracts of the 22nd Conf. on Geographic Information Science, AGILE*, volume 2019, 2019.

R. Scheepens, N. Willems, H. van de Wetering, G. Andrienko, N. Andrienko, and J. J. van Wijk. Composite density maps for multivariate trajectories. *IEEE Trans. on Visualization and Computer Graphics*, 17(12):2518–2527, 2011. doi: 10.1109/tvcg.2011.181.

N. Schwartges, D. Allerkamp, J.-H. Haunert, and A. Wolff. Optimizing active ranges for point selection in dynamic maps. In *Proc. of ICA Generalisation Workshop (ICA'13)*, 2013. doi: 10.1016/j.comgeo.2009.03.006. 10 pages.

M. I. Shamos and D. Hoey. Closest-point problems. In *Proc. of 16th Symp. on Foundations of Computer Science, FOCS 1975*, pages 151–162. IEEE Computer Society, 1975. doi: 10.1109/SFCS.1975.8.

J. R. Shewchuk and B. C. Brown. Fast segment insertion and incremental construction of constrained Delaunay triangulations. *J. of Computational Geometry*, 48(8):554–574, 2015. doi: 10.1016/j.comgeo.2015.04.006.

B. Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *The craft of information visualization*, pages 364–371. Elsevier, 2003. doi: 10.1016/B978-155860915-0/50046-9.

B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman & Hall, 1986. ISBN 9780412246203.

P. Simonetto, D. Archambault, and S. G. Kobourov. Event-based dynamic graph visualisation. *IEEE Trans. on Visualization and Computer Graphics*, 26(7):2373–2386, 2020. doi: 10.1109/TVCG.2018.2886901.

A. Slingsby and E. van Loon. Exploratory visual analysis for animal movement ecology. *Computer Graphics Forum*, 35(3):471–480, 2016. doi: 10.1111/cgf.12923.

B. Speckmann and K. Verbeek. Homotopic c-oriented routing with few links and thick edges. *Computational Geometry*, 67:11–28, 2018. doi: 10.1016/j.comgeo.2017.10.005.

D. Spretke, P. Bak, H. Janetzko, B. Kranstauber, F. Mansmann, and S. Davidson. Exploration through enrichment: a visual analytics approach for animal movement. In *Proc. Advances in Geographic Information Systems (ACM SIGSPATIAL'11)*, pages 421–424. ACM, 2011. doi: 10.1145/2093973.2094038.

S. Steiniger, D. Burghardt, and R. Weibel. Recognition of island structures for map generalization. In *Advances in Geographic Information Systems (ACM SIGSPATIAL'06)*, pages 67–74. ACM, 2006. doi: 10.1145/1183471.1183484.

E. W. Stienen, P. Desmet, B. Aelterman, W. Courtens, S. Feys, N. Vanermen, H. Verstraete, M. van de Walle, K. Deneudt, F. Hernandez, R. Houthoofdt, B. Vanhoorne, W. Bouten, R. Buijs, M. M. Kavelaars, W. Müller, D. Herman, H. Matheve, A. Sotillo, and L. Lens. Bird tracking – gps tracking of lesser black-backed gulls and herring gulls breeding at the southern north sea coast. version 5.6, 2017. Research Institute for Nature and Forest (INBO).

G.-D. Sun, Y.-C. Wu, R.-H. Liang, and S.-X. Liu. A survey of visual analytics techniques and applications: State-of-the-art research and future challenges. *J. of Computer Science and Technology*, 28:852–867, 09 2013. doi: 10.1007/s11390-013-1383-8.

E. Tanin, R. Beigel, and B. Shneiderman. Incremental data structures and algorithms for dynamic query interfaces. *SIGMOD Rec.*, 25(4): 21–24, 1996. doi: 10.1145/245882.245891.

E. R. Tufte. The visual display of quantitative information. *J. for Healthcare Quality (JHQ)*, 7(3):15, 1985. doi: 10.1097/01445442-198507000-00012.

T. C. van Dijk, K. Fleszar, J.-H. Haunert, and J. Spoerhase. Road segment selection with strokes and stability. In *Int. Workshop on Map Interaction (MapInteract'13)*, pages 72–77. ACM, 2013. doi: 10.1145/2534931.2534936.

T. C. van Dijk, A. van Goethem, J.-H. Haunert, W. Meulemans, and B. Speckmann. Map schematization with circular arcs. In *Proc. of 8th Int. Conf. on Geographic Information Science, GIScience 2014*, volume 8728 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2014. doi: 10.1007/978-3-319-11593-1_1.

A. van Goethem, W. Meulemans, B. Speckmann, and J. Wood. Exploring curved schematization of territorial outlines. *IEEE Trans. on Visualization and Computer Graphics*, 21(8):889–902, 2015. doi: 10.1109/TVCG.2015.2401025.

M. van Kreveld, T. van Lankveld, and M. de Rie. (α, δ)-sleeves for reconstruction of rectilinear building facets. In *Progress and New Trends in 3D Geoinformation Sciences*, pages 231–247. Springer, 2013. doi: 10.1007/978-3-642-29793-9_13.

J. Vauhkonen, I. Korpela, M. Maltamo, and T. Tokola. Imputation of single-tree attributes using airborne laser scanning-based height, intensity, and alpha shape metrics. *Remote Sensing of Environment*, 114(6):1263–1276, 2010. doi: 10.1016/j.rse.2010.01.016.

M. Williams, W. Kuhn, and M. Painho. Interactive maps: What we know and what we need to know. *J. of Spatial Information Science*, 6(1): 59–115, 2013. doi: 10.5311/JOSIS.2013.6.105.

H.-Y. Wu, B. Niedermann, S. Takahashi, M. J. Roberts, and M. Nöllenburg. A survey on transit map layout - from design, machine, and human perspectives. *Computer Graphics Forum (Proc. Eurographics)*, 39(3):619–646, 2020. doi: 10.1111/cgf.14030.

Y. Wu, N. Cao, D. Gotz, Y. Tan, and D. A. Keim. A survey on visual analytics of social media data. *IEEE Trans. Multimedia*, 18(11): 2135–2148, 2016. doi: 10.1109/tmm.2016.2614220.

P. Yoeli. The logic of automated map lettering. *The Cartographic J.*, 9(2):99–108, 1972. doi: 10.1179/000870472787352505.

X. Zhang and E. Guilbert. A multi-agent system approach for feature-driven generalization of isobathymetric line. In *Advances in Cartography and GIScience. Volume 1*, pages 477–495. Springer, 2011. doi: 10.1007/978-3-642-19143-5_27.

X. Zhang, S.-H. Poon, S. L., M. Li, and V. C. S. Lee. Consistent dynamic map labeling with fairness and importance. *Computer Aided Geometric Design*, 81:101892, 2020. doi: 10.1016/j.cagd.2020.101892.