Similarity Measures and Algorithms for Cartographic Schematization

Wouter Meulemans

Similarity Measures and Algorithms for Cartographic Schematization

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de rector magnificus prof.dr.ir. C.J. van Duijn, voor een commissie aangewezen door het College voor Promoties, in het openbaar te verdedigen op dinsdag 2 september 2014 om 16:00 uur

door

Wouter Meulemans

geboren te Breda

Part of the work in this thesis is supported by the Netherlands' Organization for Scientific Research (NWO) under project no. 639.022.707.

The work in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics).

Implementations used for experimental results were made in Java using the NetBeans IDE. All figures were created in the Ipe extensible drawing editor. Color schemes were obtained from `www.colorbrewer2.org`.

Cover Design: Wouter Meulemans
Printing: Eindhoven University Press

A catalogue record is available from the
Eindhoven University of Technology Library
ISBN: 978-90-386-3673-3

Contents

Acknowledgments

After almost four years, some of the main results of my research have been bundled into this thesis. Though I proudly present this as “my research”, a number of people have had a major influence on me and this research, both directly and indirectly. Therefore, I’d like to take this opportunity to thank them. I apologize for anyone I may forget to mention.

First of all, I would like to thank my supervisors, Bettina Speckmann and Kevin Buchin, for being there with council, suggestions, feedback, thoughts and ideas. Bettina was my supervisor already for a project of the Master’s Honors Programme and set me on the path of schematization, at the time described to me as “something with drawing a polygon using only horizontal and vertical lines”. I could have never imagined then that this topic would so capture my interest. I was very happy when Bettina hired me afterwards as a PhD student which allowed me to further investigate schematization research. At this point in time, Kevin also became involved as a supervisor, which led to a significant increase of attention towards the Fréchet distance: I would conjecture that no one has managed to leave Kevin’s office without being “infected” with the Fréchet distance.[1] I surely did not mind, as this posed many interesting questions and challenges. Both Bettina and Kevin have influenced my views of academia and science and taught me how to conduct, write and present research in a sound way. I greatly appreciate the way of supervising that allowed me to pursue my own research interests within computational geometry. I never heard “we cannot do that because...”, only “we can do that after...” (though it seems that we generate problems faster than we solve them).

The Algorithms group in Eindhoven has always been a friendly work environment. For this, I would like to thank everyone who has been part of the research group at some point over the last four years: Mark de Berg, Quirijn Bouts, Maike Buchin, Atlas Cook IV, Anne Driemel, Dirk Gerrits, Arthur van Goethem, Herman Haverkort, Amirali Khosravi, Maximilian Konzack, Irina Kostitsyna, Ali Mehrabi, Marcel Roeloffzen, Constantinos Tsirogiannis, Kevin Verbeek and Chris Volk. In particular, I would like to thank Arthur for our very interesting joint efforts on curved schematization. Also, special thanks go out those with whom I have shared an office, Ali, Arthur, Dirk and Marcel, for making the day-to-day activities a lot more interesting and for being a sounding board for ideas. Over the years, our group also included a number of visiting PhD students, Rafael Cano, Saeed

[1]Dirk Gerrits seems to have managed: I will simply consider him as “the exception that confirms the rule” as per the proverb.

Mehrabi, Farnaz Sheikhi and Georgina Wilcox, and several Master students (which I will not attempt to list, as I would surely forget someone). All have contributed to fascinating research presentations in our seminar and interesting discussions (about research, games, culture and many other topics). I would also like to thank the members of the Visualization group, in particular, Jack van Wijk and Michel Westenberg for their thoughts and suggestions on how to present work that combines visualization with algorithmic aspects.

Next, I would like to thank my collaborators from other institutes: Basak Alper, Thomas van Dijk, Tim Dwyer, Jan-Henrik Haunert, Nathalie Henry Riche, Wolfgang Mulzer, Andreas Reimer, André van Renssen and Jo Wood. In particular, I would like to thank Andreas, for walking up to me after my very first conference presentation at GIScience back in 2010. Ever since, we have been in close contact and, though our collaborative efforts have not been included here, the thesis would not be as it is today, if it was not for our interesting discussions. In addition, I would like to especially thank Tim and Nathalie for offering me an internship position at Microsoft: it was a great experience, one I will never forget!

Of course, I would also like to thank the members of my graduation committee, Jan-Henrik Haunert, Christian Knauer, Carola Wenk and Gerhard Woeginger, for thoroughly reading my thesis and providing useful feedback and suggestions.

On a personal level, I would like to thank my family and friends. In particular, I would like to thank my girlfriend, Emmy Dudok, for her continued love and support. She read through my thesis to provide suggestions and comments, when it was in a state far inferior to the copy you have before you today. In addition, I would like to thank my parents, Hans and Ingrid Meulemans, for encouraging my curiosity and supporting my academic studies and career. Finally, I would like to thank my brother, Maarten Meulemans, for fun discussions about and joint efforts on computer science, programming, games and much more.

Chapter 1

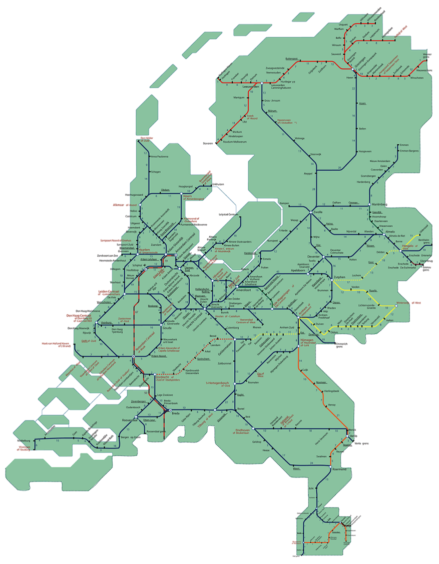
Introduction

A map is a visual representation of a geographic region, depicting a number of elements and features of the region. Elements may be geographic in nature—roads, rivers, buildings, et cetera—or indicate intangible information, related for example to social, economic or political aspects of the region. Maps are a common and intuitive way to communicate or analyze information in its geographic context. A *schematic map* depicts elements in an abstract, stylized and organized form. The purpose is to support the main information that the map has to convey. This style of visualizing geographic information typically requires a distortion of geographic reality: geographic accuracy of the map is relinquished in favor of its clarity. The organization and simple shapes in a schematic map reduce its visual complexity. This reduces the cognitive load of an observer: it is then much easier to mentally process the information in the map. As a result, the main information is immediately clear.

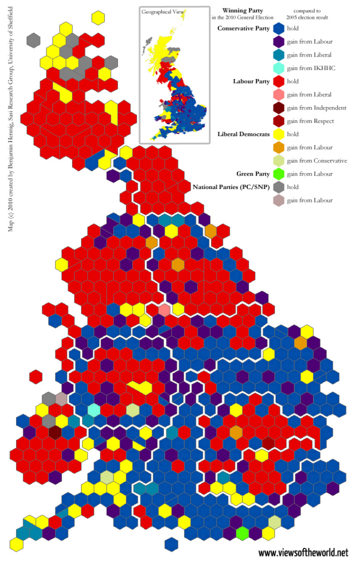
Schematic maps come in a variety of types and styles. A familiar type is a transit map such as the London Underground map or the Dutch railways map (see Figure 1.1(a)). A transit map is designed to convey connectivity information such that a traveler can easily find his or her way from one station to another. Exact geographic routes of the tracks are then mostly irrelevant as are other geographic elements such as roads and buildings. To support travelers in planning their route, it is often useful to visualize some minimal geographic context (like the country borders in the Dutch railways map). This helps the user locate stations and may provide a visual clue of which network is displayed.

Another example of a schematic map is given in Figure 1.1(b). It is a distorted view of the United Kingdom in which every constituency is represented by a simple hexagon. This ensures that the (constant) size of a constituency in the map corresponds to a representation of the seats in the British Parliament. As a result, it is much easier to assess the depicted change in voting behavior. If this map would simply color geographically accurate regions, then large regions would be more visually dominant than small regions.

The importance of maps. It is often said that 80% of all data has a geographic component (though the veracity of this claim is difficult to establish [96]). Maps are a great tool to make overwhelming amounts of data accessible. In general, maps assist analyz-

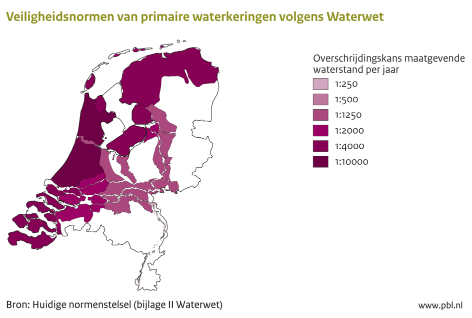


(a)

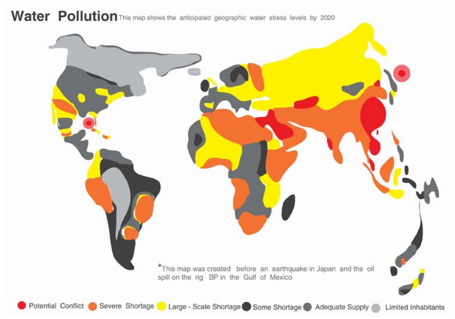


(b)

Figure 1.1 (a) The Dutch railways map [www.trein-kaart.nl, accessed November 2013]. (b) A schematic map showing change in voting behavior per constituency in the UK general election of 2010 [www.viewsoftheworld.net/?p=736, accessed April 2014].


Veiligheidsnormen van primaire waterkeringen volgens Waterwet
Overschrijdingskans maatgevende waterstand per jaar
1:250
1:500
1:1250
1:2000
1:4000
1:10000
Bron: Huidige normenstelsel (bijlage II Waterwet)
www.pbl.nl

(a)


Water Pollution
This map shows the anticipated geographic water stress levels by 2020
*This map was created before an earthquake in Japan and the oil spill on the rig BP in the Gulf of Mexico
Potential Conflict
Severe Shortage
Large - Scale Shortage
Some Shortage
Adequate Supply
Limited Inhabitants

(b)

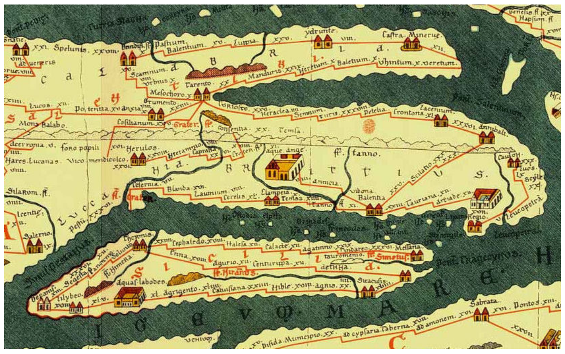
Figure 1.2 (a) Map showing safety norms for flood risk throughout the Netherlands [www.pbl.nl/nl/node/56494, accessed July 2014]. (b) Map showing estimated water shortage and pollution issues in 2020 [http://ftrctlb.com/node/169, accessed May 2014].

ing, communicating and discussing about information in a spatial setting. They are used for example in navigation, spatial planning, risk assessment and disaster management. Maps support rhetoric arguments as well as decision-making and opinion-building processes. By presenting information in an accessible visual way, maps eliminate the need for a person to crunch numbers. This takes advantage of the human natural capabilities of processing visual information. A map illustrating flood risks (Figure 1.2(a)) or expected water problems (Figure 1.2(b)) is much more striking, easier to interpret and more memorable than a long list of place names and numbers. When geographic accuracy is not a primary concern, schematic maps are desired to emphasize the essential information.

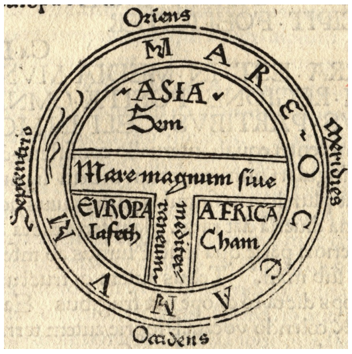
Maps that visualize information in a structural and organized way are not a modern invention. Various old schematic maps can be found that do not focus on spatial accuracy, but rather on high-level structures and information. The *Tabula Peutingeriana* (Figure 1.3(a)) depicts the road network of the Roman Empire spanning from modern Spain to India. This is a very elongated map with clear geographic distortion. The road network itself also has a schematic appearance, using only few lines to connect the various cities.

Figure 1.3(b) shows an example of a *mappa mundi*, a medieval world map. These medieval maps are not designed for navigation, but rather focus on the high-level structural organization of the continents, typically centered on Jerusalem. This example places Asia at the top, Europe in the bottom left, and Africa in the bottom right corner. The continents are visualized as two quartercircles (Europe and Africa) and one semicircle (Asia). A T-shape is used to represent the main bodies of water that separate the continents.

Making schematic maps. *Cartographic schematization* is the process involved in creating schematic maps from detailed information. Traditionally, this is done manually by cartographers. It involves selecting and prioritizing the important information as well as visualizing it in a suitable way. Hence, the design of a map is a laborious and time-



(a)



(b)

Figure 1.3 Old schematic maps. (a) Part of the *Tabula Peutingeriana*, dated around the fifth century. This part shows the southern part of modern Italy and Sicily. (b) A highly schematic world map in *Etymologiae* as printed by Günther Zainer in 1492.

consuming process. With the advent of computers and digitized data collection, new opportunities emerged to produce maps using computers. It in fact enables a highly flexible scenario in which computers produce personalized maps on demand using the digitally available information. Especially in time-critical situations (such as disaster management), it is desirable to obtain a suitable up-to-date map with a simple click of a button. But even in situations without strict time constraints, computers can assist in making maps: they are typically far more efficient than humans in processing large amounts of data. Moreover, in changing from paper maps to digital maps, new possibilities emerged to interact with maps. Combining computer science and cartography, *automated cartography* is a research field in which automated methods to produce maps are studied.

To compute a schematic map, we need methods that allow computers to perform cartographic schematization. That is, we need automated ways to transform detailed data into a schematic representation. However, this leaves some loose ends. Users do not have the expertise or time to find their own data sources nor may they be aware of the most effective way to visualize the information. Instead, a user expresses the need for a map on a certain topic. Based on a simple query, the system should deduce the map type, required data, level of detail, et cetera. This derived information can then be used by the schematization process to compute schematic representations. Finally, these representations are assembled into a map and presented to the user in a suitable way.

In many cases, systems are not targeted at solving all possible map needs. Rather, they aim to support users in specific areas, such as route maps, mapping geopolitical information or transit maps. The user need may then be more straightforward and data may already be available. In such cases, the addition of automated schematization methods are likely sufficient to compute schematic maps without human intervention.

In this thesis we investigate automated cartographic schematization, the key component for computing schematic maps. That is, we want to find the answer to the following question: how can a computer transform geographic data into a suitable schematic representation? The answer depends on the characteristics of a good schematic map. We identify four common properties.

(i) Schematic maps have a low visual complexity. Only the necessary information is shown and this is represented using only a few geometric shapes.
(ii) Simple geometric shapes are used to convey the schematic nature of the map.
(iii) Geographic relations, such as adjacent countries and relative directions between places, are maintained as well as possible.
(iv) Used geometric shapes resemble the actual geographic shapes that they represent.

The first two properties ensure that a map has a schematic “look and feel”. They relate to a low cognitive load and a map that visually conveys its schematic nature. The third and fourth criterion relate to the recognizability of a schematic map. The *mental map* (or cognitive map) is the recollection of a person about geographic entities and relations between these entities [168]. It is easier for a user to recognize a schematic map when it has a strong correlation to the user’s mental map. A mental map is often distorted [167]; in fact, it has recently been conjectured that a mental map more resembles a schematic map [173]. However, changes in the relations between regions may greatly interfere with this correspondence and thus should be avoided. Geographic relations alone may not be sufficient to obtain a recognizable map. Hence, the fourth property makes this explicit and states that a schematic map should not only correspond to a mental map but also to geographic reality.

Some properties are in conflict: a trade-off must be made between how important certain properties are. For example, reducing the number of geometric elements improves upon the visual complexity (i) but reduces the resemblance (iv). Depending on the nature of a specific schematic map, some properties are considered more important than others.

In this thesis we study *algorithms* to automatically compute a schematic map based on geographic data. To design and evaluate algorithms, we need a way of formalizing the properties mentioned above. Especially the fourth property (resemblance) poses interesting challenges. To formalize this, we use *similarity measures* that quantify resemblance between shapes. In Section 1.1 we analyze cartographic schematization more thoroughly and briefly discuss the wider field of automated cartography. General concepts of similarity measures and algorithms are discussed in Section 1.2 and Section 1.3 respectively.

1.1 Cartographic schematization

The abstract depiction in a schematic map emphasizes important structures over the finer details that make up reality. The essential information is clarified by removing unnecessary details. Such details could otherwise cause clutter and distract from or even obscure the main information. The information that is shown is visualized in a structured, stylistic way to reduce the cognitive load. This assists the observer in seeing the primary information that the schematic map intends to convey. Due to their structural stylized appearance, schematic maps are also referred to as diagrams or diagrammatic maps. In fact, the distortion in schematic maps may be reason for some to consider schematic maps not as actual “maps” [82]. Many definitions of the word “map” have been used throughout history [12][1]. The International Cartographic Association’s current definition of a map [111] includes all visual representations in which “spatial relations are of primary relevance”; this is most certainly the case for schematic maps.

Because a schematic map has an organized and stylistic appearance, it is immediately clear that the map in question is not a geographically accurate map. This makes schematic maps very suitable to convey high-level structures and summarized, spatially inaccurate or uncertain information. For example, schematic maps are suitable to convey the results of analysis or simulation: e.g. Wolf and Flather [178] use schematized maps to illustrate their results of modeling and analyzing the North Sea storm of 1953 (see Figure 1.4).

Suppose that inaccurate information is visualized in the context of an unschematized map that is (or appears) geographically accurate. The displayed information—which is less accurate than the detailed map—can then be unjustly interpreted as equally accurate. This is referred to as an *illusion of accuracy* [118, 128, 179]. In general, an illusion of accuracy may occur when various types of information are represented with different spatial

[1]See also `www.maphist.nl/discpapers.html`, accessed April 2014.

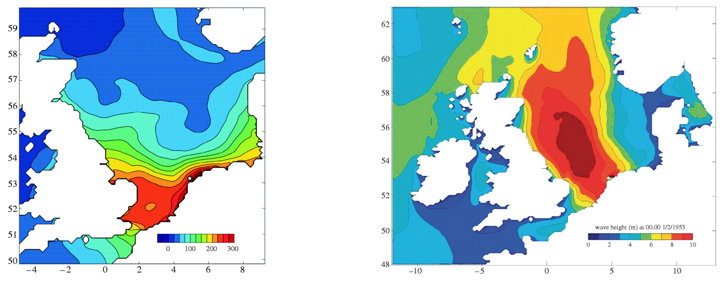


Figure 1.4 Two examples of the schematic maps that illustrate results of modeling and analysis by Wolf and Flather [178].

precision or accuracy in a single map. Schematic maps avoid this illusion by using very simple and stylistic geometric shapes to visually convey that the map is not geographically accurate. An example of this illusion may occur with a metro map. Stations are purposefully displaced to emphasize connections and to make room for placing the station names. If the map appears too accurate—due to the addition of, e.g., detailed rivers, shorelines or roads—, a traveler may decide to walk from one station to another, whereas they are actually too far apart.

1.1.1 Characterization

We identified four common properties of schematic maps. To facilitate the design of algorithms that produce good schematic maps, we require a formalization of these properties. The formalization depends on the nature of the schematic map. For the purpose of this thesis, we use a basic characterization of schematic maps. It is based on two characteristics: the type of geographic object and the geometric style.

Type of object. For the first characteristic, the type of object, we distinguish between networks and regions. Networks can, for example, consist of roads, railways or rivers. A network is fully defined by its linear features; areas enclosed by these features carry no inherent value. Transit maps are an example of maps that show schematic networks.

Regions, on the other hand, correspond to geographic elements that span an area. Regions can effectively be described via their boundaries, but the semantics of these boundaries are different from networks. The geographic element is the area enclosed by the boundaries; the boundary itself has no other meaning than separating two regions. Examples are administrative regions, such as countries, provinces and states. We refer to the boundaries of such administrative regions as *territorial outlines*. These elements are used in many types of schematic maps to provide geographic context, to indicate regions of interest or to display certain properties of a region.

Geometric style. The second characteristic of a schematic map is its geometric style. Examples of these styles are given in Figure 1.5. Following the success of Beck's map of the London Underground [87], octilinear designs have been popular for transit maps all over

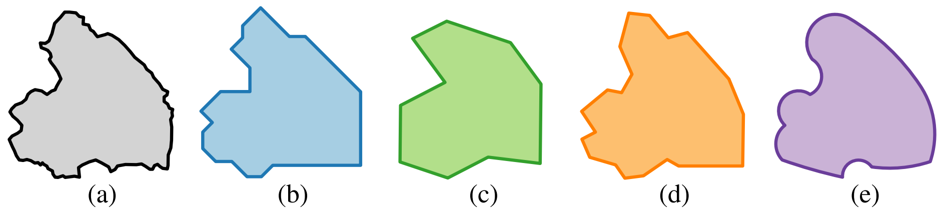


Figure 1.5 Geometric styles illustrated on Drenthe, a province of the Netherlands (a). (b) $\mathcal{C}$-oriented schematization, result of Chapter 7. (c) Parallelism, using the method by Reimer and Meulemans [150]. (d) Unrestricted schematization, result of Chapter 7. (e) Curved schematization, using the method by Van Goethem *et al.* [92].

the world [137]. In such designs, all lines are either horizontal, vertical or diagonal (45 degrees). More generally, a schematic map may restrict its lines to adhere to a certain discrete set $\mathcal{C}$ of orientations; we refer to this as *$\mathcal{C}$-oriented schematization*. Other frequently used orientation sets are rectilinear (horizontal and vertical lines only) and hexilinear (angles are multiples of 60 degrees). Research into the automated construction of $\mathcal{C}$-oriented schematic maps has mostly focused on networks (e.g. [47, 126, 133]).

Another geometric style is *parallelism* [150]. Lines should be drawn parallel to one another whenever possible. This implies that a low number of orientations should be used, but no prescribed orientations are enforced. Therefore, parallelism can be seen as a more flexible variant of $\mathcal{C}$-oriented maps.

Sometimes, no specific line orientations are enforced nor are relative orientations considered. This results in *unrestricted* schematization. This effectively equates schematization to simplification (a reduction in detail) with a low target complexity. Results of such approaches (e.g. [56, 78]) sometimes lack a “schematic look” that arises from applying a stronger geometric style.

Smooth curves are frequently used to represent elements in manually drawn schematic maps [149, 151]. This exploits the high expressive power of curves: often only a few curves are needed to accurately represent a shape. Research into automated methods for curved schematization has recently been initiated [79, 91, 92]. Curved schematization can be refined depending on the type of curves (e.g. Bézier curves or circular arcs).

One geometric style is not necessarily better than another. Rather, it is a design decision: the intent of the map may play a large role in selecting the best geometric style. Moreover, some styles fit more naturally to a certain geographic object than others.

Other characteristics. Schematic maps may be characterized through a number of other properties, resulting in a (possibly more fine-grained) classification of subtypes in schematic maps. For example, Reimer [149] distinguishes eight types of schematic maps, depending on origin and intent: (topological) schematic maps; chorematic diagrams; geodesign maps; geopolitical maps; propaganda maps; mass media maps and map-like infographics; (drawings of) mental maps; and schematic maps in education. All

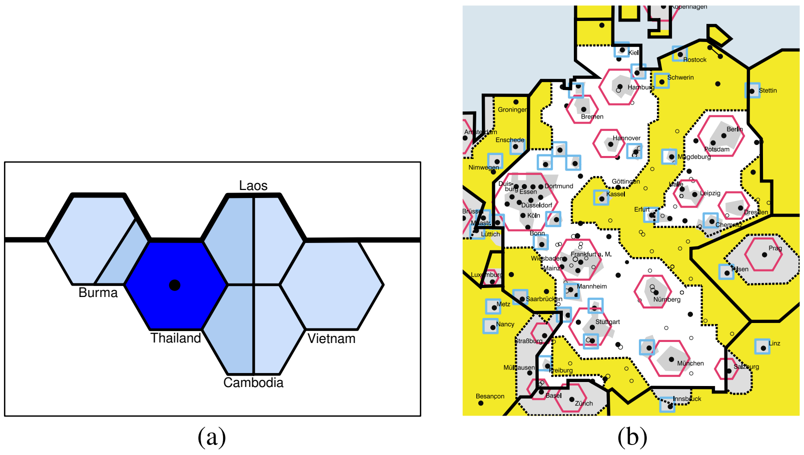


Figure 1.6 (a) Part of a chorematic diagram depicting the economic influence of Thailand on its neighboring countries. From translation by Reimer [149, Figure 13]; original map by Bruneau and Marcotte [31]. (b) A schematic geodesign map depicting agglomerative and influential regions in and around Germany. The region boundaries are stylized: many lines are either horizontal, vertical, or 45-degree diagonal. Modified to emphasize region boundaries from original map in *Raumordnungspolitischer Orientierungsrahmen 1993*.

of these schematic maps have in common that minimalistic representations are used to convey a message as clearly as possible without emphasizing on geographic accuracy. We provide a brief description of the first three.

In Reimer's classification a *schematic map* visualizes (typically linear) elements with a focus on topological aspects. In these maps, geographic relations are distorted for greater clarity. In the characterization we introduced above, this mostly refers to schematic networks but also allows for an inclusion of schematic region boundaries. We provided an example of such a map—which indeed combines a schematic network with a schematic region boundary—in Figure 1.1(a).

A *chorematic diagram* summarizes the results of an analysis of processes (typically geographic, social, economic or political) in their geographic context. Hence, schematization is often used in these diagrams to visually convey the summarizing nature of the map. An example of a chorematic diagram is given in Figure 1.6(a).

Geodesign maps are similar to chorematic diagrams: many techniques employed for geodesign maps correspond to those for chorematic diagrams. However, they differ in their intent. Rather than displaying the results of analysis ("what is observed"), they visualize strategic political scenarios and high-level spatial design and planning ("what is desired"). Again, the nature of the information warrants the use of a schematic map style. An example of a schematic geodesign map is given in Figure 1.6(b).

1.1.2 $\mathcal{C}$-oriented schematization of territorial outlines

In this thesis we focus on $\mathcal{C}$-oriented schematization of territorial outlines. We need to know what defines such a schematic map and how we can assess its quality, in order to formulate algorithmic problems and design algorithms. Therefore, we formulate the four beforementioned general properties as four criteria for $\mathcal{C}$-oriented schematization of territorial outlines.

(1) Schematic outlines should use few line segments.
(2) All line segments should be oriented according to $\mathcal{C}$.
(3) The geographic relations between regions of the schematic outlines are equivalent to those of the geographic regions (correct *topology*).
(4) Schematic outlines *resemble* the corresponding geographic outlines.

The combination of a low number of lines and the restricted line orientations results in a schematic outline. Individually though, these restrictions are not always sufficient. As illustrated in Figure 1.7(a), a map with many $\mathcal{C}$-oriented lines is still visually complex and does not appear schematized. Although the choice for $\mathcal{C}$-oriented lines is a design decision (see Section 1.1.1), it is a strong visual cue that the map is schematic in nature, in contrast to using unrestricted geometry. Figure 1.7(b) illustrates this by showing an outline consisting of only few line segments: despite its low visual complexity, it lacks a stylistic restriction of geometry and has the risk of being misinterpreted as a simple but accurate representation. The combination ensures a low visual complexity and avoids a potential illusion of accuracy.

The third criterion requires us to maintain relations between geographic regions. This is referred to as correct topology. Incorrect topology interferes with an observer's mental map, assuming that the mental map contains correct geographic relations. A simple example of a topology violation is a change in which regions share borders, as illustrated in Figure 1.8. As mentioned before, only maintaining topology is typically not sufficient for a recognizable map. Hence, the fourth criterion states that there must be some degree of resemblance between geographic reality and the schematic map.

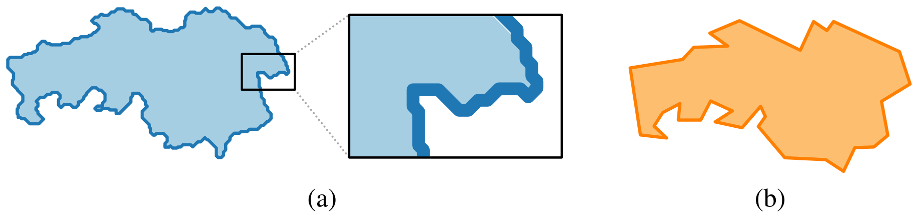


Figure 1.7 Restricted geometry or a low complexity alone may not suffice for schematization. (a) Due to its high complexity, this $\mathcal{C}$-oriented (octilinear) shape lacks a schematized appearance. (b) Due to its unrestricted linear geometry, this shape may be mistaken for a low-detail but accurate map.

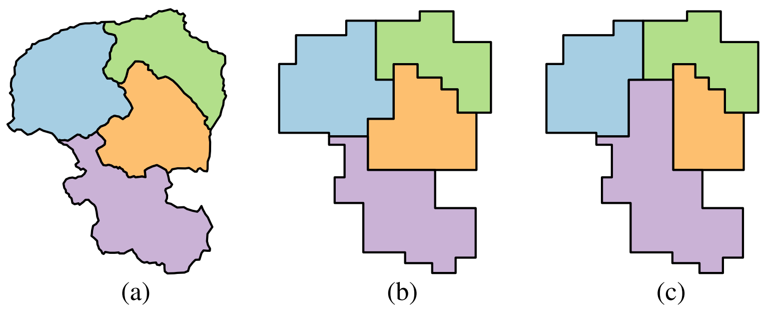


Figure 1.8 (a) Geographic outlines of four provinces in the Netherlands. (b) A schematic map with equivalent topology. (c) A change in topology: Overijssel (purple) and Groningen (green) should not be adjacent.

The first three criteria can be formalized comparatively easily (see Chapter 2). The last criterion, however, is less straightforward. A similarity measure quantifies the resemblance between shapes. However, many different measures exist. To automate $\mathcal{C}$-oriented schematization for territorial outlines, we need to develop algorithms that take the above criteria into account. These two aspects of automated schematization form the central focus of this thesis: similarity measures and algorithms for cartographic schematization. They are research topics in their own right; we briefly discuss the corresponding general concepts in Section 1.2 and Section 1.3. However, we first discuss some aspects of the field of automated cartography which encompasses automated schematization.

1.1.3 Automated cartography

Research in *automated cartography* studies automated methods for making maps. It encompasses all aspects related to map production, for a wide range of map types. In addition to schematic maps, this includes other types of maps such as topographic and thematic maps. A topographic map displays detailed geographic elements such as roads, rivers and buildings. The intent of such a map is to be usable for a wide array of possibly unknown tasks. In contrast, a thematic map is designed for a single purpose and focuses on one theme, one specific type of information.

Geographic information systems [107] store and process geographic data. These systems emerged more or less simultaneously with automated cartography. Not surprisingly, automated cartography is closely related to—and often considered a subfield of—the underlying scientific field, *geographic information science*. The methods developed for automated cartography may use these systems to retrieve detailed geographic data and transform them into a map.

Automated cartography poses many interesting algorithmic challenges. One such problem is the placement of textual labels in maps. This labeling problem can be formalized in a variety of ways; not surprisingly, it has received significant attention in both automated cartography and algorithmic research (e.g. [49, 58, 65, 81, 108, 115, 174, 183]).

Thematic maps often require specialized techniques and algorithms depending on the type of thematic information (see e.g. the work of Verbeek [170] for several examples). In addition to schematization, the problems of *simplification* and *generalization* are particularly relevant in the context of this thesis. Simplification refers to a reduction of detail. Generalization is concerned with producing scale-appropriate representations for detailed (e.g. topographic) maps.

Schematization, simplification and generalization. We introduced two concepts, simplification and generalization, that are very similar to the concept of schematization. Below, we briefly relate them to each other, highlight the differences and provide some more details for generalization and simplification.

Simplification is a means to an end: the reduction of detail. By itself, simplification does not produce a map as it lacks a design aspect. It can be used to achieve results with very high or very low detail, depending on the parametrization. It is, however, an essential part of both generalization and schematization.

Generalization and schematization are inherently different in their purpose and thus in their quality criteria. Due to the nature of topographic maps, generalization seeks to maximize the amount of detail, but is constrained by legibility of the map. It never uses strong stylization as this results in unwarranted distortion. For generalization, simplification is often parameterized by some maximal error that is derived from the intended scale.

Schematization on the other hand aims to remove any unnecessary details, retaining only those features that support the purpose of the map. In some ways it seeks to minimize the complexity of the map, but is constrained to maintain a functional level of detail. In contrast to generalization, schematic maps usually have some degree of distortion or stylization to support the map's intent. For schematization, simplification is typically parameterized by some low maximal complexity; the resemblance with the original map is optimized.

Generalization. An important problem in automated cartography is *generalization*. This is the process of deriving scale-appropriate maps from detailed geographic data.[2] It may refer to the manual process as performed by cartographers or to the automated process; in the context of this thesis, however, we shall use it to refer to the automated process. A representation is scale-appropriate if it provides as much details as possible, without making the details unreadable. Generalization involves a variety of different *generalization operators* [147] which describe a spatial transformation on the data. The purpose of these operators is to transform overly detailed geographic information into a scale-appropriate representation from a given map scale. Simplification is one of the important generalization operators. Figure 1.9 illustrates generalization by showing maps at different scales. At the highest scale (a), many details such as building outlines and pedestrian paths are shown. At a medium scale (b), small details such as exact building outlines have been eliminated. At a small scale (c), most details of the city have been removed, showing only major roads and an indication of land use.

[2]Most literature on map generalization is restricted to maps at a given scale. For the purpose of this thesis, we consider generalization only in this context. However, a broader definition may be posed that includes the construction of other maps.

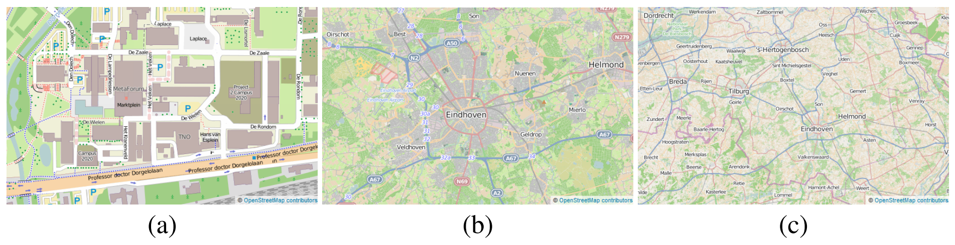


Figure 1.9 A topographic map at three different scales. Generalization makes the map appropriate for the given scale. Images of OpenStreetMap [`www.openstreetmap.org`, accessed April 2014].

Simplification. Reduction of detail is an important part of the schematization process and the generalization process. This is referred to as *simplification*. Simplification forms a central problem in automated cartography and has been studied extensively. The purpose of simplification is to reduce the number of points used to represent a geographic element, given as a sequence of points (vertices). Hence, the output of simplification is a shorter sequence of vertices that represents the geographic element. Simplification should maintain high similarity, though some precision is unavoidably lost in the process.

We distinguish two variants: *vertex-restricted* and *nonvertex-restricted* simplification (see Figure 1.10). For the former, we require that the resulting sequence uses only vertices that are also present in the original sequence. For the latter, the simplified sequence may use "new" points which are not present in the original sequence. As it has more freedom in placing the vertices, nonvertex-restricted simplification is more powerful than the vertex-restricted variant. However, it also poses more algorithmic challenges. The most well-known methods for simplification are Visvalingham-Whyatt [172], Imai-Iri [109] and Douglas-Peucker [69]; each of these is a vertex-restricted method. The simplification that occurs during $\mathcal{C}$-oriented schematization is—almost by necessity—nonvertex-restricted: a $\mathcal{C}$-oriented outline using the input vertices need not exist.

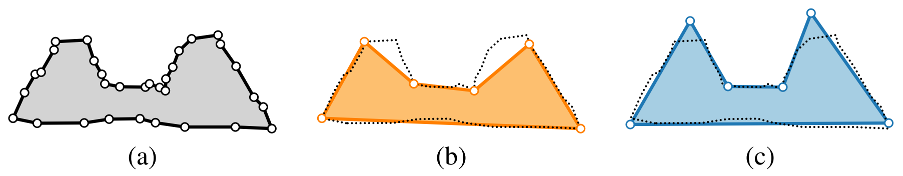


Figure 1.10 (a) Original shape with 32 points. (b) Vertex-restricted simplification with 6 vertices, computed using the Visvalingam-Whyatt algorithm [172]. (c) Nonvertex-restricted simplification with 6 vertices, computed using the algorithm of Chapter 7.

1.2 Similarity measures

To assess the quality of a schematic map, we need a way to quantify the resemblance of shapes. In fact, measuring resemblance between shapes is a fundamental problem that arises in many applications such as handwriting recognition and trajectory analysis. Regardless of the application, there is one central question: what does it mean for two shapes to be similar? The answer, however, depends on the intended application, that is, the kind of shapes that are being compared.

A *similarity measure* is a mathematical way to formally quantify resemblance. A large variety of such measures exists; each has different properties, advantages and disadvantages. Here we briefly discuss two examples of similarity measures and some general concepts; a detailed comparison in the context of $\mathcal{C}$-oriented schematization is given in Chapter 3.

A prominent measure in the mathematical literature is the *Hausdorff distance* [104]. Informally, this measure is defined as follows. Suppose we find for every point on one shape the closest point on the other and vice versa. If we measure the distance between each point and its nearest other point, the Hausdorff distance is the maximum measured distance (see Figure 1.11(a)). The higher the Hausdorff distance, the lower the similarity. In the context of curves (such as territorial outlines), this measure is often unsatisfactory as it fails to take the continuity of shapes into account; this is illustrated in Figure 1.11(b).

The *Fréchet distance* [83] is conceptually similar to the Hausdorff distance, but it is more suitable for curves and other continuous shapes such as surfaces. It pairs each point on one curve with a nearby point on the other to obtain a "matching". The Fréchet distance is the maximal distance between the pairs in the best matching. However, it also requires that this matching respects the order of the points along the curve: suppose that a point on one curve—let's call this point p—is matched to a point q on the other curve. All points before p on the first shape must be matched to points before q on the other and vice versa. By imposing this "order constraint", the Fréchet distance accounts for the continuity of the curve: it treats them as actual curves rather than as an arbitrary set of points. In particular, this requires the first and last points along the curves to be matched. The Fréchet distance in the example of Figure 1.11(c) is much greater than the Hausdorff distance. Despite the

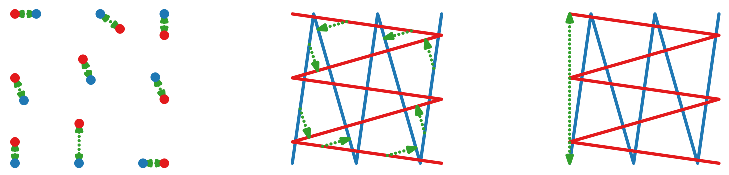

(a) (b) (c)

Figure 1.11 (a) Hausdorff distance between two sets of points, red and blue, corresponds to the longest green dotted line. (b) The Hausdorff distance between these curves is low. (c) The Fréchet distance enforces continuity in how the points are matched.

added conceptual complexity of continuity, the Fréchet distance can be computed with efficient algorithms [10]. The Fréchet distance knows many variants and has received significant attention in algorithmic research (see Section 2.4 and Chapter 4).

For cartographic schematization, similarity is reduced when shapes (e.g. countries) are displaced or rotated or have a significantly different size. Therefore, a similarity measure for schematization should take such changes into account. In other words, when shapes undergo such transformations, the measured similarity should be lower. This is the case for both the Hausdorff distance and the Fréchet distance. However, in other applications, some transformations may not affect the similarity (for that specific application). For example, with handwriting recognition, neither the exact position nor the size may be important to the letters that are written. A similarity measure that does not change under certain transformations is called *invariant* under that transformation. Common invariances include translation invariance (displacement), scale invariance (size) and rotation invariance. With rotation invariance, the two curves in Figure 1.11(b) are considered equal: rotating one by 90 degrees results in two identical curves. For schematization, similarity measures should not be invariant under any transformation.

1.3 Algorithms

In this thesis we focus on the algorithmic aspects of similarity measures and automated schematization. *Algorithms research* is part of computer science and focuses on mathematically provable techniques to solve problems. Typically, this involves the following steps. First, the high-level, typically informal problem is modeled using mathematical formalism. This model allows us to prove certain properties of the problem. In algorithms research, the focus often lies with proving correctness and time bounds. A proof of correctness provides a guarantee that the algorithm indeed computes what it should compute. A proof of time bounds relates to the execution time of an algorithm. We may prove a maximum number of steps that an algorithm needs to compute the correct answer (an upper bound), or a minimal number of steps that *any* correct algorithm has to perform (a lower bound). This is typically done in asymptotic terms depending on the complexity of the input (e.g. number of points). This allows the analysis to abstract from the influences of computer architecture and implementation details on the exact number of steps. Instead, this highlights the most important factors that dominate the execution time for large instances. Ideally, the lower and upper bound match: this means that the algorithm is asymptotically as fast as possible. A similar analysis is used to bound the memory use of an algorithm: the number of bits in the computer's memory required to perform the algorithm. Especially if the input consists of a large number of elements, such memory bounds are important: a machine has only limited memory available; memory allocation and access are often time-consuming operations.

Computational geometry. *Computational geometry* [23] is the subfield of algorithmic research that deals with problems best stated in a geometric form. That is, the problems are formulated using geometric elements, such as points, lines, curves and surfaces. Geographic features of maps are typically described by such geometric elements: problems that arise in automated cartography are inherently geometric. Hence, it is not surprising that there is a large interchange of research between computational geometry and automated cartography. This is also obvious from the numerous methods that stem from computational geometry that address problems found in automated cartography. Especially label placement [49, 115, 174] and simplification [1, 25, 27, 95, 109] have received significant attention. In terms of schematic maps, the focus lies with network schematization for transit maps [47, 126, 133]. Similarity measures for geometric shapes, such as the Fréchet distance, are naturally part of computational geometry. Therefore, techniques developed in computational geometry are essential to address the problem of $\mathcal{C}$-oriented schematization of territorial outlines.

Optimization algorithms. Many algorithmic problems are optimization problems: they ask to minimize or maximize some property of the output, given a certain set of constraints. For example, the simplification problem (Section 1.1.3) can be stated as follows: "minimize the Fréchet distance to the input (the optimization criterion) given that the simplified curve uses at most a fixed number of points (a constraint)". A high-level problem often admits more than one formulation as an optimization problem. For example, simplification can be formulated conversely as "minimize the number of points, ensuring that the simplified curve has at most a given Fréchet distance to the input". When developing an algorithm for an optimization problem, we would like to prove that it indeed computes an optimal result (proof of correctness). That is, we wish to prove that no other solution, that adheres to the constraints, has a better value for the optimization criterion.

In addition, we would like our algorithm to be efficient: the upper bound on the number of steps is a polynomial function—preferably of low degree—in the input complexity. However, there exists a large collection of problems for which it is generally believed that no efficient algorithms exist (though this remains unproven). These problems are referred to as *NP-hard* problems [86]. If an efficient algorithm cannot be found for a problem, the cause may be that it is such an NP-hard problem. Proving that a problem is indeed NP-hard then gives theoretic evidence that an efficient algorithm is unlikely to exist.

If algorithms cannot efficiently compute an optimal solution, we may turn to *approximation algorithms* [169]. Such algorithms efficiently compute a solution which is not necessarily an optimal solution. For an approximation algorithm, we prove that this computed solution is near the optimal solution (e.g. at most a factor 2 worse). If such guarantees cannot be given, we refer to the algorithm as a *heuristic algorithm*.

Another way of dealing with NP-hard problems is to aim for optimal solutions, but use techniques to reduce the execution time in practice. One such technique is *Integer Linear Programming* [53]. An integer linear program (ILP) consists of an optimization criterion and a set of constraints, all formulated as linear functions on integer variables. Because solving an ILP optimally is an NP-hard problem [86], no guarantees on efficiency can be given. However, many problems can be formulated as an ILP; efficient implementations for solving such problems have therefore received significant attention. These methods work well in practice and find an optimal solution or close-to-optimal solution for reasonably complex problem sizes. By transforming a given problem into an ILP, it is possible to leverage these implementations to solve the problem optimally.

1.4 Contributions

In this thesis we investigate automated $\mathcal{C}$-oriented schematization for territorial outlines. That is, we wish to develop algorithms that transform detailed geographic outlines into abstract but recognizable representations. Consequently, this thesis consists of three parts. In the first part we study similarity measures to quantify recognizability for schematic outlines. The second part deals with algorithms to compute a $\mathcal{C}$-oriented schematic outline from a given geographic outline. In the last part we widen the scope of our schematization algorithms and investigate their use for cartographic generalization and other styles of schematization. Before these three parts, in Chapter 2, we introduce some general concepts, notations and terminology that are used throughout this thesis. After the three parts, in Chapter 10, we identify and discuss some general open questions for automated cartographic schematization.

In addition to the contributions contained in this thesis, the author published some other results related to computing the Fréchet distance [38] and schematization in other geometric styles [66, 67, 68, 91, 92].

1.4.1 Part I—Similarity measures

Review. In Chapter 3 we review and discuss similarity measures in the context of $\mathcal{C}$-oriented schematization of territorial outlines. We give examples to show that many measures, when optimized, may have undesirable behavior. Based on this review, we choose to further investigate the Fréchet distance.

This chapter is partially based on joint work with Kevin Buchin, André van Renssen and Bettina Speckmann. Part of this work appeared in the proceedings of the 7th International Conference on Geographic Information Science [127].

Computation. In Chapter 4 we present a new algorithm to compute the Fréchet distance. Before the methods presented in this chapter, the algorithm by Alt and Godau [10] was the asymptotically fastest algorithm for nearly 20 years. Other algorithms have been developed to speed up computation under various assumptions. We provide the first asymptotic improvement that does not pose additional constraints on the input curves. Given two curves described by n vertices, our algorithm computes the Fréchet distance in $O(n^2\sqrt{\log n}(\log\log n)^{3/2})$ time. We also show that there is a discrepancy between different computational models.

This chapter is based on joint work with Kevin Buchin, Maike Buchin and Wolfgang Mulzer. This work appeared in the proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms [35].

Matchings. In Chapter 5 we seek to describe the similarity between curves, rather than distilling the similarity to a single number. To this end, we further refine the Fréchet distance. The Fréchet distance is defined by a matching between the points along both curves that minimizes the maximum distance between two matched points. Although the Fréchet distance itself yields only a number, the matchings used in its definition are a description of the similarity between the curves. We consider Fréchet matchings: matchings that result in the Fréchet distance. For two curves, there are typically many Fréchet matchings. However, some describe the similarity more accurately or more intuitively than others. We define locally correct Fréchet matchings to distinguish intuitive from unintuitive matchings. We prove that a locally correct Fréchet matching exists between any two curves and show how to compute one in $O(n^3 \log n)$ time. For the discrete variant, we provide an algorithm that runs in $O(n^2)$ time.

This chapter is based on joint work with Kevin Buchin, Maike Buchin and Bettina Speckmann. This work appeared in the proceedings of the 20th European Symposium on Algorithms [36].

1.4.2 Part II—Schematization algorithms

Global optimization. In Chapter 6 we formulate schematization as a so-called map-matching problem. *Map matching* concerns the task of finding a path or cycle in a given graph—embedded in the Euclidean plane, $\mathbb{R}^2$—that resembles a given curve. Typically, this is used to find a driven route in a road network, based on a sequence of possibly inaccurate GPS locations that together form the curve; map matching then finds the path that most likely represents to the driven route corresponding to the GPS locations (e.g. the path with the lowest Fréchet distance). This idea can also be used to compute a $\mathcal{C}$-oriented schematization of a territorial outline. First, we define a graph with edges adhering to a set $\mathcal{C}$ of orientations (the "road network"). The geographic outline represents the curve: the vertices describing the outline are interpreted as a "GPS sequence". A cycle in the graph with low Fréchet distance to the curve then corresponds to a $\mathcal{C}$-oriented schematization. Efficient algorithms exist to solve the map-matching problem with the Fréchet distance [9]. However, these compute a path that may visit vertices and edges of the graph more than once: the resulting schematization may be topologically invalid. Hence, we investigate the problem with a simplicity constraint, that is, vertices of the graph may occur at most once along the cycle. We prove that this problem is NP-hard: an efficient algorithm is unlikely to exist. This proof in fact extends to show that it is even NP-hard to compute a path or cycle that approximates the Fréchet distance of the optimal solution. Finally, we show that the problem admits an ILP formulation (that is, it can be solved via an Integer Linear Program). The interval graph involved in developing this formulation is also used to define a brute-force algorithm that can solve small instances in practice. We discuss some experimental results and compare the performance of the ILP and the brute-force algorithm.

Local operations. In Chapter 7 we investigate the effectiveness of a heuristic schematization algorithm. This algorithm consists of two steps. In the first step, the outline is converted into a similar outline that uses line segments that adhere to a given orientation set $\mathcal{C}$. In the second step, the outline is simplified by iteratively performing local operations without introducing new orientations. Throughout these steps, we ensure that the topology remains valid and that the area of the outline is preserved. The algorithm can also simplify an outline by omitting the first step. We prove that, regardless of $\mathcal{C}$, this algorithm can reduce the complexity of any simple polygon until it is convex. We provide experimental results to show that this algorithm produces results that capture the most salient features for a variety of outlines and orientation sets.

This chapter is based on joint work with Kevin Buchin, André van Renssen and Bettina Speckmann. Part of this work appeared in the proceedings of the 7th International Conference on Geographic Information Science [127] and the proceedings of the 19th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems [42].

1.4.3 Part III—Widening the scope

Generalization. In Chapter 8 we widen the scope of the schematization algorithm presented in Chapter 7. To this end, we apply this algorithm to generalize buildings and urban areas. We present several extensions such that the algorithm can be used for wall squaring as well as simplification and aggregation of buildings. We present experiments that indicate that the algorithm works well and is comparatively fast in this application.

This chapter is based on joint work with Kevin Buchin and Bettina Speckmann. This work appeared in the proceedings of the 19th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems [42].

Other geometric styles. In Chapter 9 we consider the use of the map-matching techniques presented in Chapter 6 for different geometric styles. Using straightforward additions to these techniques, we argue that it can also be used for parallelism and curved schematization. Furthermore, we introduce a new geometric style: *isothetic schematization*. In an isothetic schematization, all used line segments must coincide with a line that passes through one of a small number of points. These extended techniques provide interesting schematic outlines and illustrate the flexibility of the map-matching approach.

Chapter 2

Preliminaries

In this chapter we introduce and formalize the terminology and notation that is used throughout this thesis. Though many of the concepts can also be applied to higher-dimensional space, we assume that all geometric objects reside in the two-dimensional Euclidean plane ($\mathbb{R}^2$). Every point $p = (x, y)$ in this plane is uniquely defined by two coordinates, x and y. The Euclidean distance between two points, $p = (x, y)$ and $q = (a, b)$, is denoted by $\|p - q\| = \sqrt{(x-a)^2 + (y-b)^2}$.

2.1 Polygonal curves and polygons

Polygonal curves. A *polygonal curve* P is defined by a sequence $\langle p_0, ..., p_{n-1}\rangle$ of *vertices* (points in $\mathbb{R}^2$). Its consecutive vertices are connected by line segments; these line segments are referred to as *edges*. We distinguish two types of polygonal curves, depending on whether they are cyclic (*closed*) or not (*open*). That is, an open polygonal curve has a distinct startpoint and endpoint: vertices p_0 and p_{n-1} in the sequence. In a closed polygonal curve, p_0 and p_{n-1} are also connected by an edge; the sequence is cyclic and

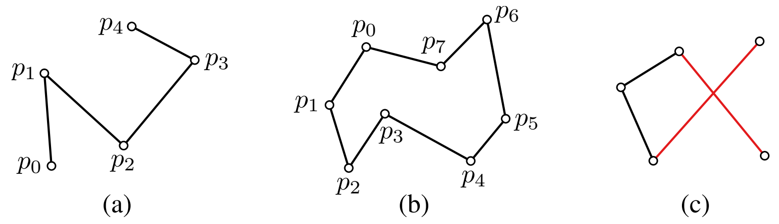


Figure 2.1 (a) A simple open curve of complexity 4. (b) A simple closed curve of complexity 8. (c) Example of a nonsimple open curve: the red edges intersect.

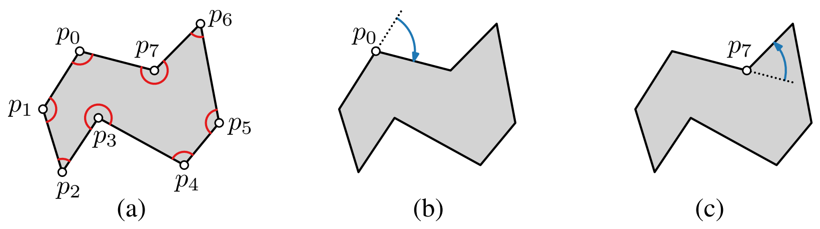


Figure 2.2 (a) A polygon (gray area) and its boundary given by black lines. Its interior angles are indicated in red; it has two reflex vertices, p_3 and p_7. (b) A positive exterior angle at convex vertex p_0. (c) A negative exterior angle at reflex vertex p_7.

can be shifted to turn any vertex into "p_0". Figure 2.1(a–b) illustrates these terms. For a closed polygonal curve, we treat the vertices circularly, that is, $p_i = p_{i \bmod n}$. Throughout this thesis, we shall refer to polygonal curves simply as *curves*. Unless indicated otherwise, curves refer to open curves.

The *complexity* of a curve is determined by its number of edges. Thus, an open curve of complexity n has $n + 1$ vertices; a closed curve of complexity n has n vertices. For a curve of complexity n, its edges are given by the line segments connecting p_i and p_{i+1} for $i \in \{0, \ldots, n - 1\}$. We call a polygonal curve *simple* if no edges intersect, except at common endpoints. Figure 2.1 illustrates two simple and one nonsimple curve.

Polygons. A *polygon* P represents the area that is enclosed by a closed curve. In other words, the boundary of a polygon, denoted by ∂P, is a closed curve. A polygon is simple if its boundary is simple; unless indicated otherwise, we implicitly assume that polygons are simple. The complexity of a polygon is the number of edges that are used to describe its boundary. A polygon of complexity n is described by n vertices.

A polygon has two types of vertices: a vertex is called *convex* if the angle inside the polygon boundary between its two incident edges is at most π, and it is called *reflex* otherwise. The *exterior angle* of a vertex is defined as the angle between one edge and the extension of the other. The angle is negative if and only if the vertex is reflex. These concepts are illustrated in Figure 2.2. The sum of all exterior angles of a simple polygon is always equal to 2π.

2.2 Graphs and planar subdivisions

Graphs. A *graph* $G = (V, E)$ is defined by its *vertices* V and *edges* E. The vertices are elements that are connected with other vertices via the edges. Thus, an edge is defined as a pair of vertices and E is a subset of $V \times V$. An edge (u, v) is *incident* to its endpoints u and v. For a vertex v, we refer to the edges that end at v as the incident edges. The number of incident edges is referred to as the *degree* of a vertex. Vertices of degree 3 or higher are referred to as *junctions*. The *complexity* of a graph is its number of edges.

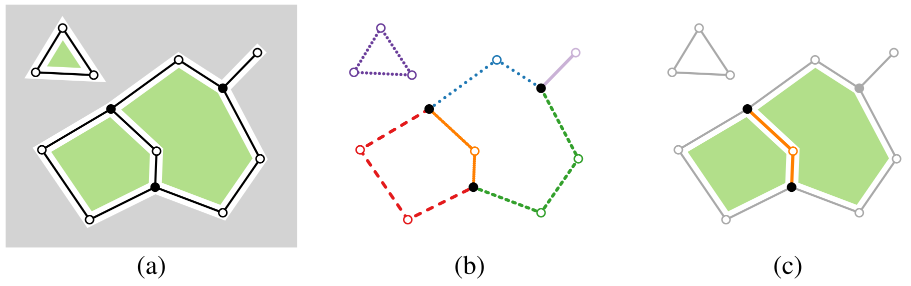


Figure 2.3 (a) A plane graph. Vertices are given as dots, the solid dots represent junctions. Edges are indicated with black line segments. It has three bounded faces (green); the unbounded face is given in gray. (b) The disjoint borders in the graph, each indicated with a different color or style. (c) A border and its two incident faces.

A *path* in a graph is a sequence of vertices $\langle v_0, \ldots, v_k \rangle$ such that $(v_i, v_{i+1}) \in E$ for all $i \in \{0, \ldots, k-1\}$. A path is called *simple* if each vertex occurs at most once in the sequence. A *cycle* is a path that starts and ends at the same vertex (i.e., $v_0 = v_k$). A cycle is *simple* if each vertex occurs at most once except for the common endpoint $v_0 = v_k$ which occurs exactly twice. Like the vertices of a polygon, we treat the vertices of a cycle circularly: vertex v_i is used as a shorthand for $v_{i \bmod k}$.

We call a graph *embedded* if each vertex v is a point in $\mathbb{R}^2$ and each edge (u, v) is the line segment between vertices u and v. A *plane* graph is an embedded graph in which no two vertices coincide on the same point and no two edges intersect (except at a common endpoint). We call a region of a plane graph enclosed by edges a *face*. Most faces are *bounded*, having finite area. However, each plane graph has exactly one *unbounded* face with infinite area. A *border* is a maximal path in a plane graph such that all vertices (except possibly the first and last) have degree 2. A graph is partitioned into a set of disjoint borders. A border has exactly two incident faces, one on each side; in some cases, this may actually be the same face. Two faces are said to have a border in common if they are both incident on the border. We call a pair of faces *adjacent* if they have one or more borders in common. A plane graph and the corresponding terminology are illustrated in Figure 2.3.

Subdivisions. We use the term (planar) *subdivision* to refer to a plane graph. In particular, we use it to refer to objects that represent regional elements such as territorial outlines. The bounded faces in a subdivision thus correspond to territories. For territorial outlines, we may usually assume that vertices of degree 0 or 1 do not occur. Moreover, each border typically has two distinct incident faces. However, these are not general restrictions for a subdivision.

2.3 Formalizing automated $\mathcal{C}$-oriented schematization

In this section we formalize the automated $\mathcal{C}$-oriented schematization problem for territorial outlines. The input, geographically detailed territorial outlines, are given as a subdivision. The required output, the schematization, must also be a subdivision that adheres to the following four criteria (see Chapter 1.1.2).

(1) Schematic outlines should use few line segments.
(2) All line segments should be oriented according to $\mathcal{C}$.
(3) The geographic relations between regions of the schematic outlines are equivalent to those of the geographic regions (correct *topology*).
(4) Schematic outlines *resemble* the corresponding geographic outlines.

To formalize the problem, we need to define these criteria precisely. The first criterion is defined rather easily: we use the complexity of the resulting subdivision. The second and third criterion can be defined straightforwardly, though are slightly more involved. We therefore discuss them in the upcoming sections. A satisfying formalization of the last criterion is more complicated; we postpone a detailed discussion of this to Chapter 3.

2.3.1 Orientations

The *orientation* of a line segment is its counterclockwise angle in $[0, \pi)$ with respect to the horizontal axis. Given a discrete set $\mathcal{C}$ of orientations, a line segment is $\mathcal{C}$*-oriented* if its orientation is contained in $\mathcal{C}$. A subdivision is $\mathcal{C}$-oriented if all edges are $\mathcal{C}$-oriented. For a nontrivial $\mathcal{C}$-oriented subdivision, $\mathcal{C}$ must contain at least two distinct orientations.

An orientation set $\mathcal{C}$ is described by a set of angles. For example, $\mathcal{C} = \{0, \frac{\pi}{4}, \frac{\pi}{2}\}$ describes a set with a horizontal, vertical, and one diagonal orientation (from bottomleft to topright). A regular orientation set contains orientations that are evenly spaced. Such a set is characterized by some initial angle β and the number of orientations c. The set of orientations is then $\{\beta + i \cdot \frac{\pi}{c} \mid 0 \leqslant i < c\}$. Typically, β is either zero (the first orientation is horizontal) or $\frac{\pi}{2}$ (the first orientation is vertical). We refer to such sets as horizontal

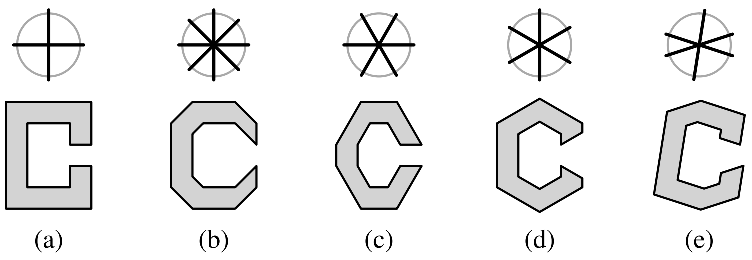


Figure 2.4 Four common regular orientation sets and one irregular set: (a) rectilinear, $\mathcal{C}_2$; (b) octilinear, $\mathcal{C}_4$; (c) hexilinear, $\mathcal{C}_3$; (d) hexilinear, $\mathcal{C}_3(\frac{\pi}{2})$; (e) irregular, $\mathcal{C} = \{\frac{2\pi}{20}, \frac{9\pi}{20}, \frac{18\pi}{20}\}$.

and vertical respectively. These sets are different only for odd values of c. We use $\mathcal{C}_c(\beta)$ to denote a regular set with c orientations and initial angle β. If β is 0, we abbreviate this to $\mathcal{C}_c$. Common orientation sets include rectilinear ($\mathcal{C}_2$), hexilinear ($\mathcal{C}_3$ or $\mathcal{C}_3(\frac{\pi}{2})$), and octilinear ($\mathcal{C}_4$) orientations. Figure 2.4 illustrates some orientation sets.

2.3.2 Topology for subdivisions

A subdivision, which is computed as a result of schematization, must have the same topology as the input subdivision to avoid large interferences with a user's mental map. Intuitively, this means that the adjacencies between the faces of the schematic subdivision are identical to those in the input. The borders that separate the faces may change in detail or shape, but no structural changes in the adjacencies occur. We formalize this as follows.

First, consider a face f in a subdivision S. The boundaries that delineate the region of f consist of one or more *cycles* of borders in S. A cycle that encloses f is referred to as an *outer cycle*. Conversely, if f encloses the cycle, it is an *inner cycle*. If face f is bounded, it has exactly one outer cycle and zero or more inner cycles. If face f is unbounded, it has no outer cycle and one or more inner cycles. Figure 2.5(a) shows a face f with an inner and outer cycle.

An inner or outer cycle of face f consists of one or more borders in S; each of these has a second incident face which is adjacent to f. The sequence that describes these adjacent faces in counterclockwise order is referred to as the *adjacency order* of that cycle. In the case of a vertex of degree 1, the border is considered to occur twice (consecutively) along the cycle. Figure 2.5(b) illustrates an adjacency order. Note that a face may be adjacent to itself. In case of subdivisions that represent territorial outlines, this usually does not occur: if a face is adjacent to itself, then the border in the subdivision does not correspond to a boundary that separates the territory from another.

Consider two subdivisions S and S'. We assume that a bijective function $\mathcal{F}$ maps the faces of S to the faces of S'. This function describes the correspondence between the faces. In other words, we may consider the faces to have a unique label in S; this label occurs exactly once in S'. Subdivisions S and S' are *topologically equivalent* if the following conditions are met.

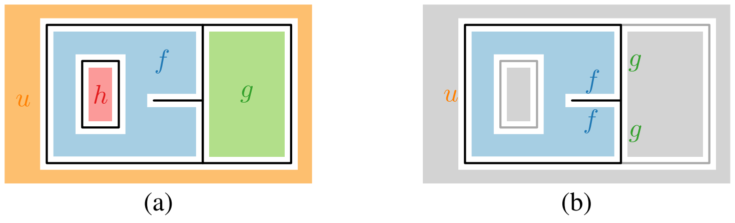


Figure 2.5 (a) A subdivision with 3 bounded faces and one unbounded face. Face f has one inner cycle as it encloses face h. (b) The adjacency order of the outer cycle of f is $\langle g, f, f, g, u \rangle$.

- The unbounded faces correspond according to $\mathcal{F}$. If face f is the unbounded face of S, then $\mathcal{F}(f)$ is the unbounded face of S'.
- For each bounded face, the adjacency order of the outer cycle correspond according to $\mathcal{F}$. Let $\langle a_0, \ldots, a_{k-1} \rangle$ denote the adjacency order of the outer cycle of a face f. Let $\langle a'_0, \ldots, a'_{k'-1} \rangle$ denote the adjacency order of the outer cycle of $\mathcal{F}(f)$. It holds that $k = k'$ and $\mathcal{F}(a_i) = a'_{i+d \bmod k}$ for all $i \in \{1, \ldots, k-1\}$ and some $d \in \{0, \ldots, k-1\}$.
- For all (bounded and unbounded) faces, each inner cycle has a corresponding inner cycle such that the adjacency orders correspond according to $\mathcal{F}$. Let $\langle a_0, \ldots, a_{k-1} \rangle$ denote the adjacency order of an inner cycle of a face f. There is an inner cycle of $\mathcal{F}(f)$ with adjacency order $\langle a'_0, \ldots, a'_{k'-1} \rangle$ such that $k = k'$ and $\mathcal{F}(a_i) = a'_{i+d \bmod k}$ for all $i \in \{1, \ldots, k-1\}$ and some $d \in \{0, \ldots, k-1\}$.

The third condition requires that some inner cycle in face $\mathcal{F}(f)$ exists to match an inner cycle of face f. Due to the unique labels of faces, this in fact requires an exact one-to-one correspondence. Figure 2.6 illustrates three subdivisions, two of which have equivalent topology. As illustrated in (c), subdivisions that display symmetries in their adjacencies can sometimes be structurally “relabeled” while maintaining topology. The topology is indeed the same in terms of adjacencies. To address this issue, other constraints can be considered, such as directional constraints [41] or orthogonal-order constraints [88]. Also, measuring similarity may avoid these problems.

A schematization algorithm is *topologically safe* if it guarantees that the input and output subdivision are topologically equivalent. As topological equivalence is transitive, it is sufficient to ensure that each iteration in an iterative algorithm maintains the correct topology. For this topological equivalence, we require two subdivisions which are plane graphs: intersections may therefore not be introduced. Moreover, it implies that there is a one-to-one correspondence between the borders as well: no borders can be fully removed.

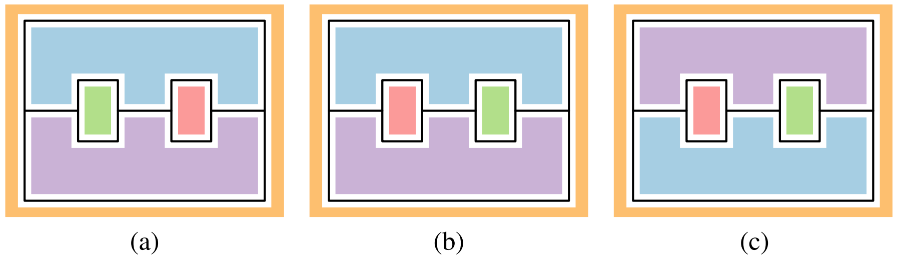


Figure 2.6 Three subdivisions; corresponding faces according to $\mathcal{F}$ indicated with equal colors. (a) A subdivision with 4 bounded faces. (b) A subdivision with topology different from (a) as the red and green face occur in different order along the outer cycle of the blue and purple face. (c) A subdivision that is topologically equivalent to (a), all adjacency orders match.

2.4 The Fréchet distance

The Fréchet distance is a similarity measure to compute the resemblance between two polygonal curves. Due to our findings in Chapter 3, this distance and its computation play an important role in later chapters. Therefore, we provide its definition and several variants here and briefly discuss basic concepts of computation.

2.4.1 Definition

Let P and Q be two polygonal curves, defined by vertices $\langle p_0, \ldots, p_m \rangle$ and $\langle q_0, \ldots, q_n \rangle$. We interpret P as a continuous function $P\colon [0,m] \to \mathbb{R}^2$ such that $P(i+\lambda) = (1-\lambda)p_i + \lambda p_{i+1}$ for $i \in \{0, \ldots, m-1\}$ and $\lambda \in [0,1]$. In particular, this means that $P(i) = p_i$ for $i \in \{0, \ldots, m\}$. Similarly, we interpret Q as a continuous function $Q\colon [0,n] \to \mathbb{R}^2$. An orientation-preserving homeomorphism between P and Q is a continuous function $\psi\colon [0,m] \to [0,n]$ with a continuous inverse, such that $\psi(0) = 0$ and $\psi(m) = n$. Such a homeomorphism matches each point of P to a unique point on Q and vice versa. Let Ψ be the set of all orientation-preserving homeomorphisms between P and Q. The *Fréchet distance* between P and Q is defined as

$$d_{\mathrm{F}}(P,Q) \equiv \inf_{\psi \in \Psi} \max_{x \in [0,m]} \|P(x) - Q(\psi(x))\|,$$

where $\|\cdot\|$ denotes the Euclidean distance. A low Fréchet distance indicates a high similarity between the two curves.

A reparametrization of curve P is a continuous nondecreasing function $\sigma\colon [0,1] \to [0,m]$, such that $\sigma(0) = 0$ and $\sigma(1) = m$. Similarly, a reparametrization of Q is a function $\theta\colon [0,1] \to [0,n]$. A *matching* $\mu = (\sigma, \theta)$ between P and Q is a pair of reparametrizations; this describes an orientation-preserving homeomorphism between P and Q or a limit thereof. Using $\mathcal{M}$ to denote the set of all matchings, we can define the Fréchet distance also as

$$d_{\mathrm{F}}(P,Q) \equiv \min_{(\sigma,\theta) \in \mathcal{M}} \max_{x \in [0,1]} \|P(\sigma(x)) - Q(\theta(x))\|.$$

We call a matching μ that results in the Fréchet distance a *Fréchet matching*. An example is given in Figure 2.7(a). Note that this matching is often not unique: typically, many matchings have the same maximal distance. In Chapter 5 we investigate ways of distinguishing between different Fréchet matchings.

Variants. The definition above is stated for open polygonal curves in $\mathbb{R}^2$; the definition can be straightforwardly generalized for closed curves [10, 121], nonpolygonal curves [152] and curves embedded in higher dimensional space [10, 38]. Also, the definition can be extended for surfaces [37, 40, 60, 90].

There are two frequently used variants of the Fréchet distance: the *weak* Fréchet distance [10] and the *discrete* Fréchet distance [3, 75]. The weak Fréchet distance is characterized by no longer requiring a one-to-one matching. In this context, a reparametrization is any continuous (not necessarily nondecreasing) function $\sigma\colon [0,1] \to [0,m]$ with

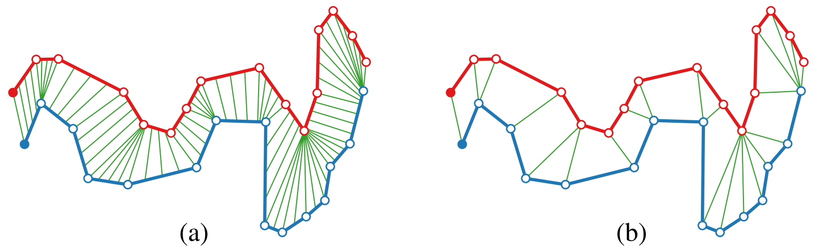


Figure 2.7 (a) A Fréchet matching for two curves. Green lines represent a sample of the continuous matching. (b) A discrete Fréchet matching matches only the vertices.

$\sigma(0) = 0$ and $\sigma(1) = m$. The discrete Fréchet distance changes which points are matched: the continuous Fréchet distance matches all points along the two curves, whereas the discrete Fréchet distance matches only the vertices. Note that a discrete matching is not a one-to-one matching, one vertex may correspond to multiple consecutive vertices on the other curve. This is illustrated in Figure 2.7(b). There are also a number of variants that modify the matching in other ways, such as speed limits [32, 120] and partial matchings [10, 39, 70, 121]. These variants are briefly discussed in Chapter 5.

The Fréchet distance uses the Euclidean distance between two points. Other distance measures have also been used, such as polyhedral distance functions [38] or geodesic and link distances [74].

2.4.2 Computation

Algorithmic study of the Fréchet distance was initiated by Alt and Godau [10]. Their techniques have been widely used in the many strands of research involving the Fréchet distance that followed. The results presented in Chapter 4 and Chapter 5 also use these techniques. Therefore, we summarize their results here.

Decision and search. Alt and Godau [10] solved the computation of the Fréchet distance between two polygonal curves, P and Q, in two steps. First, they developed an algorithm to decide whether $d_{\mathrm{F}}(P, Q) \leqslant \varepsilon$ for some given value of $\varepsilon \geqslant 0$. Then, they showed how to use this algorithm to compute the Fréchet distance by efficiently searching through possible values of ε. Many of the later algorithms, including the algorithm presented in Chapter 4, adhere to this "decision-and-search" paradigm. The only algorithm—to exactly compute the Fréchet distance—that breaks from this tradition is given by Buchin *et al.* [38].

Decision. In order to decide whether $d_{\mathrm{F}}(P, Q) \leqslant \varepsilon$ holds, Alt and Godau [10] introduced the *free-space diagram*. We denote the free-space diagram of curves P and Q by $\mathrm{FSD}(P, Q)$. This diagram represents the joint parameter space of P and Q (interpreted as functions): the domain of $\mathrm{FSD}(P, Q)$ is $[0, m] \times [0, n]$. Every point (s, t)

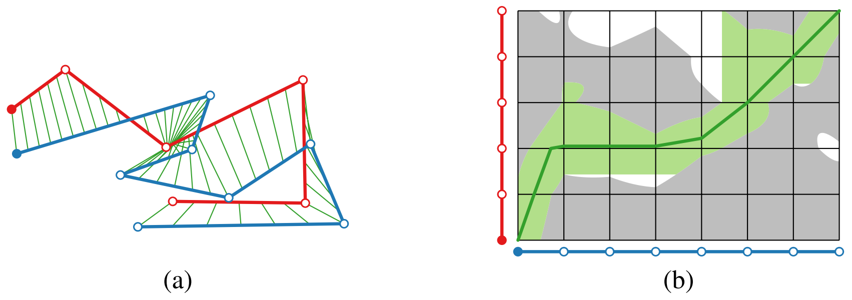


Figure 2.8 (a) Two curves with a Fréchet matching. (b) The corresponding free-space diagram. The reachable space is green; the unreachable free space is white; the green line represents a Fréchet matching.

in the diagram corresponds to two points: $P(s)$ and $Q(t)$. A point (s,t) in the free-space diagram is called *free* if $\|P(s) - Q(t)\| \leqslant \varepsilon$. The union of all free points is referred to as the *free space*; we denote the free space by $\mathrm{free}(P,Q) = \{(s,t) \mid (s,t) \in [0,m] \times [0,n] \wedge \|P(s) - Q(t)\| \leqslant \varepsilon\}$, using ε implicitly.

A homeomorphism between P and Q corresponds to a strictly bimonotone path from $(0,0)$ to (m,n) in the joint parameter space. A matching, possibly being the limit of a homeomorphism, thus corresponds to a (not necessarily strict) bimonotone path. Thus, $d_{\mathrm{F}}(P,Q) \leqslant \varepsilon$ if and only if there is a bimonotone path through the free space. Generalizing this notion, a point (s,t) is *reachable* if there is a bimonotone path from $(0,0)$ to (s,t) in the free space. Evidently, the corresponding *reachable space* is a subset of the free space. We denote this space by $\mathrm{reach}(P,Q)$. To decide whether the Fréchet distance is at most ε, we must therefore decide whether $(m,n) \in \mathrm{reach}(P,Q)$. An example of a free-space diagram and its corresponding free and reachable space is given in Figure 2.8.

The vertices of P and Q partition the free-space diagram into square *cells*, each representing a pair of edges. There are thus m columns and n rows of cells in the diagram. The cell in column i and row j represents the subdomain $[i,i+1] \times [j,j+1]$ of $\mathrm{FSD}(P,Q)$, for $i \in \{0,\ldots,m-1\}$ and $j \in \{0,\ldots,n-1\}$; we denote this cell by $C(i,j)$. Alt and Godau [10] showed that within one cell the free space is a convex region. Hence, it is sufficient to consider only the *cell boundaries*; each boundary corresponds to a vertex of one curve and an edge of the other.

For a cell $C(i,j)$, we denote its left boundary by $L_{i,j}$ and its bottom boundary by $B_{i,j}$. The right side of a cell is the left side of the cell in the next column. Thus, the right side is denoted by $L_{i+1,j}$. Similarly, the top side is denoted by $L_{i,j+1}$. We define a *door* of a vertical side $L_{i,j}$ as the points of that side that are in the free space, i.e., as $L_{i,j} \cap \mathrm{free}(P,Q)$. We denote the door of a side $L_{i,j}$ by $L^{\mathrm{F}}_{i,j}$. The door is *open* if the intersection is nonempty; it is *closed* otherwise. A *reach-door* $L^{\mathrm{R}}_{i,j}$ is the subset of the door that is reachable: $L^{\mathrm{R}}_{i,j} = L^{\mathrm{F}}_{i,j} \cap \mathrm{reach}(P,Q)$. Again, a reach-door is open if there are indeed reachable points on the corresponding side and closed otherwise. Similarly, we define the door $B^{\mathrm{F}}_{i,j}$ and reach-door $B^{\mathrm{R}}_{i,j}$ of a horizontal boundary $B_{i,j}$.

To compute $L^{\mathrm{R}}_{i+1,j}$, it suffices to know $L^{\mathrm{R}}_{i,j}$ and $B^{\mathrm{R}}_{i,j}$. To this end, we require only three cases, as illustrated in Figure 2.9.

(a) If $B^{\mathrm{R}}_{i,j}$ is open, then all points in $L^{\mathrm{F}}_{i+1,j}$ are reachable: thus $L^{\mathrm{R}}_{i+1,j} = L^{\mathrm{F}}_{i+1,j}$.
(b) If $B^{\mathrm{R}}_{i,j}$ is closed but $L^{\mathrm{R}}_{i,j}$ is open, then any point on $L^{\mathrm{F}}_{i+1,j}$ that is above the lower endpoint of $L^{\mathrm{R}}_{i,j}$ is reachable. Note that, if this lower endpoint is above the upper endpoint of $L^{\mathrm{F}}_{i+1,j}$ or $L^{\mathrm{F}}_{i+1,j}$ is closed, then $L^{\mathrm{R}}_{i+1,j}$ is closed.
(c) If both doors are closed, then $L_{i+1,j}$ cannot be reached and thus is closed as well.

Similarly, we can compute $B^{\mathrm{R}}_{i,j+1}$ from $L^{\mathrm{R}}_{i,j}$ and $B^{\mathrm{R}}_{i,j}$ This leads to an $O(mn)$-time dynamic program to propagate the reachability information from cell to cell.

Search. To compute the Fréchet distance, we find the smallest value of ε such that (m, n) is in the reachable space of the free-space diagram. Alt and Godau [10] argued that the Fréchet distance is one of a set of *critical values*. Conceptually, as ε is increased from zero, the free space expands. At certain *critical events*, a structural change of the free space occurs. Alt and Godau distinguish three of such events, defining the critical values (refer to Figure 2.10 for illustrations of these events).

(A) Point $(0, 0)$ or (m, n) becomes part of the free space; the associated critical values are $\|P(0) - Q(0)\|$ and $\|P(m) - Q(n)\|$ respectively.
(B) A door opens a cell boundary, creating a passage between two cells; the associated critical value is the minimal distance between the vertex and edge corresponding to the boundary.
(C) A reach-door opens (from case (b) in the computation), creating a new passage from one cell boundary to another within the same row or column. This corresponds to two vertices of one curve and an edge of the other. The associated critical value is the distance from one of the vertices to the intersection of the edge and the bisector of the vertices (if this intersection exists).

There are two critical events of type A, $m(n+1)+n(m+1)$ events of type B, and $nm(m-1)/2 + mn(n-1)/2 = mn(m+n)/2 - 1$ events of type C. By using parametric search on the critical values arising from type-C events, the Fréchet distance can be computed in $O(mn \log mn)$ time [10].

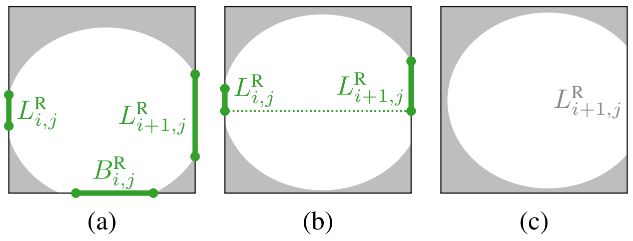


Figure 2.9 The three cases for computing $L^{\mathrm{R}}_{i+1,j}$. (a) If $B^{\mathrm{R}}_{i,j}$ is open, the complete right door is reachable. (b) If $L^{\mathrm{R}}_{i,j}$ is open but $B^{\mathrm{R}}_{i,j}$ is closed, the lower endpoint of $L^{\mathrm{R}}_{i,j}$ determines the lowest reachable point on $L^{\mathrm{R}}_{i+1,j}$. (c) Otherwise, neither $L^{\mathrm{R}}_{i,j}$ nor $B^{\mathrm{R}}_{i,j}$ is open; the right door is not reachable either.

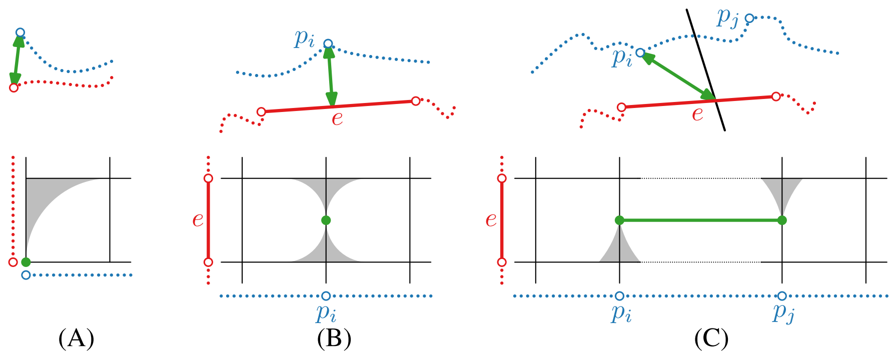


Figure 2.10 Three critical events for computing the Fréchet distance in which a structural change of the free space occurs. At the top, the situation of the curves is illustrated; at the bottom, the situation in the free-space diagram. (A) Distance between the endpoints. (B) Minimal distance between vertex p_i and edge e: a passage between two cells. (C) Distance between vertex p_i and the intersection of edge e and the bisector p_i and p_j. The bisector is indicated with a black line. This event corresponds to a passage within a row.

Part I

Similarity Measures

Chapter 3

A Review of Similarity Measures

To judge the quality of a schematic outline, we wish to compare it to its geographic outline and quantify their resemblance. In other words, we wish to find a similarity measure that is suitable for $\mathcal{C}$-oriented schematization. A number of similarity measures exists, each with different properties. In this chapter we review various similarity measures based on their suitability for schematization. These similarity measures quantify the dissimilarity: the higher the computed measure the less similar the outlines. Hence, we refer to these as distance measures: a low distance implies a high similarity. We would like to model schematization as an optimization problem in which the similarity is optimized under a set of constraints. The resulting outline should then be the best schematization.

In this chapter we refer to the geographic and schematic outline simply as the input and solution respectively. To evaluate similarity measures that can be used as an optimization criterion for schematization, we assume the following four constraints:

(i) both the input and the solution are simple polygons;
(ii) the complexity of the solution is bounded by a constant;
(iii) the solution is a rectilinear polygon ($\mathcal{C}_2$);
(iv) the area of the solution is equivalent to the area of the input.

Under these constraints we consider whether the optimal solution according to a distance measure always corresponds to the intuitively best schematization. We show that there is a polygon with higher distance (for each considered distance measure) that captures the shape of the input significantly better. Hence, while it is desirable that the solution has a low distance to the input, minimizing the distance does not ensure the best shape.

Note that the third constraint, rectilinearity, is seemingly one of the strictest $\mathcal{C}$-oriented constraints. However, we note that our arguments and examples in this chapter generalize to other orientation sets. That is, similar arguments can be made for other choices for $\mathcal{C}$. The fourth constraint does not arise from the schematization criteria. This constraint is used to develop a schematization algorithm in Chapter 7. Hence, our rationale here applies to this situation, where the input and solution must have equivalent area. Again, the arguments and examples do not rely on this constraint.

3.1 Fréchet distance

The *Fréchet distance* is defined in Section 2.4. We observe that it is determined by a maximum; it is thus rather sensitive to outliers. Consider the example in Figure 3.1. The input is an 8-sided polygon that we wish to schematize with a rectangle. The solution with the smallest Fréchet distance tries to approximate the narrow strip, while "ignoring" most of the polygon. As a result, the rectangle with optimal distance has a much higher width than height. However, the majority of the input polygon shows a rectangle that has much lower width than height. Note that the strip can be arbitrarily narrow without significantly changing the result with minimal Fréchet distance.

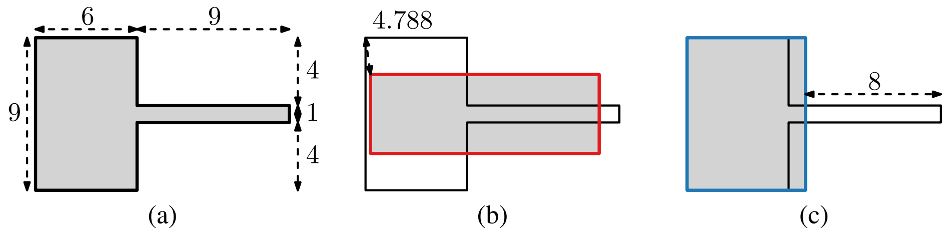


Figure 3.1 An 8-sided polygon (a) and two 4-sided schematizations: with minimal Fréchet distance (b), and with better shape (c).

As mentioned in Chapter 2, variants of the Fréchet distance exist including the weak and discrete Fréchet distance. However, these variants remain sensitive to outliers. To alleviate this problem, average and integral Fréchet distances have been considered [44]: all distances are taken into account rather than only the maximal distance. Buchin [44] describes and compares multiple intuitive definitions of the average Fréchet distance. However, all variants have disadvantages [44]. Normalization is one of the problems that arises. There are two main candidates: the length of one of the polygons or the length of the matching. However, as argued by Buchin [44], both yield unintuitive results. Below, we briefly explain the issues that she identified and relate them to schematization.

Consider two curves P and Q. A normalization that uses the length of P may "skip" certain parts of Q by matching the part to a single point of P (technically speaking, an infinitesimally small interval). This decreases the "weight" of the part; as a result, the average or integral Fréchet distance essentially does not account for this part. For schematization, this is undesirable as, for example, a large peninsula may be not accounted for by the similarity measure. This is illustrated in Figure 3.2(b).

The length of the matching—the length of the path in the free-space diagram—is problematic as it penalizes long matchings for the integral Fréchet distance. For example, in a schematization of Norway, the shoreline may be much shorter than the geographic shoreline due to the fjords (see Figure 3.3). However, the border with Sweden may stay roughly the same length and may be relatively closer to the geographic border. Hence, a good matching in this scenario may be comparatively long. For the average variant, this normalization may result in unintuitive matchings. By artificially increasing the length

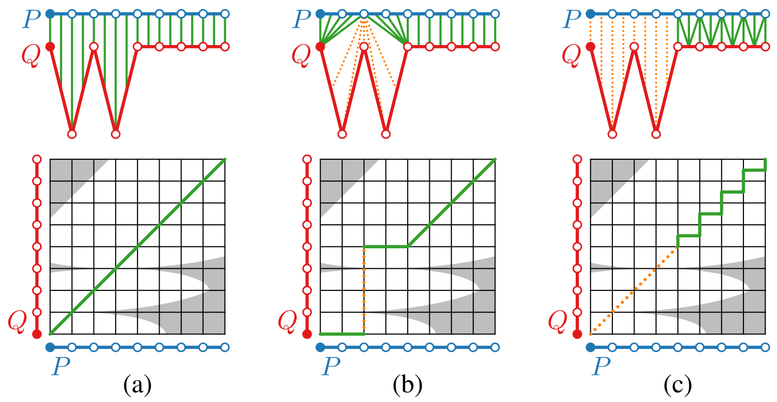


Figure 3.2 (a) An intuitive matching between P and Q. (b) Normalization according to P allows us to neglect certain parts (orange). (c) Normalization according to the matching allows us to decrease the influence of parts (orange).

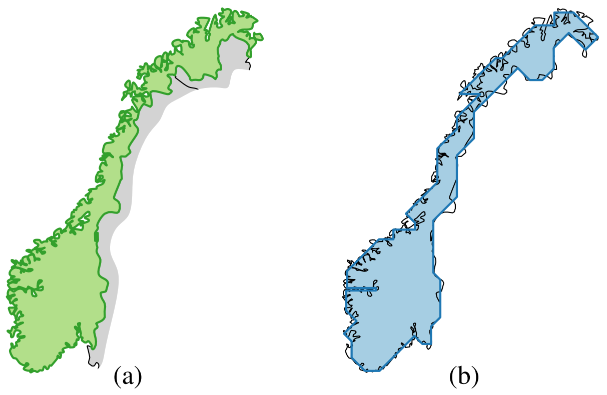


Figure 3.3 (a) The shoreline of Norway is significantly longer than its borders with adjacent countries. (b) A schematic outline with 50 edges for Norway, computed using the techniques of Chapter 7. The shoreline has been shortened much more than the country borders.

of a matching where the polygons are close, we increase the "weight" of those parts. This results in underestimating the influence of large distances elsewhere, such as at a peninsula; this is illustrated in Figure 3.2(c).

In the above we assumed that the Fréchet distance considers only the boundary of a polygon. It is also possible to consider homeomorphisms defined over the interior of the polygon. As Buchin *et al.* [40] show, the difference between the boundary and interior variant may be arbitrarily large. However, the outlier sensitivity remains.

3.2 Symmetric difference

The *symmetric difference* between two polygons P and Q is defined as the total area that is covered by one polygon but not by the other: it is exactly the area in which they differ from each other. Interpreting P and Q as the set of all points enclosed by the polygon, the symmetric difference is defined as

$$|P \cup Q| - |P \cap Q|.$$

Unlike the Fréchet distance, the symmetric difference does not consider the continuity of the curves. That is, parts that are covered by both may belong to different conceptual parts of the polygons. Consider the example in Figure 3.4. The input is a 12-sided polygon which we would like to schematize with an 8-sided rectilinear polygon. The schematization with minimal symmetric difference loses the vertical axis of symmetry and has a "C-shape" instead of a "U-shape".

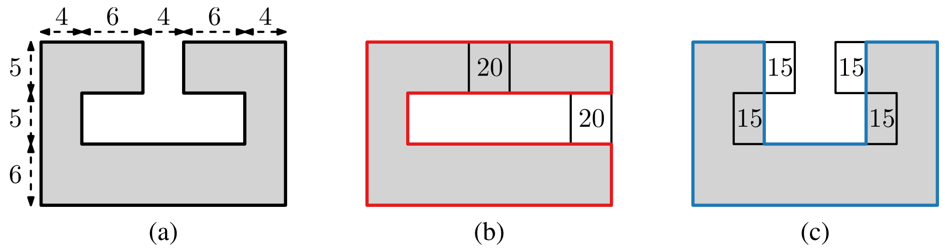


Figure 3.4 A polygon (a) and two 8-sided schematizations: with minimal symmetric difference (b), and with better shape (c).

3.3 Hausdorff distance

The *Hausdorff distance* measures the distance between two compact subsets, P and Q, of a metric space. It finds for each element in P the closest element in Q, and vice versa, and then takes the maximum of the resulting pairwise distances. Assuming that the Euclidean metric space, the Hausdorff distance is expressed in the following formula:

$$\max\left\{\sup_{p\in P}\inf_{q\in Q}\|p-q\|\,,\ \sup_{q\in Q}\inf_{p\in P}\|p-q\|\right\}.$$

The Hausdorff distance interprets a polygon as the set of points that lie in its interior and on its boundary. Similar to the symmetric difference, this interpretation disregards the continuous nature of the polygons. Thus, we expect similar problems to arise with the Hausdorff distance. This is indeed the case, as illustrated in Figure 3.5. The input is the same polygon as we used for the symmetric difference; the resulting schematizations are also similar. Again, we see that the axis of symmetry is lost and that the schematization

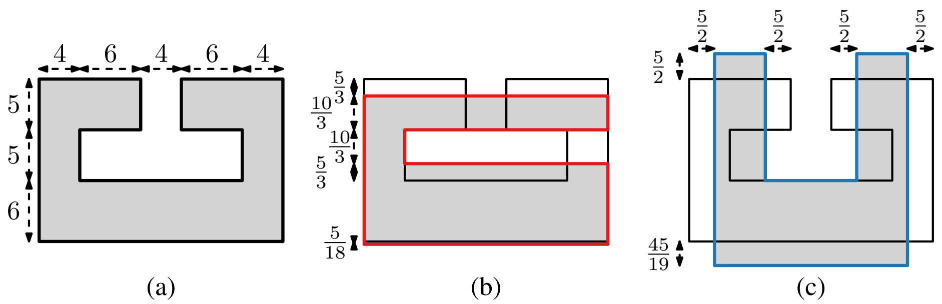


Figure 3.5 A polygon (a) and two 8-sided schematizations: with minimal Hausdorff distance (b), and with better shape (c).

with minimal Hausdorff distance has a "C-shape". We observe that the maximal distances in this example are obtained at the boundary of the polygons, rather than at an interior point. Thus, restricting the set of points that are used in the Hausdorff distance to only the boundary points results in the same issues.

3.4 Turning angle distance

The *turning angle distance* is defined by Arkin *et al.* [14]. It is based on a turning function, defined as follows. Let P be a curve, interpreted as a function on the domain $[0, 1]$ (rather than $[0, n]$). Its turning function $\Theta_P(x)$ is defined for $x \in [0, 1]$, expressing the angle of the tangent at $P(x)$ with respect to some reference angle. The turning angle distance between two curves P and Q is expressed as

$$\left(\int_0^1 |\Theta_P(x) - \Theta_Q(x)|^p \, dx \right)^{\frac{1}{p}},$$

for some $p > 0$. The use of $p = 2$ is suggested by Arkin *et al.* [14].

For polygons P and Q, the distance is the minimum over all possible ways to "cut" their boundaries into open curves and all rotations of Q. This distance is designed to invariant under translation, rotation and scaling. This already indicates that the distance measure is likely unsuitable for schematization: rotation and translation affect similarity for the schematization of territorial outlines. Scale invariance may be acceptable in combination with area preservation.

Even if we were able to eliminate the invariance, another problem arises. Arkin *et al.* [14] indicate that the measure is sensitive to nonuniform noise. In their application (nearly) uniform noise is a reasonable assumption. For schematization, however, this is not the case. For example, consider again Norway and its fjords (Figure 3.3). Its shoreline is significantly longer than its borders with adjacent countries. Though this is not noise, it indicates that territorial outlines are nonuniform: the characteristics may vary along the boundary. Figure 3.6 shows an example of how nonuniform "noise" influences the

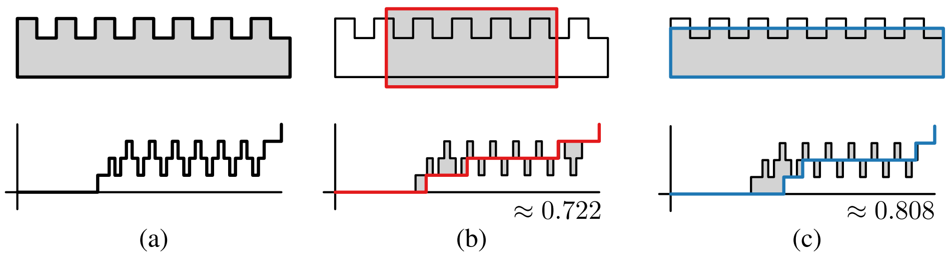


Figure 3.6 A polygon (a) and two 4-sided schematizations: with minimal turning angle distance (b), and with better shape (c). The turning functions are given in the bottom row.

optimal schematization with respect to the turning angle distance. The more "battlements" we add to the shape, the worse this effect becomes; the schematization with minimal turning angle distance tends towards a square.

3.5 Cyclic dynamic time warp distance

The *cyclic dynamic time warp distance* [124] is the dynamic time warp distance [22], adapted for polygons. Let $P = \langle p_0, \ldots, p_{m-1} \rangle$ and $Q = \langle q_0, \ldots, q_{n-1} \rangle$ be two polygons with m and n vertices respectively. A cyclic alignment between P and Q is a sequence $\langle (i_0, j_0), \ldots, (i_{k-1}, j_{k-1}) \rangle$ such that, for $0 \leqslant l < k$, the following conditions hold:

- $0 \leqslant i_{l+1} - i_l \leqslant 1$;
- $0 \leqslant j_{l+1} - j_l \leqslant 1$;
- $(i_l, j_l) \neq (i_{l+1}, j_{l+1})$.

For ease of notation, we assume that (i_k, j_k) represents (i_0, j_0). Let Φ be the set of all cyclic alignments. The cyclic dynamic time warp distance is defined as

$$\min_{\phi \in \Phi} \sum_{(i,j) \in \phi} d(p_i, q_j),$$

where $d(p_i, q_j)$ is some distance function between the vertices p_i and q_j. Essentially, this is a discrete summed Fréchet distance in which the Euclidean distance has been replaced by an arbitrary function.

Marzal and Palazón [124] suggest to use curvature for the distance function. This causes the measure to be invariant under translation, rotation and scaling. As with the turning angle distance, this is undesirable for schematization purposes. Using for example the Euclidean distance also results in undesirable behavior, as is shown in Figure 3.7. The problem is caused by the discrete steps from vertex to vertex and the uneven sampling between polygon and its approximation. This is similar to the issues with integral Fréchet distance and to the problem of "nonuniform noise" for the turning angle distance.

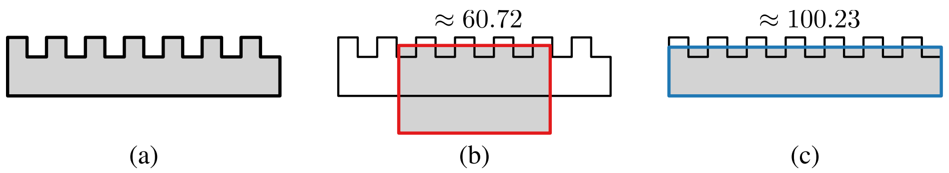


Figure 3.7 A polygon (a) and two 4-sided approximations: with minimal cyclic dynamic time warp distance (b), and with better shape (c).

To overcome the drawbacks of a discrete measure, a continuous version of the dynamic time warping distance has been considered [129]. However, it uses a different measure between points on the curve, such that the distance between curves is invariant under translation. Again, we consider this to be undesirable for schematization. Using the Euclidean distance instead results in an integral Fréchet distance.

3.6 Conclusions

In this chapter we considered a number of common similarity measures and their appropriateness for quantifying resemblance for the purpose of cartographic schematization. However, each of these measures exhibits some undesirable behavior: optimizing for the similarity measure may result in unintuitive results. We consider the outlier sensitivity of the Fréchet distance to be the least severe. Hence, we further investigate this similarity measure in the upcoming chapters. In Chapter 4 we consider the computation of the Fréchet distance and give the first asymptotic improvement for two general curves since the result of Alt and Godau [10] in 1995. In Chapter 5 we introduce a stricter notion of a Fréchet matching in order to reduce the effects of outliers on the quality of the matching.

Chapter 4

Computing the Fréchet distance

To judge the quality of a schematization, we opted to further investigate the Fréchet distance. A first step in this investigation is the computation of the Fréchet distance for two given polygonal curves. In 1995, Alt and Godau [10] published an $O(n^2 \log n)$ algorithm to compute the Fréchet distance for two curves of complexity n. Previous to the work presented in this chapter, this algorithm has not been improved, despite the large amount of research that followed. The quadratic execution time for the decision problem remained the best known bound for general curves. If we cannot improve on a quadratic bound for a geometric problem despite many efforts, a possible cause may be the underlying 3SUM-hardness [85]. This situation led Helmut Alt to make the following conjecture [7].

Conjecture 4.1. [ALT'S CONJECTURE] *Let P, Q be two polygonal curves in the plane. It is* 3SUM*-hard to decide whether the Fréchet distance between P and Q is at most* 1.

Here, "1" can be considered as an arbitrary constant, which can be changed to any other bound by scaling the curves. Until recently, the only known lower bound was $\Omega(n \log n)$ in the algebraic-computation-tree model [34]. Assuming the Strong Exponential Time Hypothesis, Bringmann [30] shows that no $O(n^{2-\delta})$ algorithm exists for any $\delta > 0$.

Recently, Agarwal *et al.* [2, 3] showed how to achieve a subquadratic-time algorithm for the *discrete* Fréchet distance. They asked whether their result can be generalized to the case of the original (continuous) Fréchet distance. In this chapter we address this question and provide an algorithm to compute the Fréchet distance in $O(n^2 \sqrt{\log n}(\log \log n)^{3/2})$ expected time between two curves of complexity n. This is the first algorithm with an execution time of $o(n^2 \log n)$, constituting the first asymptotic improvement for the general case since the original paper by Alt and Godau [10]. To achieve this result, we give the first subquadratic algorithm for the decision problem of the Fréchet distance. As in most algorithms for computing the Fréchet distance, the optimization problem is solved by performing an appropriate search over the critical values to solve the optimization problem. However, note that this requires a new search algorithm, as previously known methods have an execution time of $O((n^2 + T(n)) \log n)$ where $T(n)$ represents the computation time for the decision problem.

We emphasize that these algorithms run on a real RAM/pointer machine and do not require any bit-manipulation tricks. Therefore, our results are more in the line of Chan's recent subcubic-time algorithms for all-pairs-shortest paths [50, 51] or recent subquadratic-time algorithms for MIN-PLUS-CONVOLUTION [29] than the subquadratic-time algorithms for 3SUM [19]. If we relax the model to allow constant-time table-lookups, the execution time can be improved to be almost quadratic, up to $O(\log \log n)$ factors.

Finally, we briefly consider Alt's conjecture. We show that *nonuniformly*, the Fréchet distance can be computed in subquadratic time. More precisely, we prove that the decision problem can be solved by an algebraic decision tree [16] of depth $O(n^{2-\alpha})$, for some fixed $\alpha > 0$. It was conjectured that no such decision tree exists for 3SUM [138] and an $\Omega(n^2)$ lower bound is known in a restricted linear decision tree model [6, 76]. Under this conjecture, our result would have implied that Alt's conjecture is false. However, recently it has been proven that 3SUM admits an algebraic decision tree of depth $O(n^{3/2}\sqrt{\log n})$ [93]. Hence, our results have no implications on Alt's conjecture.

However, the $O(n^{2-\delta})$ lower bound proven by Bringmann [30] proves that the decision tree cannot be (uniformly) implemented, unless the Strong Exponential Time Hypothesis fails. This provides strong evidence for a discrepancy between the decision-tree model and the uniform complexity of the Fréchet problem. This puts it into the illustrious company of such notorious problems as SORTING $X + Y$ [84], MIN-PLUS-CONVOLUTION [29], or finding the Delaunay triangulation for a point set that has been sorted in two orthogonal directions [43]. We find that this aspect of the Fréchet distance is highly intriguing and deserves further study.

Related work. The Fréchet distance was originally introduced by Alt and Godau [10, 89] into the algorithmic literature. They showed that this distance can be computed in $O(mn \log mn)$ time for two polygonal curves, described by m and n points. Since Alt and Godau's seminal paper, there has been a wealth of research in various directions, such as extensions to higher dimensions [8, 37, 40, 44, 60, 90], approximation algorithms [11, 15, 71], the geodesic and the homotopic Fréchet distance [48, 61, 74, 97], and much more [5, 24, 39, 70, 110, 120, 121]. A fully polynomial-time approximation scheme is known for arbitrary polygonal curves, with an execution time of $O(n^2 \log(1/\epsilon))$ [38]. All subquadratic-time (exact or approximation) algorithms make further assumptions on the curves. The Fréchet distance and its variants, such as dynamic time-warping [22, 124], have found various applications, with recent work particularly focusing on geographic applications such as map matching [28, 176] (see also Chapter 6), moving-object analysis [32, 33, 94] and simplification [1, 4, 68].

4.1 Preliminaries and overview

Throughout this chapter we assume that P and Q are open polygonal curves with m and n edges respectively. Their vertices are denoted by $p_0, p_1, \dots, p_m$ and $q_0, q_1, \dots, q_n$. For the analysis, we assume that $m \leqslant n$; because $d_{\mathrm{F}}(P, Q) = d_{\mathrm{F}}(Q, P)$, this assumption does not violate its generality. First, we briefly recap the most significant concepts for deciding whether $d_{\mathrm{F}}(P, Q) \leqslant \varepsilon$ (see Section 2.4.2 for more details).

The *free-space diagram*, $\mathrm{FSD}(P, Q)$, can be used to determine whether the Fréchet distance is at most a given constant ε. This diagram represents the joint parameter space and is partitioned into m columns and n rows, and thus $m \cdot n$ cells in total. The *free space* consists of the points in the diagram that represent pairs of points on the curves that are within distance ε of one another. The *reachable space*, $\mathrm{reach}(P, Q)$, is the subset of the free space that can be reached via a bimonotone path through the free space from $(0, 0)$. To decide $d_{\mathrm{F}}(P, Q) \leqslant \varepsilon$, we must decide whether $(m, n) \in \mathrm{reach}(P, Q)$. For this, it is sufficient to consider only the boundaries of the cells. The free space of a cell boundary is referred to as a *door*; its reachable subset is a *reach-door*. A door is said to be *closed* if the interval is empty, and *open* otherwise. The reach-doors can be found in $O(mn)$ time through a simple traversal of the cells [10]. In the next sections we show how to obtain the crucial information—whether $(m, n) \in \mathrm{reach}(P, Q)$—in $o(mn)$ instead.

4.1.1 Basic approach and intuition

In our algorithm for the decision problem, we essentially want to compute $\mathrm{reach}(P, Q)$. But instead of propagating the reachability information cell by cell, we always group τ by τ cells (with $1 \ll \tau \ll n$) into an *elementary box* of cells. When processing a box, we can assume that we know which parts of the left and the bottom boundary of the box are reachable. That is, we know the reach-doors on the bottom and left boundary, and we need to compute the reach-doors on the top and right boundary of the elementary box.

These reach-doors are determined by the combinatorial structure of the elementary box. More specifically, if we know for every row and column the order of the door endpoints (including the reach-doors on the left and bottom boundary), we know which door endpoints determine the reach-doors on the top and right boundary. We call the sequence of these orders, the *(full) signature* of the box.

The total number of possible signatures is bounded by an expression in terms of τ. Hence, if we pick τ sufficiently small compared to n, we can pre-compute for all possible signatures the reach-doors on the top and right boundary. We describe a data structure to query these quickly (Section 4.2). Since the reach-doors on the bottom and left boundary are required to make the signature, we initially have only *partial* signatures. In Section 4.3, we describe how to compute these efficiently. The partial signatures are then used to preprocess the data structure such that we can quickly find the full signature once we know the reach-doors of an elementary box. With this preprocessed data structure, $d_{\mathrm{F}}(P, Q) \leqslant \varepsilon$ can be decided efficiently by traversing the free-space diagram elementary box by elementary box, as explained in Section 4.4.

4.1.2 Computational models

In this chapter we develop algorithms that depend on the model of computation. Below, we briefly discuss these computational models.

Real RAM. The standard machine model in computational geometry is the *real RAM*. Here, data is represented as an infinite sequence of storage cells. These cells can be of two different types: they can store real numbers or integers. The model supports standard operations on these numbers in constant time, including addition, multiplication, and elementary functions like square-root, sine or cosine. Furthermore, the integers can be used as indices to memory locations. Integers can be converted to real numbers in constant time, but we need to be careful about the reverse direction. The *floor* function can be used to truncate a real number to an integer, but if we were allowed to use it arbitrarily, the real RAM could solve PSPACE-complete problems in polynomial time. Therefore, we usually have only a restricted floor function at our disposal.

Word RAM. The *word RAM* is essentially a real RAM without support for real numbers. However, on a real RAM, the integers are usually treated as atomic, whereas the word RAM allows for powerful bit-manipulation tricks. More precisely, the word RAM represents the data as a sequence of w-bit words, where $w = \Omega(\log n)$. Data can be accessed arbitrarily, and standard operations, such as Boolean operations (`and`, `xor`, `shl`, $\ldots$), addition, or multiplication take constant time. There are many variants of the word RAM, depending on precisely which instructions are supported in constant time. AC^0 is the class of all functions $f\colon \{0,1\}^* \to \{0,1\}^*$ that can be computed by a family of circuits $(C_n)_{n\in\mathbb{N}}$ with the following properties:

(i) each C_n has n inputs;
(ii) there exist constants a, b, such that C_n has at most an^b gates, for $n \in \mathbb{N}$;
(iii) there is a constant d such that for all n the length of the longest path from an input to an output in C_n is at most d (i.e., the circuit family has bounded depth);
(iv) each gate has an arbitrary number of incoming edges (i.e., the fan-in is unbounded).

The general consensus seems to be that any function in AC^0 is acceptable. However, it is always preferable to rely on a set of operations as small and simple as possible. Note that multiplication is not in AC^0, but nevertheless is usually included in the word RAM instruction set.

Pointer machine. The *pointer machine* model disallows the use of constant time table lookup, and is therefore a restriction of the (real) RAM model. The data structure is modeled as a directed graph G with bounded out-degree. Each node in G represents a *record*, with a bounded number of pointers to other records and a bounded number of (real or integer) data items. The algorithm can access data only by following pointers from the inputs (and a bounded number of global entry records); random access is not possible. The data can be manipulated through the usual real RAM operations, but without support for the floor function, for reasons mentioned above.

Algebraic computation tree. *Algebraic computation trees* (ACTs) [16] are the computational analogue of binary decision trees; like these, they are mainly used for proving lower bounds. Let $x_1, \ldots, x_n \in \mathbb{R}$ be the inputs. An ACT is a binary tree with two kinds of nodes: *computation* and *branch* nodes. A computation node v has one child and is labeled with an expression of the type $y_v = y_u \oplus y_w$, where $\oplus \in \{+, -, *, /, \sqrt{\cdot}\}$ is an operator and y_u, y_w is either an input variable $x_1, \ldots, x_n$ or corresponds to a computation node that is an ancestor of v. A branch node has degree 2 and is labeled by $y_u = 0$ or $y_u > 0$, where again y_u is either an input or a variable for an ancestor. An ACT T_n accepts inputs of size n; each such input $x_1, \dots, x_n$, defines a path in T_n where children of a branch nodes are determined according to the represented condition. This path in T_n constitutes a computation that represents the answer in the encountered variables y_v. A family of algebraic computation trees $(T_n)_{n \in \mathbb{N}}$ solves a computational problem, if for each $n \in \mathbb{N}$, the ACT T_n represents a correct computation for inputs of size n.

Algebraic *decision* trees are defined as follows. We allow only branch nodes; each branch node is labeled with a predicate of the form $p(x_1, \dots, x_n) = 0$ or $p(x_1, \dots, x_n) > 0$. The leaves are labeled *yes* or *no*. Fix some $r \in \{1, \dots, n\}$. If p is restricted to be of the form $p(x_1, \dots, x_n) = \sum_{i=1}^{n} a_i x_i - b$, with at most r coefficients $a_i \neq 0$, we call the decision tree r-*linear*. Erickson [76] showed that any 3-linear decision tree for 3SUM has depth $\Omega(n^2)$. However, this bound does not say anything about more general predicates (e.g. if p may include quadratic terms). This severely limits its applicability to geometric problems. For example, there is no r-linear decision tree for the Fréchet problem, no matter the choice of r: with r-linear decision trees, we cannot even decide whether two given points p and q have Euclidean distance at most 1. Using general predicates, 3SUM admits an algebraic decision tree of depth $O(n^{3/2}\sqrt{\log n})$ [93].

4.2 Building a lookup table

4.2.1 Preprocessing an elementary box

Before it considers the input, our algorithm builds a lookup table to speed up the computation of small parts of the free-space diagram.

Let $\tau \in \mathbb{N}$ be a parameter. The *elementary box* is a partition of $[0, \tau]^2$ into τ columns and rows, thus τ^2 cells. For $(i, j) \in \{0, \dots, \tau-1\}^2$, we denote the cell $[i, i+1] \times [j, j+1]$ with $D(i, j)$. As with cells in the free-space diagram, we denote the left side of the boundary $\partial D(i, j)$ by $L_{i,j}$ and the bottom side by $B_{i,j}$. Note that $L_{i,j}$ coincides with the right side of $\partial D(i-1, j)$ and $B_{i,j}$ with the top of $\partial D(i, j-1)$. Thus, we write $L_{\tau,j}$ for the *right* side of $\partial D(\tau-1, j)$ and $B_{i,\tau}$ for the *top* side of $\partial D(i, \tau-1)$. The door on a vertical side $L_{i,j}$, the intersection of the boundary with the free space, is denoted by $L^{\mathrm{F}}_{i,j}$; the reach-door, the intersection with $\mathrm{reach}(P, Q)$, is denoted by $L^{\mathrm{R}}_{i,j}$. We use analogous notation for the horizontal sides $B_{i,j}$. Figure 4.1 shows the elementary box. For now, the elementary box is a combinatorial concept. In Section 4.3 and Section 4.4, we overlay these boxes on the free-space diagram to obtain "concrete" elementary boxes.

The *door-order* σ^{r}_j for a row j is a permutation of $\{s_0, t_0, \dots, s_\tau, t_\tau\}$, thus having $2\tau + 2$ elements. For $i \in \{1, \dots, \tau\}$, the element s_i represents the lower endpoint of $L^{\mathrm{F}}_{i,j}$, and t_i represents the upper endpoint. If the door is closed, s_i comes after t_i in σ^{r}_j. The elements s_0 and t_0 are an exception: they describe $L^{\mathrm{R}}_{0,j}$, the reach-door on the leftmost boundary in the row. The door-order σ^{r}_j represents the combinatorial order of these endpoints, as projected onto a vertical line, i.e., they are sorted into their vertical order. Some door-orders may encode the same combinatorial structure. In particular when door i is closed, the exact position of s_i and t_i in a door-order is irrelevant, up to t_i

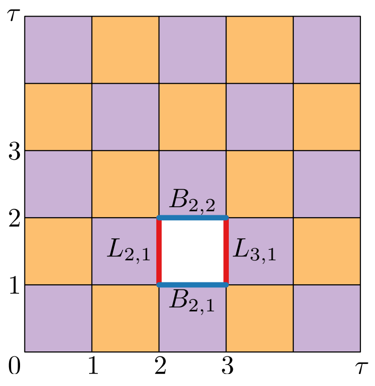


Figure 4.1 The elementary box. The cell $D(2,1)$ is shown white. Its four sides—$L_{2,1}$, $L_{3,1}$, $B_{2,1}$, $B_{2,2}$—are indicated.

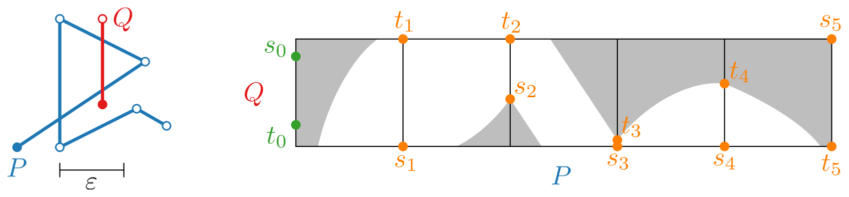


Figure 4.2 The door-order of a row (the vertical order of the points) encodes the combinatorial structure of the doors. The door-order for the row in the figure is $\langle s_1, s_3, s_4, t_5, t_3, t_0, s_2, t_4, s_0, s_5, t_1, t_2\rangle$. Note that s_0 and t_0 represent the reach-door, which is empty in this case. These are omitted in the partial door-order.

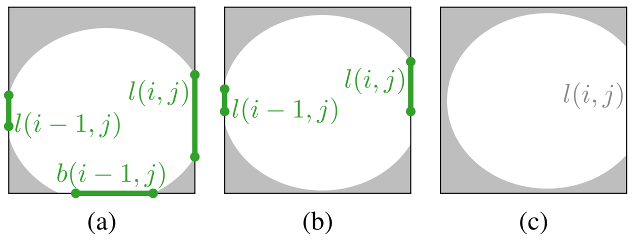


Figure 4.3 The three cases for the recursive definition of $l(i,j)$. (a) If the lower boundary is reachable, the complete right door is reachable. (b) If the lower boundary is not reachable but the left is, the lower boundary is the maximum of $l(i-1,j)$ and the lower boundary of the right door. (c) Otherwise, neither the lower nor the left boundary is reachable; the right door is not reachable either.

being before s_i. For a closed door i ($i > 0$), we assign s_i to the upper endpoint of $l(i, j)$ and t_i to the lower endpoint. The values of s_0 and t_0 are defined by the reach-door and their relative order is thus a result of the computation. We break ties between s_i and $t_{i'}$ by placing s_i before $t_{i'}$, and any other ties are resolved by index. A door-order σ_i^{c} is defined analogously for a column i. We write $x <_i^{\mathrm{c}} y$ if x comes before y in σ_i^{c}, and $x <_j^{\mathrm{r}} y$ if x comes before y in σ_j^{r}. A *partial door-order* is a door-order in which s_0 and t_0 are omitted (i.e., the reach-door is still unknown). A door-order is illustrated in Figure 4.2.

We define the *(full) signature* of the elementary box as the aggregation of the door-orders of its rows and columns. Therefore, a signature $\Sigma = (\sigma_1^{\mathrm{c}}, \dots, \sigma_\tau^{\mathrm{c}}, \sigma_1^{\mathrm{r}}, \dots, \sigma_\tau^{\mathrm{r}})$ consists of 2τ door-orders: one door-order σ_i^{c} for each column i and one door-order σ_j^{r} for each row j of the elementary box. Similarly, a *partial signature* is the aggregation of partial door-orders. For a given signature, we define the *combinatorial reachability structure* of the elementary box as follows. For each column i and for each row j, the combinatorial reachability structure indicates which door endpoints in the respective column or row define the reach-door $B_{i,\tau}^{\mathrm{R}}$ or $L_{\tau,j}^{\mathrm{R}}$.

Lemma 4.2. *Let Σ be a signature for the elementary box. Then we can determine the combinatorial reachability structure of Σ in total time $O(\tau^2)$.*

Proof. We use dynamic programming, very similar to the algorithm by Alt and Godau [10]. For each vertical edge $L_{i,j}$ we define a variable $l(i, j)$, and for each horizontal edge $B_{i,j}$ we define a variable $b(i, j)$. The $l(i, j)$ are pairs of the form (s_u, t_v), representing the reach-door $L_{i,j}^{\mathrm{R}}$. If this reach-door is closed, then $t_v <_j^{\mathrm{r}} s_u$ holds. If the reach-door is open, then it is bounded by the lower endpoint of the (free-space) door $L_{u,j}^{\mathrm{F}}$ and by the upper endpoint of $L_{v,j}^{\mathrm{F}}$. In the second case, v is always equal to i. Once again, s_0 and t_0 are special and represent the reach-door $L_{0,j}^{\mathrm{R}}$ instead of the door $L_{0,j}^{\mathrm{F}}$. The variables $b(i, j)$ are defined analogously. Now we can compute $l(i, j)$ and $b(i, j)$ recursively as follows. First, for $j \in \{0, \dots, \tau - 1\}$, we set $l(0, j)$ to the reach-door (s_0, t_0) for the row j as given by Σ. Analogously, we set the values $b(i, 0)$ for $i \in \{0, \dots, \tau - 1\}$.

Next, we describe how to find $l(i, j)$ given $l(i - 1, j)$ and $b(i - 1, j)$, based on three cases; this is also illustrated in Figure 4.3. Note that this is analogous to the regular computation of reachability (see Section 2.4.2).

(1) Suppose $b(i - 1, j)$ is open. This means that $B_{i-1,j}$ is reachable and thus reach-door $L_{i,j}^{\mathrm{R}}$ is limited only by the free-space door $L_{i,j}^{\mathrm{F}}$. Hence, we can set $l(i, j) := (s_i, t_i)$.
(2) If $b(i-1, j)$ is closed and $l(i-1, j)$ is open, we may be able to reach $L_{i,j}$ via $L_{i-1,j}$. Let s_u be the lower endpoint of $l(i - 1, j)$. We need to pass $l(i, j)$ above s_u and s_i and below t_i. Therefore, we set $l(i, j) := (\max(s_u, s_i), t_i)$, where the maximum is taken according to the order $<_j^{\mathrm{r}}$.
(3) If both $b(i - 1, j)$ and $l(i - 1, j)$ are closed, it is impossible to reach $L_{i,j}$. Hence, we must set $l(i, j)$ to be closed as well; we set $l(i, j) := l(i - 1, j)$.

The recursion for the variable $b(i, j)$ is defined similarly. We can implement the recursion in $O(\tau^2)$ time for any given signature, for example, by traversing the elementary box column by column, while processing each column from bottom to top. □

There are at most $((2\tau + 2)!)^{2\tau} = \tau^{O(\tau^2)}$ distinct signatures for the elementary box. We choose $\tau = \lambda\sqrt{\log n / \log\log n}$ for a sufficiently small constant $\lambda > 0$, so that this number becomes $o(n)$. The constant λ is used to compensate for the constant hidden by the O-notation in the number of signatures, $\tau^{O(\tau^2)}$. Thus, during the preprocessing stage we have time to enumerate all possible signatures and determine the corresponding combinatorial reachability structure inside the elementary box. This information is then stored in an appropriate data structure, as described in the next section.

4.2.2 Building the data structure

Before we describe our data structure, we first explain how the door-orders are represented. This depends on the computational model. By our choice of τ, there are $o(n)$ distinct door-orders. On the word RAM, we represent each door-order and partial door-order by an integer between 1 and $(2\tau)!$. This fits into a word of $\log n$ bits. On the pointer machine, we create a record for each door-order and partial door-order; we represent an order by a pointer to the corresponding record.

The data structure has two *stages*, as schematically depicted in Figure 4.4. In the first stage (Figure 4.4(a–b)), we compute the partial signature from the partial door-orders. For this stage we assume that we know the partial door-order for each row and for each column of the elementary box. In the next section we describe how to determine the partial door-orders efficiently to fulfill this assumption. In the second stage (Figure 4.4(c–d)), we have obtained the reach-doors for the left and bottom sides of the elementary box, and we determine the full signature. The details of our method depend on the computational model. One way uses table lookup and requires the word RAM; the other way works on the pointer machine, but is a bit more involved.

Word RAM. We organize the lookup table as a large tree T. In the first stage, each level of T corresponds to a row or column of the elementary box. Thus, there are 2τ levels.

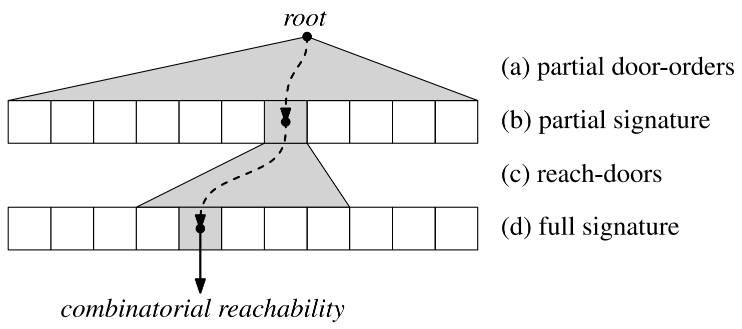


Figure 4.4 Overview of the data structure. (a–b) Using partial door-orders, we find the partial signature of an elementary box. (c–d) Using the reach-doors on the bottom and left boundary of a box, we find the full signature and the combinatorial reachability.

Each node has $(2\tau)!$ children, representing the possible partial door-orders for the next row or column. Since we represent door-orders by positive integers, each node of T may store an array for its children; we can choose the appropriate child for a given partial door-order in constant time. Thus, determining the partial signature for an elementary box requires $O(\tau)$ steps on a word RAM.

For the second stage, we again use a tree structure. Now the tree has $O(\tau)$ *layers*, each with $O(\log \tau)$ levels. Again, each layer corresponds to a row or column of the elementary box. The levels inside each layer then implement a balanced binary search tree that allows us to locate the endpoints of the reach-door within the partial signature. Since there are 2τ endpoints, this requires $O(\log \tau)$ levels. Thus, it takes $O(\tau \log \tau)$ time to find the full signature of a given elementary box.

Pointer model. Unlike in the word RAM model, we are not allowed to store a lookup table on every level of the tree T, and there is no way to quickly find the appropriate child for a given door-order. Instead, we must rely on batch processing to achieve a reasonable execution time.

Thus, suppose that during the first stage we want to find the partial signatures for a set B of elementary boxes; for each box in B we know the partial door-order for each row and each column. Recall that we represent the door-order by a pointer to the corresponding record. With each such record, we store a queue of elementary boxes that is empty initially.

We now simultaneously propagate the boxes in B through T, proceeding level by level. In the first level, all of B is assigned to the root of T. Then, we go through the nodes of one level of T, from left to right. Let v be the current node of T. We consider each elementary box b assigned to v. We determine the next partial door-order for b, and we append b to the queue for this partial door-order. This queue is addressed through the corresponding record, so all elementary boxes with the same next partial door-order end up in the same queue. Next, we go through the nodes of the next level, again from left to right. Let v' be the current node. The node v' corresponds to a next partial door-order σ that extends the known signature of its parents. We consider the queue stored at the record for σ. By construction, the elementary boxes that should be assigned to v' appear consecutively at the beginning of this queue. We remove these boxes from the queue and assign them to v'. After this, all the queues are empty, and we can continue by propagating the boxes to the next level. During this procedure, we traverse each node of T a constant number of times, and in each level of the T we consider all boxes in B. Since T has $o(n)$ nodes, the total running time is $O(n + |B|\tau)$.

For the second stage, the data structure works just as in the word RAM case, because no table lookup is necessary. Again, we need $O(\tau \log \tau)$ steps to process one box.

After the second stage we obtain the combinatorial reachability structure of the box in constant time, since we precomputed this information for each box. Thus, we have shown the following lemma, independently of the computational model.

Lemma 4.3. *For $\tau = \lambda\sqrt{\log n / \log\log n}$ with a sufficiently small constant $\lambda > 0$, we can construct in $o(n)$ time a data structure of size $o(n)$ such that*

- *given a set B of elementary boxes where the partial door-orders are known, we can find the partial signature of each box in total time $O(n + |B|\tau)$;*
- *given the partial signature and the reach-doors on the bottom and left boundary of an elementary box, we can find the full signature in $O(\tau \log \tau)$ time;*
- *given the full signature of an elementary box, we can find the combinatorial reachability structure of the box in constant time.*

4.3 Preprocessing a given input

Next, we perform a second preprocessing phase that considers the input curves P and Q. Our final goal is to compute the intersection of $\mathrm{reach}(P, Q)$ with the cell boundaries, taking advantage of the data structure from Section 4.2. For this, we aggregate the cells of $\mathrm{FSD}(P, Q)$ into (concrete) elementary boxes consisting of $\tau \times \tau$ cells. There are mn/τ^2 such boxes. We may avoid rounding issues by either duplicating vertices or handling a small part of $\mathrm{FSD}(P, Q)$ without lookup tables.

The goal is to determine the signature for each elementary box S. At this point this is not quite possible yet, since the signature depends on the intersection of $\mathrm{reach}(P, Q)$ with the lower and left boundary of S. Nonetheless, we can find the partial signature, in which the positions of s_0, t_0 (the reach-door) in the partial door-orders σ_i^{r}, σ_j^{c} are still to be determined.

As illustrated in Figure 4.5, we aggregate each column of elementary boxes into a vertical *strip*, that is, τ consecutive columns of cells in $\mathrm{FSD}(P, Q)$. Likewise, we aggregate each row of elementary boxes into a horizontal strip. Thus, there are m/τ vertical and n/τ horizontal strips. Let A be a vertical strip, corresponding to a subcurve P' of P with τ edges. The following lemma implies that we can build a data structure for A such that, given any edge of Q, we can efficiently find its partial door-order within the elementary box in A.

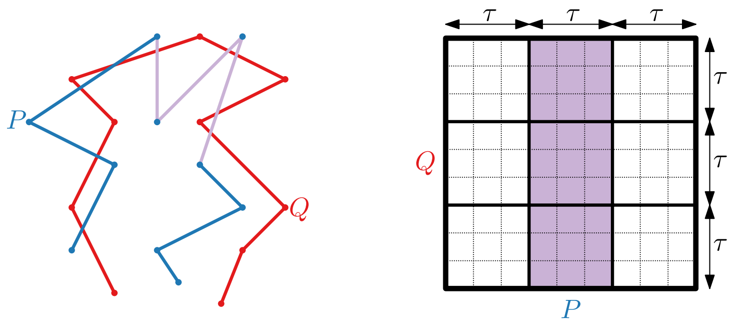


Figure 4.5 The free-space diagram is subdivided into mn/τ^2 elementary boxes of size $\tau \times \tau$. A strip is a column of elementary boxes: it corresponds to a subcurve of P with τ edges.

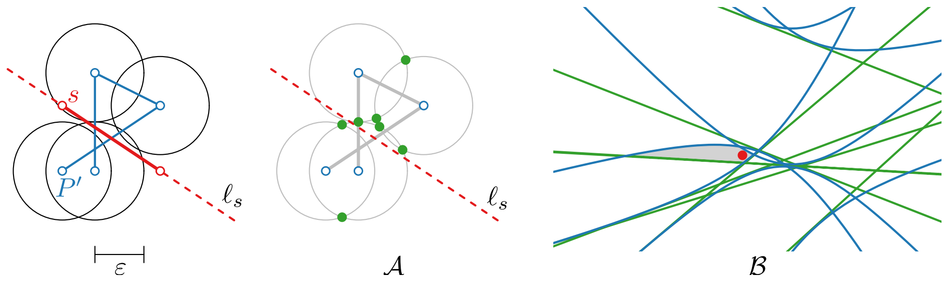


Figure 4.6 By using the arrangement $\mathcal{A}$ defined by circles with radius ε centered at vertices of P', we can determine the partial door-order of each segment s on Q. This is done by locating the dual point of ℓ_s in the dual arrangement $\mathcal{B}$. The dual arrangement also contains pseudolines to determine when ℓ_s leaves a circle of $\mathcal{A}$.

Lemma 4.4. *There exists a constant c such that the following holds: given a subcurve P' with τ edges, we can compute in $O(\tau^c)$ time a data structure that requires $O(\tau^c)$ space and that allows us to determine the partial door-order of any edge of Q in time $O(\log \tau)$.*

Proof. Consider the arrangement $\mathcal{A}$ of circles with radius ε whose centers are the vertices of P' (see Figure 4.6). The partial door-order of a line segment s is determined by the intersections of s with the arcs of $\mathcal{A}$ and, for a circle not intersecting s, by whether s lies inside or outside of the circle. Let ℓ_s be the line spanned by line segment s. Suppose we wiggle ℓ_s. The order of intersections of ℓ_s and the arcs of $\mathcal{A}$ changes only when ℓ_s moves over a vertex of $\mathcal{A}$ or if ℓ_s leaves or enters a circle.

We use the standard duality transform that maps a line $\ell : y = ax + b$ to the point $\ell^* : (a, -b)$, and vice versa. Consider a circle C in $\mathcal{A}$ with center (c_x, c_y). Elementary geometry shows that the set of all lines that are tangent to C from above dualizes to the curve $t_a^*(C)\colon y = c_x x - c_y - \varepsilon\sqrt{1 + x^2}$. Similarly, the lines that are tangent to C from below dualize to the curve $t_b^*(C)\colon y = c_x x - c_y + \varepsilon\sqrt{1 + x^2}$. Define $C^* := \{t_a^*(C), t_b^*(C) \mid C \in \mathcal{A}\}$. Since any pair of distinct circles C_1, C_2 has at most four common tangents, one for each choice of above or below C_1 and above or below C_2, it follows that any two curves in C^* intersect at most once.

Let V be the set of vertices in $\mathcal{A}$, and let V^* be the lines dual to the points in V (note that $|V| = O(\tau^2)$). Since for any vertex $v \in V$ and any circle $C \in \mathcal{A}$ there are at most two tangents through v on C, each line in V^* intersects each curve in C^* at most once. Thus, the arrangement $\mathcal{B}$ of the curves in $V^* \cup C^*$ is an arrangement of *pseudolines* with complexity $O(\tau^4)$. Furthermore, it can be constructed in the same expected time, together with a point location structure that finds the containing cell in $\mathcal{B}$ of any given point in time $O(\log \tau)$ [159, Chapter 6.6.1].

Now consider a line segment s and the supporting line ℓ_s. As observed in the first paragraph, the combinatorial structure of the intersection between ℓ_s and $\mathcal{A}$ is completely determined by the cell of $\mathcal{B}$ that contains the dual point ℓ_s^*. Thus, for every cell $f(s) \in \mathcal{B}$, we construct a list $L_{f(s)}$ that represents the combinatorial structure of $\ell_s \cap \mathcal{A}$. There are $O(\tau^4)$ such lists, each having size $O(\tau)$. We can compute $L_{f(s)}$ by traversing the zone of ℓ_s in $\mathcal{A}$. Circles intersect at most twice and also a line intersects any circle at most twice; hence, the zone has complexity $O(\tau 2^{\alpha(\tau)})$, where $\alpha(\cdot)$ denotes the inverse Ackermann function [159, Theorem 5.11]. Since $O(\tau 2^{\alpha(\tau)}) \subset O(\tau^2)$, we can compute all lists in $O(\tau^6)$ time.

Given the list $L_{f(s)}$, the partial door-order of s is determined by the position of the endpoints of s in $L_{f(s)}$. There are $O(\tau^2)$ possible ways for this, and we build a table $T_{f(s)}$ that represents them. For each entry in $T_{f(s)}$, we store a representative for the corresponding partial door-order. As described in the previous section, the representative is a positive integer in the word RAM model and a pointer to the appropriate record on a pointer machine.

The total size of the data structure is $O(\tau^6)$ and it can be constructed in the same time. A query works as follows: given s, we can compute ℓ_s^* in constant time. Then we use the point location structure of $\mathcal{B}$ to find $f(s)$ in $O(\log \tau)$ time. Using binary search on $T_{f(s)}$ (or an appropriate tree structure in the case of a pointer machine), we can then determine the position of the endpoints of s in the list $L_{f(s)}$ in $O(\log \tau)$ time. This bound holds both on the word RAM and on the pointer machine. □

Lemma 4.5. *Given the data structure of Lemma 4.3, the partial signature of each elementary box can be determined in $O(n\tau^{c-1} + mn(\log \tau)/\tau)$ time for some constant c.*

Proof. For each of the m/τ vertical strips, we build the data structure from Lemma 4.4. We query this data structure to determine the partial door-order of each of the n rows in the corresponding elementary boxes. Thus, in total, this takes $O(\frac{m}{\tau}(\tau^c + n \log \tau)) = O(m\tau^{c-1} + mn \log \tau/\tau)$ time. We repeat the procedure with the horizontal strips; this step runs in $O(n\tau^{c-1} + mn \log \tau/\tau)$.

We now know, for each elementary box in $\mathrm{FSD}(P, Q)$, the partial door-order for each row and each column. We use the data structure of Lemma 4.3 to combine them. As there are mn/τ^2 boxes, the number of steps is $O(n + mn/\tau) = O(mn/\tau)$. Hence, the partial signature for each elementary box is computed in $O(n\tau^{c-1} + mn(\log \tau)/\tau)$. □

4.4 Solving the decision problem

With the data structures and preprocessing steps from the previous sections, we have all elements in place to determine whether $d_{\mathrm{F}}(P, Q) \leqslant \varepsilon$. We know for each elementary box its partial signature. In addition, we have a data structure to derive its full signature (and with it, the combinatorial reachability structure) when its reach-doors are known. What remains to be shown is that we can efficiently process the free-space diagram to determine whether $(m, n) \in \mathrm{reach}(P, Q)$. This is captured in the following lemma.

Lemma 4.6. *If the partial signature for each elementary box is known, we can determine whether $(m, n) \in \mathrm{reach}(P, Q)$ in time $O(mn(\log \tau)/\tau)$.*

Proof. We go through all of the elementary boxes of $\mathrm{FSD}(P, Q)$, processing them one row at a time, going from left to right in each row. Initially, we know the full signature for box S in the lower left corner of $\mathrm{FSD}(P, Q)$. We use the signature to determine the intersections of $\mathrm{reach}(P, Q)$ with the upper and right boundary of S. There is a subtlety here: the signature gives us only the combinatorial reachability structure. Hence, we need to map the resulting s_i, t_j back to the corresponding vertices on the curves. On the word RAM, this can be done easily through constant-time table lookups. On the pointer machine, we use representative records for the s_i, t_i elements and use $O(\tau)$ time before processing the box to store a pointer from each representative record to the appropriate vertices on P and Q.

We proceed similarly for the other boxes. By the choice of the processing order of the elementary boxes, we always know the incoming reach-doors on the bottom and left boundary when processing a box. Given the incoming reach-doors, we determine the full signature and find the structure of the outgoing reach-doors in total time $O(\tau \log \tau)$, using Lemma 4.3. Again, we need $O(\tau)$ additional time on the pointer machine to establish the mapping from the abstract s_i, t_i elements to the concrete vertices of P and Q. In total, we spend $O(\tau \log \tau)$ time per box and there are mn/τ^2 boxes. Thus, it takes time $O(mn(\log \tau)/\tau)$ to process all boxes, as claimed. □

As a result, we obtain the following theorem for a pointer machine (and by extension, for the real RAM model). For the word RAM model, we obtain a slightly faster algorithm (see Section 4.5).

Theorem 4.7. *On a pointer machine, the decision version of the Fréchet problem can be solved in $O(mn(\log\log n)^{3/2}/\sqrt{\log n})$ time on a pointer machine, assuming $m \leqslant n$ and $m = \Omega(\log^3 n)$.*

Proof. Set $\tau = \lambda\sqrt{\log n/\log\log n}$ for a sufficiently small constant $\lambda > 0$. By applying Lemmas 4.3, 4.5, and 4.6 in sequence, we derive that the decision problem can be solved in $O(n\tau^{c-1} + mn(\log \tau)/\tau)$. The second term dominates the first if m is sufficiently large, that is, if $m = \Omega(\tau^c/\log \tau)$. Filling in our choice for τ results in $m = \Omega(\log^{c/2} n/((\log\log n)^{c/2}(\log\log n - \log\log\log n)))$. Discarding the divisor, this can be simplified to $m = \Omega(\log^{c/2} n)$. From the proof of Lemma 4.4, we know that $c = 6$, thus resulting in the posed restriction on m. □

4.5 Improved bound on word RAM

We now explain how the execution time of our algorithm can be improved if our computational model allows for constant time table-lookup. We use the same τ as above (up to a constant factor). However, we change a number of things. "Signatures" are represented differently and the data structure to obtain combinatorial reachability structures is changed accordingly. Furthermore, we aggregate elementary boxes into clusters and determine "partial door-orders" for multiple boxes at the same time. Finally, we traverse the free-space diagram based on the clusters to decide $d_{\mathrm{F}}(P, Q) \leqslant \varepsilon$.

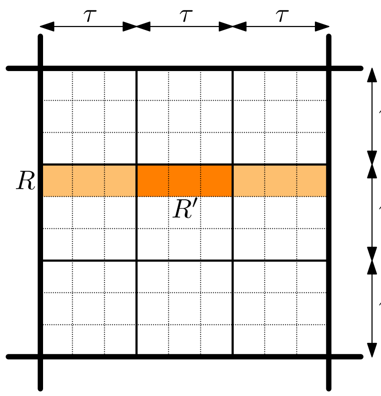


Figure 4.7 A cluster consists of $\tau \times \tau$ elementary boxes, thus of $\tau^2 \times \tau^2$ cells. A row R and its corresponding R' for the central elementary box are indicated in orange.

Clusters and extended signatures. We introduce a second level of aggregation in the free-space diagram (see Figure 4.7): a *cluster* is a collection of $\tau \times \tau$ elementary boxes, that is, $\tau^2 \times \tau^2$ cells in $\mathrm{FSD}(P, Q)$. Let R be a row of cells in $\mathrm{FSD}(P, Q)$ of a certain cluster. As before, the row R corresponds to an edge e on Q and a subcurve P' of P with τ^2 edges. We associate with R an ordered set $Z = \langle e_0, z'_0, z_1, z'_1, z_2, z'_2, \ldots, z_k, z'_k, e_1 \rangle$ with $2k + 3$ elements. Here, k is the number of intersections of e with the circles with radius ε centered at the τ vertices of P' (all but the very first). Hence, k is bounded by 2τ and $|Z|$ is bounded by $4\tau + 3$. The order of Z indicates the order of these intersections with e directed along Q. Elements e_0 and e_1 represent the endpoints of e and take a special role. In particular, these are used to represent closed doors and to snap open doors to edge e. The elements z'_i are placeholders for the positions of the endpoints of the reach-doors: z'_0 represents a possible reach-door endpoint between e_0 and z_1, the element z'_1 is an endpoint between z_1 and z_2, etc.

Consider a row R' of an elementary box inside a row R of a cluster; let e denote the edge of Q that corresponds to R'. The *door-index* of R' is an ordered set $\langle s_0, t_0, \ldots, s_\tau, t_\tau \rangle$ of size $2\tau + 2$. Similar to a door-order, elements s_0 and t_0 represent the reach-door at the leftmost boundary of R'; the elements s_i and t_i ($1 \leqslant i \leqslant \tau$) represent the door at the right boundary of the i^{th} cell in R'. However, instead of rearranging the set to indicate relative positions, the elements s_i and t_i simply refer to an element in Z. If the door is open, they refer to the intersections with e (possibly snapped to e_0 or e_1). If the door is closed, s_i is set to e_1 and t_i is set to e_0. The elements s_0 and t_0 are special: they represent the reach-door and refer to one of the elements z'_i. A *partial door-index* is a door-index without s_0 and t_0. The advantage of a door-index over a door-order is that the reach-door is always at the start. Hence, completing a partial door-index to a full door-index can be done in constant time. Since a door-index has size $2\tau + 2$, the number of possible door-indices for R' is $\tau^{O(\tau)}$.

We define the door-indices for the columns analogously. We concatenate the door-indices for the rows and the columns to obtain the *indexed signature* for an elementary box. Similarly, we define the *partial indexed signature*. The total number of possible indexed signatures remains $\tau^{O(\tau^2)}$.

For each possible partial indexed signature Σ we build a lookup table T_Σ as follows: the input is a word with 4τ fields of $O(\log \tau)$ bits each. Each field stores the positions in Z of the endpoints of the ingoing reach-doors for the elementary box: 2τ fields for the left side, 2τ fields for the lower side. The output consists of a word that represents the indices for the elements in Z that represent the outgoing reach-doors for the upper and right boundary of the box. Thus, the input of T_Σ is a word of $O(\tau \log \tau)$ bits, and T_Σ has size $\tau^{O(\tau)}$. Hence, for all partial indexed signatures combined, the size is $\tau^{O(\tau^2)} = o(n)$ by our choice of τ.

Preprocessing a given input. During the preprocessing of the given input, we use *superstrips* consisting of τ strips. That is, a superstrip is a column of clusters and consists of τ^2 columns of the free-space diagram. Lemma 4.4 still holds, albeit with a larger constant c (originally, $c = 6$). Essentially, τ has become τ^2 in this analysis; hence, the construction time and space of the data structure become $O((\tau^2)^6) = O(\tau^{12})$. The query time remains logarithmic in τ as $O(\log \tau^2) = O(\log \tau)$. The data structure gets as input a query edge e, and it returns in $O(\log \tau)$ time a word that contains τ fields. Each field represents the partial door-index for e in the corresponding elementary box and thus consists of $O(\tau \log \tau)$ bits. Hence, the word size is $O(\tau^2 \log \tau) = O(\log n)$ by our choice of τ. Thus, the total time for preprocessing the input, building a data structure for each of the $O(m/\tau^2)$ superstrips and for processing all n rows is $O(m/\tau^2\ (\tau^c + n \log \tau)) = O(m\tau^{c-2} + mn(\log \tau)/\tau^2)$. We repeat this procedure for the horizontal superstrips, taking $O(n\tau^{c-2} + mn(\log \tau)/\tau^2)$ time.

We now have parts of the partial indexed signature for each elementary box packed into different words. To obtain the partial indexed signature, we need to rearrange the information such that the partial door-indices of the rows in one elementary box are in a single word. This corresponds to computing a transposition of a matrix, as is illustrated in Figure 4.8. For this, we need the following lemma, which was proven—in slightly different form—by Thorup [165, Lemma 9].

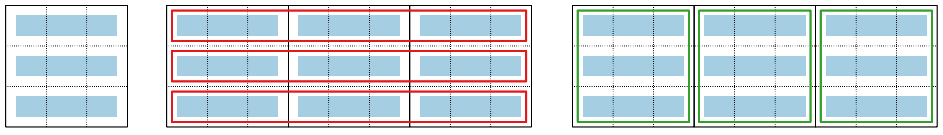


Figure 4.8 (a) A field represents the partial door-index of a row in an elementary box. (b) The fields are grouped into words per row in a cluster. (c) Transposition yields the desired organization, where a word represents the partial door-indices of rows in an elementary box.

Lemma 4.8. *Let X be a sequence of τ words that contain τ fields each, so that X can be interpreted as a $\tau \times \tau$ matrix. Then we can compute in time $O(\tau \log \tau)$ on a word RAM a sequence Y of τ words with τ fields each that represents the* transposition *of X.*

Proof. The algorithm is recursive and solves the following more general problem. Let X be a sequence of a words that represents a sequence M of b different $a \times a$ matrices, such that the j^{th} word in X contains the fields of the j^{th} row of each matrix in M from left to right. Compute a sequence of words Y that represents the sequence M' of the transposed matrices in M.

The recursion works as follows. If $a = 1$, there is nothing to be done. Otherwise, we split X into the sequence X_1 of the first $a/2$ words and the sequence X_2 of the remaining words. X_1 and X_2 now represent a sequence of $2b$ matrices, each of size $(a/2) \times (a/2)$, which we transpose recursively. After the recursion, we put the $(a/2) \times (a/2)$ submatrices back together in the obvious way. To finish, we need to transpose the off-diagonal submatrices. This can be done simultaneously for all matrices in time $O(a)$, by using appropriate bit-operations (or table lookup).

Hence, the execution time obeys a recursion of the form $T(a) = 2T(a/2) + O(a)$, giving $T(a) = O(a \log a)$, as desired. □

By applying the lemma to the words that represent τ consecutive rows in a superstrip, we obtain the partial door-indices of the rows for each elementary box. This takes total time $O((m/\tau^2)\cdot(n/\tau)\cdot\tau \log \tau) = O(mn(\log \tau)/\tau^2)$ for the vertical superstrips and, similarly, $O(mn(\log \tau)/\tau^2)$ for the horizontal superstrips. By using an appropriate lookup table to combine the partial door-indices of the rows and columns, we obtain the partial indexed signature for each elementary box in total time $O(n\tau^{c-2} + mn(\log \tau)/\tau^2)$.

The actual computation. We traverse the free-space diagram cluster by cluster (recall that a cluster consists of $\tau \times \tau$ elementary boxes). The clusters are processed column by column from left to right, and inside each column from bottom to top. Before processing a cluster, we walk along the left and lower boundary of the cluster to determine the incoming reach-doors. This is done by performing a binary search for each box on the boundary, and determining the appropriate elements z'_i that correspond to the incoming reach-doors. Using this information, we assemble the appropriate words that represent the incoming information for each elementary box. Since there are mn/τ^4 clusters, this step requires time $O((mn/\tau^4)\tau^2 \log \tau) = O(mn(\log \tau)/\tau^2)$. We then process the elementary boxes inside the cluster, in a similar fashion. Now, however, we can process each elementary box in constant time through a single table lookup, so the total time is $O(mn/\tau^2)$. Hence, including the preprocessing time, the total execution time of our algorithm is $O(n\tau^{c-2} + mn(\log \tau)/\tau^2)$. By our choice of $\tau = \lambda\sqrt{\log n/\log\log n}$ for a sufficiently small $\lambda > 0$, we obtain the following theorem.

Theorem 4.9. *On a word RAM machine, the decision version of the Fréchet problem can be solved in $O(mn(\log\log n)^2/\log n)$ time, assuming $m \leqslant n$ and $m = \Omega(\log^6 n)$.*

Note that we again require m to be sufficiently large, such that the τ^{c-2} term is dominated by $m(\log \tau)/\tau^2$. As c increased from 6 to 12, this bound has increased slightly.

4.6 Solving the optimization problem

The optimization version of the Fréchet problem, i.e., computing the Fréchet distance, can be done in $O(mn \log n)$ time using parametric search with the quadratic-time algorithm for solving the decision version as a subroutine [10]. We showed that the decision problem can be solved in subquadratic time. However, this does not directly yield a faster algorithm for the optimization problem: if the execution time of the decision problem is $T(m, n)$, parametric search gives an $O((T(m, n) + mn) \log n)$ time algorithm [10]. There is an alternative randomized algorithm by Har-Peled and Raichel [98]. Their algorithm also requires $O((T(m, n) + mn) \log n)$ time, but below we adapt it to obtain the following lemma.

Lemma 4.10. *The Fréchet distance of two curves with m and n vertices (assuming $m \leqslant n$) can be computed by a randomized algorithm in $O(n^2 + mn2^{\alpha(n)} + T(m, n) \log n)$ expected time, where $T(m, n)$ is the time for the decision problem.*

Recall that the possible values of the Fréchet distance are limited to a set of critical values (see Section 2.4.2 or [10]).

(A) Point $(0, 0)$ or (m, n) becomes part of the free space; the associated critical values are $\|P(0) - Q(0)\|$ and $\|P(m) - Q(n)\|$ respectively.
(B) A new passage is created on a cell boundary; the associated critical value is the minimal distance between the vertex and edge corresponding to the boundary.
(C) A new passage is created from one cell boundary to another within the same row or column. This corresponds to two vertices of one curve and an edge of the other. The associated critical value is the distance from one of the vertices to the intersection of the edge and the bisector of the vertices.

We extend the type-A events to include all vertex-vertex pairs, not just the endpoints of the two curves. If we also include type-C events (vertex-vertex-edge tuples) with no intersection, we can sample a critical value uniformly at random in constant time. We refer to these events and values as *extended critical events* and *extended critical values*. The algorithm now works as follows (see Har-Peled and Raichel [98], or the arXiv version [99] for more details): first, we sample a set S of $K = 4mn$ extended critical values uniformly at random. Next, we find $a, b \in S$ such that the Fréchet distance lies between a and b and such that $[a, b]$ contains no other value from S. In the original algorithm this is done by sorting S and performing a binary search using the decision version. Using linear-time median-finding instead, this step can be done in $O(K + T(m, n) \log K)$ time. Alternatively, the execution time of this step could be reduced by picking a smaller K. However, this does not improve the final bound, since it is dominated by a $O(mn2^{\alpha(n)})$ component. The interval $[a, b]$ with high probability contains only a small number of the remaining extended critical values. For $K = 4n^2$ (i.e., $m = n$) the probability that $[a, b]$ has more than $2cn \ln n$ extended critical values is at most $1/n^c$ [99, Lemma 6.2]. Using the same ideas, we prove this result in its more general form.

Lemma 4.11. *Let S be a set of $4mn$ randomly sampled extended critical values for two curves of m and n edges. Let $a, b \in S$ such that the Fréchet distance lies between a and b and such that $[a, b]$ contains no other value from S. Assuming $m \leqslant n$, the probability that $[a, b]$ contains more than $2cn \ln n$ extended critical values is at most $1/n^c$.*

Proof. The number of extended critical values of type A is $(m + 1)(n + 1)$; of type B is $m(n+1)+(m+1)n$; of type C is $mn(m+n)/2-1$. Hence, assuming sufficiently large n, the total number of extended critical values is $N = 3mn+2m+2n+mmn/2+mnn/2 \leqslant 3mn + 2m + 2n + mnn \leqslant 2mnn$. Let d^* denote the (extended) critical value that determines the Fréchet distance. Let U^- denote the largest $cn \ln n$ extended critical values that are smaller than or equal to d^*; similarly, U^+ denotes the smallest $cn \ln n$ extended critical values that are larger than or equal to d^*. The derivation below proves that the probability that S contains no value from U^- is at most $1/n^c$.

$$\left(1 - \frac{|U^-|}{N}\right)^{|S|} \leqslant \left(1 - \frac{cn \ln n}{2mnn}\right)^{4mn} = \left(1 + \frac{-2c \ln n}{4mn}\right)^{4mn} \leqslant e^{-2c \ln n} = \frac{1}{n^{2c}}$$

The same holds for U^+. Hence, the probability that S contains a value of both U^- and U^+ is at least $(1 - (1/n^{2c}))^2 = 1 - 2/n^{2c} + 1/n^{4c} \geqslant 1 - 2/n^{2c}$. If S contains a value from both U^- and U^+, then the number of extended critical values in $[a, b]$ is bound by their combined size, that is, by $2cn \ln n$. Thus, the probability that $[a, b]$ contains more critical values is at most $2/n^{2c}$. This bound is slightly stronger than the one given by Har-Peled and Raichel [98]; using $n^{2c} \geqslant 2n^c$, we obtain the desired result. □

Let K' denote the number of extended critical values in the interval $[a, b]$. The remainder of the algorithm determines the critical values in the interval $[a, b]$ and performs another binary search. As the critical values are included in the extended critical values, this number of critical values is also bounded by K'. Excluding the time to determine the critical values, this therefore takes $O(K' + T(m, n) \log K')$ time with median-finding. Thus, the crucial part is to determine the critical values.

In $O(mn)$ time we can check for all possible type-A and type-B events whether the corresponding critical value lies in $[a, b]$. It remains to determine the critical values corresponding to type-C events. These critical values are found by a standard sweepline algorithm. To this end, take an edge e of P and the vertices of Q. The sweep starts with circles of radius a around the vertices of Q, and it increases the radii until they reach b. During this sweep, the algorithm maintains the order in which the circle arcs intersect e. A critical value of a type-C event corresponds to the event that two different circles intersect e in the same point. Besides these events the sweepline algorithm requires the following events: (a) a circle intersects one of the vertices of e; (b) a circle intersects e for the first time. These correspond exactly to extended type-A or type-B critical values involving e or a vertex of e. Hence, if we perform a sweep for each edge of P (and similarly for Q), the total number of events is $O(K')$. The overall execution time of all sweeps ignoring the time for initialization is $O(K' \log n)$.

It remains to show that we can quickly find the initial order in which the circles intersect e. First, compute the arrangement $\mathcal{A}$ of circles with radius a around the vertices of Q. This takes $O(n^2)$ time [54]. To find the intersection order, traverse in $\mathcal{A}$ the zone of the line ℓ spanned by e. The time for the traversal is bounded by the complexity of the zone. Since the circles pairwise intersect at most twice and ℓ intersects each circle only twice, the complexity of the zone is $O(n2^{\alpha(n)})$ [159, Theorem 5.11]. As we need to construct $\mathcal{A}$ only once and $m \leqslant n$, this adds a total of $O(n^2 + mn2^{\alpha(n)})$ to the execution time. Hence, the overall time is $O(T(m,n)\log(n) + n^2 + mn2^{\alpha(n)} + K' \log n)$. The case $K' > 8n \ln n$ has probability less than $1/n^4$, and we always have $K' = O(mn^2)$. Thus, the last term adds $o(1)$ to the expected running time. Given $K' \leqslant 8n \ln n$, the last term is $O(n \log^2 n)$ and is subsumed by the $O(n^2)$ term. Lemma 4.10 follows.

Theorem 4.12 now results from Lemma 4.10, Theorem 4.7, and Theorem 4.9.

Theorem 4.12. *The Fréchet distance of two open polygonal curves with m and n edges ($m \leqslant n$) can be computed by a randomized algorithm in $O(n^2+mn\sqrt{\log n}(\log\log n)^{3/2})$ time on a pointer machine (assuming $m = \Omega(\log^3 n)$) and in time $O(n^2+mn(\log\log n)^2)$ on a word RAM (assuming $m = \Omega(\log^6 n)$).*

We observe that this algorithm is faster than Alt and Godau's $O(mn \log n)$-time algorithm if $m = \omega(n/\log n)$, that is, if m is near-linear in n.

4.7 Decision trees

Our results also have implications for the decision-tree complexity of the Fréchet problem. Since in that model we account only for comparisons between the input elements, the preprocessing comes for "free". As a result, the size of the elementary boxes can be increased. For this section, we assume that the two curves have equal complexity (i.e., $m = n$). This is the most interesting case from a theoretic point of view in light of Alt's conjecture.

Before we consider the continuous Fréchet problem, we first note that the techniques of Agarwal *et al.* [3] can be used to obtain a similar result for the *discrete* Fréchet problem: suppose we have two sequences $P = \langle p_1, p_2, \dots, p_n\rangle$ and $Q = \langle q_1, q_2, \dots, q_n\rangle$. For $\delta > 0$, we define a directed graph G_δ with vertex set $P \times Q$. In G_δ, there is an edge between two vertices (p_i, q_j), (p_i, q_{j+1}) if and only if both $d(p_i, q_j) \leqslant \delta$ and $d(p_i, q_{j+1}) \leqslant \delta$. The condition is similar for an edge between vertices (p_i, q_j) and (p_{i+1}, q_j), and vertices (p_i, q_j) and (p_{i+1}, q_{j+1}). There are no further edges in G_δ. Now the problem is to find the smallest δ for which G_δ has a path from (p_1, q_1) to (p_n, q_n). For the discrete Fréchet problem we obtain the following bound, where we use $\widetilde{O}(\cdot)$ to indicate $O(\cdot)$ up to polylogarithmic factors.

Theorem 4.13. *There is an algebraic computation tree for the* discrete *Fréchet problem of depth $\widetilde{O}(n^{4/3})$.*

Proof. We first discuss the decision problem, where we are given curves P, Q and a value δ, and we need to decide whether we can reach (p_n, q_n) from (p_1, q_1). For the discrete case, the analogue of the reachable free space is just an $n \times n$ boolean matrix M. In this matrix, the bit in the i^{th} column and the j^{th} row indicates whether the pair (p_i, q_j) can be reached from (p_1, q_1) in G_δ.

As shown by Katz and Sharir [112], we can compute a representation of the set of points (p_i, q_j) with $||p_i - q_j|| \leqslant \delta$ in $\widetilde{O}(n^{4/3})$. This information suffices to complete M without further comparisons. As shown by Agarwal *et al.* [3], one can then solve the optimization problem at the cost of another $O(\log n)$-factor, which is absorbed into the $\widetilde{O}$-notation. □

Given our results above, we prove an analogous statement for the continuous Fréchet distance. This is formalized in the following lemma.

Theorem 4.14. *There exists an algebraic decision tree for the Fréchet problem (decision version) of depth $O(n^{2-\alpha})$, for a fixed constant $\alpha > 0$.*

Proof. We reconsider the steps of our algorithm. The only phases that actually involve the input are the second preprocessing phase and the traversal of the elementary boxes. The reason of our choice for τ was to keep the time for the first preprocessing phase polynomial. This is no longer a problem. By Lemma 4.5 and Lemma 4.6, the remaining cost is bounded by $O(n\tau^{c-1} + n^2(\log \tau)/\tau)$, where c is the constant from Lemma 4.4. Choosing $\tau = n^{1/c}$, we get a decision tree of depth $O(n \cdot n^{\frac{c-1}{c}} + n^2 \log n/n^{1/c})$. This is $O(n^{2-(1/c)} \log n) = O(n^{2-\alpha})$ for, say, $\alpha = 1/2c$. □

We now briefly consider Alt's conjecture. It was conjectured that no decision tree with depth $O(n^{2-\alpha})$ exists for 3SUM [138]. Assuming linear time reductions, this conjecture in combination with our result would imply that Alt's conjecture is false. However, recently it has been proven that 3SUM admits an algebraic decision tree of depth $O(n^{3/2}\sqrt{\log n})$ [93]. Thus, Alt's conjecture remains an open problem.

4.8 Conclusion

In this chapter we have improved upon the long-standing quadratic upper bound for the decision version of the Fréchet problem. We have given an $O(mn(\log \log n)^{3/2}/\sqrt{\log n})$-time algorithm to decide $d_{\mathrm{F}}(P, Q) \leqslant \varepsilon$ for two curves of complexity m and n with $m \leqslant n$. We have shown how this decision algorithm can be used to compute the Fréchet distance in $O(n^2 + mn\sqrt{\log n}(\log \log n)^{3/2})$ time. If we allow constant-time table-lookup, we obtain an execution time of $O(n^2 + mn(\log \log n)^2)$, in close reach of $O(n^2)$ for $m = n$. Finally, we proved that the decision problem has an algebraic decision tree of depth $O(n^{2-\alpha})$, for some $\alpha > 0$ and where n is the number of vertices of the curves.

This leaves us with intriguing open research questions. Since 3SUM is known to have a decision tree with subquadratic depth [93], our bound has no implications on Alt's conjecture. It thus remains an open problem whether the decision problem is 3SUM-hard. Assuming the Strong Exponential Time Hypothesis, no subquadratic-time algorithm exists [30]. This indicates a discrepancy between the decision tree and the uniform complexity. Can we establish a connection between the Fréchet distance and other problems that exhibit this discrepancy, such as MIN-PLUS-CONVOLUTION? On the other hand, this lower bound for the uniform complexity also shows that our algorithm is tight, up to subpolynomial factors. However, for the optimization, it remains an open problem whether we can devise a quadratic-time algorithm. Can we develop such an algorithm on the word RAM, that is, with constant-time table-lookup?

Finally, we identify two limitations of our current methods. The first limitation is that the curves must reside in the two-dimensional plane. This two-dimensionality is essential in order to apply dual transformations (e.g. for the proof of Lemma 4.4). This is in contrast to the method by Agarwal *et al.* [3] which generalizes to higher dimensions. Can we extend our results to also work with higher-dimensional polygonal curves?

The second limitation is that for both the decision and optimization algorithm, m must be sufficiently large in order for the bound to improve upon the known results. Our decision algorithms assume that m is at least polylogarithmic in n. Can we remove this requirement by finding other means of computing the door-orders of rows? Furthermore, to compute the Fréchet distance, we need an arrangement of circles on the vertices of Q, leading to an $O(n^2)$ term in the execution time. This means that the algorithm is faster than Alt and Godau's algorithm [10] only if $m = \omega(n/\log n)$. Is it possible to improve upon the $O(mn \log n)$ bound for any combination of values for m and n?

Chapter 5

Locally Correct Fréchet Matchings

The Fréchet distance results in a single number that indicates the similarity between curves. However, a "description" of the similarity is often required, rather than only a quantification. Such a description should precisely describe which parts of the curves correspond to another. In this chapter we investigate Fréchet matchings—a matching that results in the Fréchet distance—for the purpose of describing similarity. A Fréchet matching exactly describes a point-to-point correspondence between the curves and is thus inherently a description of similarity. There are applications that directly use a matching, for example, to map a GPS track to a street network [176] or to morph between the curves [74]. In such situations a "good" matching is important. Furthermore, we believe that many applications of the (discrete) Fréchet distance, such as protein alignment [181] and detecting patterns in movement data [33], would profit from good Fréchet matchings. For simplification and schematization, a large distance may be unavoidable in some places along a given outline. However, that does not imply we need to accept unnecessarily large distances in other places. Therefore, a good matching may allow us to assess the resemblance more accurately.

There are often many different Fréchet matchings for two curves. Not all of these matchings capture the similarity between the curves well (see Figure 5.1). This is caused by the Fréchet distance being defined to minimize the maximal distance: Fréchet matchings are relatively unrestricted in places where the curves are locally much closer than the Fréchet distance. To alleviate this drawback, integral or average Fréchet distances have been considered. However, these approaches introduce other problems or undesirable behavior [44] (see also Section 3.1). Therefore, we desire another way to distinguish the quality of Fréchet matchings, based on how well they describe the similarity.

We restrict the set of Fréchet matchings to "natural" matchings by introducing *locally correct* Fréchet matchings: matchings that for any two matched subcurves are again a Fréchet matching on these subcurves. This is a strictly stronger concept: any locally

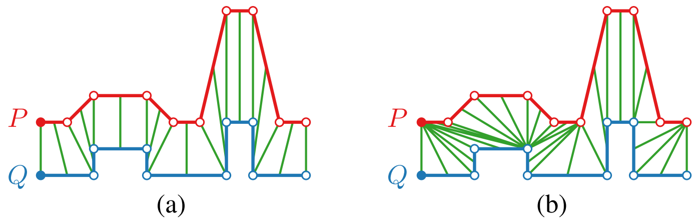


Figure 5.1 Two Fréchet matchings for curves P and Q. Matching (a) describes the similarity more accurately than matching (b).

correct Fréchet matching is a Fréchet matching, but not vice versa. In Section 5.2 we prove that there exists such a locally correct Fréchet matching for any two polygonal curves. Based on this proof we describe in Section 5.3 an $O((m+n)mn \log mn)$-time algorithm to compute such a matching, where m and n denote the complexity of the two polygonal curves. We consider the discrete Fréchet distance in Section 5.4. Under this metric we give an algorithm that computes a locally correct matching in $O(mn)$ time and $O(\min\{m, n\})$ space.

Related work. The Fréchet distance has received significant attention in a range of variants and applications. In the previous chapter we already listed a number of results related to computing or using the Fréchet distance. Here we focus on approaches that change or restrict the allowed matchings.

Efrat *et al.* [74] introduced Fréchet-like metrics, the geodesic width and link width, to restrict to matchings suitable for curve morphing. Their method is suitable only for non-intersecting curves. Moreover, geodesic width and link width do not resolve the problem illustrated in Figure 5.1: both matchings also have minimal geodesic width and minimal link width. Maheshwari *et al.* [120] studied a restriction called "speed limits", which may exclude all Fréchet matchings and may cause undesirable effects near "outliers" (see Figure 5.2). Buchin *et al.* [32] describe a framework for restricting Fréchet matchings, which they illustrate by restricting slope and path length. Note that the former corresponds to speed limits. We briefly discuss such constraints at the end of Section 5.3.

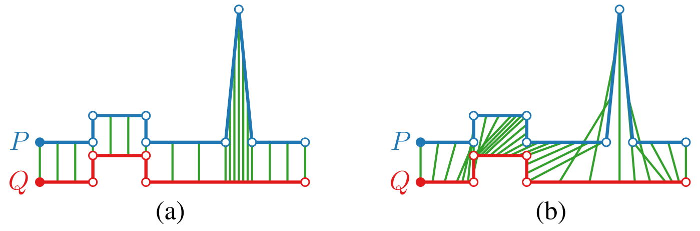


Figure 5.2 Two Fréchet matchings. (a) A locally correct matching. (b) Imposing speed limits may yield a matching that is not locally correct.

Another approach lies in only partially matching the two curves. This allows for a number of interpretations. Alt and Godau [10] consider matching one curve to a subcurve of another; their $O(mn \log n)$-time decision algorithm was later improved to $O(mn)$ by Maheshwari *et al.* [121]. However, this proves too restricted if outliers occur on both curves or are spread over multiple locations. Therefore, Buchin *et al.* [39] instead maximize the length of the matched subcurves, constrained by a maximal Fréchet distance. This allows for fully ignoring parts of the curves. Driemel and Har-Peled [70] allow shortcuts between vertices, but a shortcut is replaced by a single line segment. This line segment must then still be matched to the other curve. Buchin *et al.* [45] showed that this problem is NP-hard if any shortcut (i.e., between points along the edges of the curves) is allowed and provide a 3-approximation algorithm.

5.1 Definition

Consider two open polygonal curves $P = \langle p_0, \ldots, p_m \rangle$ and $Q = \langle q_0, \ldots, q_n \rangle$ with m and n edges respectively. Recall that the Fréchet distance $d_{\mathrm{F}}(P, Q)$ is defined as

$$\min_{(\sigma,\theta)\in\mathcal{M}} \max_{x\in[0,1]} \|P(\sigma(x)) - Q(\theta(x))\|,$$

where $\mathcal{M}$ contains all matchings (homeomorphisms and limits thereof) between P and Q. A matching μ describes two reparametrizations, σ and θ, for P and Q respectively. We denote by $P_\sigma(t)$ the actual location according to reparametrization σ, that is, $P_\sigma(t) = P(\sigma(t))$. By $P_\sigma[a, b]$ we denote the subcurve of P in between $P_\sigma(a)$ and $P_\sigma(b)$. Similarly, we denote the reparametrized curve Q by Q_θ.

We use $d_\mu(x)$ to represent the Euclidean distance between the two matched points, that is, $d_\mu(x) = \|P_\sigma(x) - Q_\theta(x)\|$. The maximum distance over a range is denoted by $d_\mu[a, b] = \max_{a\leqslant t\leqslant b} d_\mu(t)$. Note that these notations use P and Q implicitly. Thus, we may write the definition of the Fréchet distance as

$$d_{\mathrm{F}}(P, Q) = \min_{\mu\in\mathcal{M}} d_\mu[0, 1].$$

A *Fréchet matching* is a matching μ that realizes the Fréchet distance, that is, $d_\mu[0, 1] = d_{\mathrm{F}}(P, Q)$ holds.

As illustrated in Figure 5.1, not all Fréchet matchings describe the similarity equally well. To distinguish between matchings, we introduce *locally correct* Fréchet matchings, for which the matching between any two matched subcurves is a Fréchet matching. This is formalized as follows.

Definition 5.1. [LOCAL CORRECTNESS] *Given two polygonal curves P and Q, a matching $\mu = (\sigma, \theta)$ is* locally correct *if for all a, b with $0 \leqslant a \leqslant b \leqslant 1$*

$$d_\mu[a, b] = d_F(P_\sigma[a, b], Q_\theta[a, b]).$$

Observe that the above condition must also hold for $a = 0$ and $b = 1$: $d_\mu[0,1] = d_\mathrm{F}(P_\sigma[0,1], Q_\theta[0,1]) = d_\mathrm{F}(P,Q)$. That is, the maximal distance according to a locally correct Fréchet matching is indeed the Fréchet distance. Thus, a locally correct Fréchet matching must be a Fréchet matching as well. However, not every Fréchet matching is locally correct; indeed, Figure 5.1(a) illustrates a locally correct Fréchet matching whereas Figure 5.1(b) does not. The question arises whether a locally correct matching always exists and if so, how to compute it. We address these questions in the upcoming sections.

5.2 Existence

In this section we resolve the first question, whether a locally correct Fréchet matching exists. That is, we prove the following theorem.

Theorem 5.2. *For any two open polygonal curves P and Q, there exists a locally correct Fréchet matching.*

We prove Theorem 5.2 by induction on the number of edges in the curves. First, we present the lemmas for the two base cases: one of the two curves is a point and both curves are line segments. In the following, m and n again denote the number of edges of P and Q, respectively.

Lemma 5.3. *For two polygonal curves P and Q with $m = 0$, a locally correct matching is $\mu = (\sigma, \theta)$, where $\sigma(t) = 0$ and $\theta(t) = t \cdot n$ for $0 \leqslant t \leqslant 1$.*

Proof. Since $m = 0$, P is just a single point, p_0. The Fréchet distance between a point and a curve is the maximal distance between the point and any point on the curve: $d_\mathrm{F}(p_0, Q_\theta[a,b]) = d_\mu[a,b]$. This implies that the matching μ is locally correct. □

Lemma 5.4. *For two polygonal curves P and Q with $m = n = 1$, a locally correct matching is $\mu = (\sigma, \theta)$, where $\sigma(t) = \theta(t) = t$ for $0 \leqslant t \leqslant 1$.*

Proof. As both curves consist of a single edge, the free-space diagram of P and Q is a single cell and thus the free space is a convex area for any value of ε [10]. Since $\mu = (\sigma, \theta)$ is linear, we know that $d_\mu[a,b] = \max\{d_\mu(a), d_\mu(b)\}$: if there would be a t with $a < t < b$ such that $d_\mu(t) > \max\{d_\mu(a), d_\mu(b)\}$, then the free space at $\varepsilon = \max\{d_\mu(a), d_\mu(b)\}$ would not be convex. Since $d_\mu[a,b] = \max\{d_\mu(a), d_\mu(b)\} \leqslant d_\mathrm{F}(P_\sigma[a,b], Q_\theta[a,b]) \leqslant d_\mu[a,b]$, we conclude that μ is locally correct. □

For induction, we split the two curves based on critical events (see Figure 5.3). To properly use the induction hypothesis, we require that one of the subcurves has strictly less edges than the full curve; the other may not increase its number of edges. The only events that would cause such a problem are the type-A events and the type-B events on the left or bottom boundary of cell $C(0,0)$ or on the right or top boundary of cell $C(m-1, n-1)$. As it turns out, we can simply ignore these events. Roughly speaking, type-A events do not give any choice as they correspond to the endpoints of the two curves—we must match these points. In addition, the ignored type-B events are subsumed by the type-A

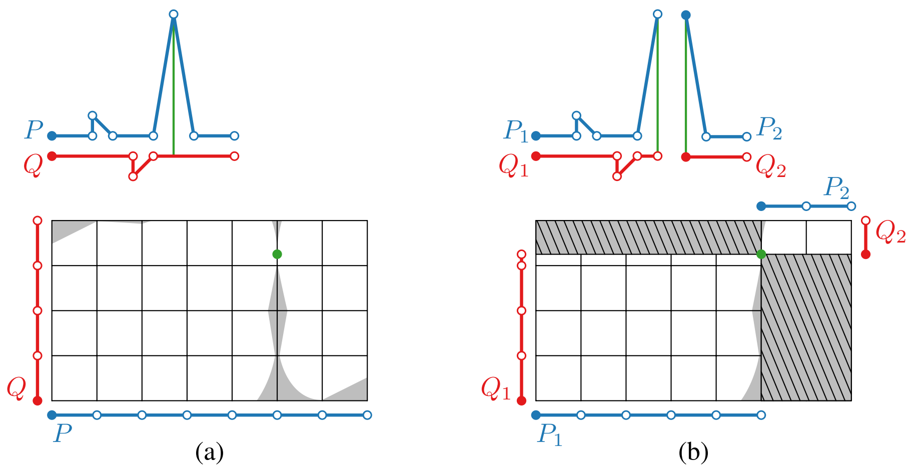


Figure 5.3 (a) Curves with the free-space diagram for $\varepsilon = d_{\mathrm{F}}(P, Q)$ and the corresponding realizing event. (b) The event splits each curve into two subcurves. The hatched areas of the free-space diagram are eliminated by the split.

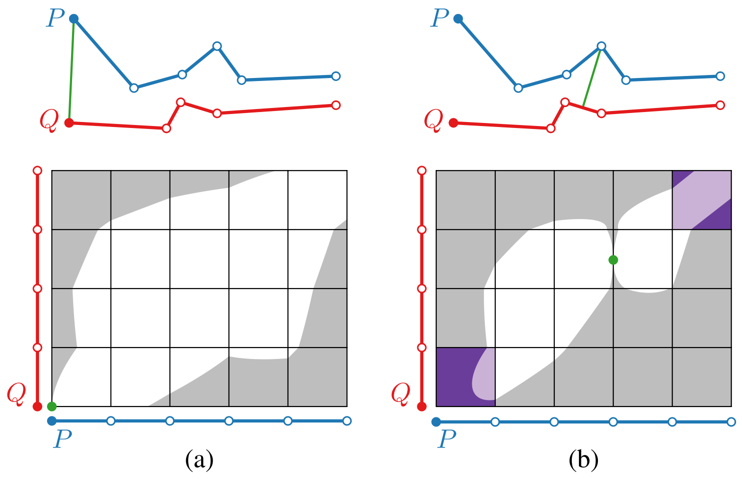


Figure 5.4 (a) Splitting on a type-A event does not yield a reduction in complexity of the subproblems. (b) By "ignoring" $C(0, 0)$ and $C(m - 1, n - 1)$ (marked in purple), we find an event that does properly split the problem into smaller subproblems.

events. Thus, we call a free-space diagram *feasible* at value ε, if a bimonotone path exists in the free space from some point on the top or right boundary of $C(0,0)$ to some point on the bottom or left boundary of cell $C(m-1,n-1)$. This is illustrated in Figure 5.4.

To provide some intuition as to why these events are not essential, we argue as follows. Suppose that ε is the lowest value at which the free-space diagram is feasible: thus a path exists between cell $C(0,0)$ and $C(m-1,n-1)$. By going straight from $(0,0)$ to the start of the path and from the end of the path to (m,n), we obtain a bimonotone path that describes a matching. Due to the convexity of the free space in a cell, we know that either the maximal distance is given by ε, the point $(0,0)$ (the start of the two curves) or (m,n) (the end of both curves). As the last are forced in any matching, we can thus conclude that the obtained matching is indeed a Fréchet matching. This justifies that we may indeed ignore the indicated events. As we prove below, this allows us to in fact obtain a locally correct Fréchet matching.

A *realizing event* is a critical event at the minimal value ε such that the corresponding free-space diagram is feasible. We call events *concurrent* if they occur at the same value of ε. Let $\mathcal{E}$ denote the set of concurrent realizing events for two curves. A *realizing set* E_r is a subset of $\mathcal{E}$ such that the free space admits a bimonotone path from cell $C(0,0)$ to $C(m-1,n-1)$ without using an event in $\mathcal{E}\backslash E_\mathrm{r}$. Note that a realizing set cannot be empty. When $\mathcal{E}$ contains more than one realizing event, some may be "insignificant": they are never required to actually make a path in the free-space diagram. A realizing set is *minimal* if it does not contain a strict subset that is a realizing set. Such a minimal realizing set contains only "significant" events.

Lemma 5.5. *For two polygonal curves P and Q with $m > 1$ and $n \geqslant 1$, there exists a minimal realizing set.*

Proof. Let $\mathcal{E}$ denote the nonempty set of all concurrent events at the minimal critical value: $\mathcal{E}$ is a realizing set. By definition, the empty set cannot be a realizing set. Hence, $\mathcal{E}$ contains a minimal realizing set. □

The following lemma directly implies that a locally correct Fréchet matching always exists. Informally, it states that curves have a locally correct matching that is "closer" (except in cell $C(0,0)$ or $C(m-1,n-1)$) than the distance of their realizing set. Furthermore, this matching is linear inside every cell. The lemma is proven using induction: a minimal realizing set is used to split the curves into pieces; combining the locally correct matchings of the pieces results in a single, locally correct matching.

Lemma 5.6. *If the free-space diagram of two polygonal curves P and Q is feasible at value ε, then there exists a locally correct Fréchet matching $\mu = (\sigma, \theta)$ such that $d_\mu(t) \leqslant \varepsilon$ for all t with $\sigma(t) \geqslant 1$ or $\theta(t) \geqslant 1$, and $\sigma(t) \leqslant m-1$ or $\theta(t) \leqslant n-1$. Furthermore, μ is linear in every cell.*

Proof. We prove this by induction on $m+n$. The base cases ($m = 0$, $n = 0$, and $m = n = 1$) follow from Lemma 5.3 and Lemma 5.4. The matchings prescribed by these lemmas are indeed linear.

For induction, we assume that $m \geqslant 1$, $n \geqslant 1$, and $m + n > 2$. By Lemma 5.5, a minimal realizing set E_r exists for P and Q, say at value ε_r. The set contains realizing events $e_1, \ldots, e_k$ ($k \geqslant 1$), numbered in lexicographic order. By definition, $\varepsilon_\mathrm{r} \leqslant \varepsilon$ holds. Suppose that E_r splits curve P into $P_1, \ldots, P_{k+1}$ and curve Q into $Q_1, \ldots, Q_{k+1}$, where P_i has m_i edges and Q_i has n_i edges. By definition of a realizing event, none of the events in E_r occur on the right or top boundary of cell $C(m-1, n-1)$. Hence, for any $i \in \{1, \ldots, k+1\}$, it holds that $m_i \leqslant m$, $n_i \leqslant n$, and $m_i < m$ or $n_i < n$. Since a path exists in the free-space diagram at ε_r through all events in E_r, the induction hypothesis implies that, for any $i \in \{1, \ldots, k+1\}$, a locally correct matching $\mu_i = (\sigma_i, \theta_i)$ exists for P_i and Q_i such that μ_i is linear in every cell and $d_{\mu_i}(t) \leqslant \varepsilon_\mathrm{r}$ for all t with $\sigma_i(t) \geqslant 1$ or $\theta_i(t) \geqslant 1$, and $\sigma_i(t) \leqslant m_i - 1$ or $\theta_i(t) \leqslant n_i - 1$. Combining these matchings with the events in E_r yields a matching $\mu = (\sigma, \theta)$ for (P, Q). As we argue below, this matching is locally correct and satisfies the additional properties.

The matching of an event corresponds to a single point (type B) or a horizontal or vertical line (type C) in the free-space diagram. By induction, μ_i is linear in every cell. Since all events occur on cell boundaries, the cells of the matchings and events are disjoint. Therefore, the matching μ is also linear inside every cell.

For $i < k + 1$, d_{μ_i} is at most ε_r at the point where μ_i enters cell (m_i, n_i) in the free-space diagram of P_i and Q_i. We also know that d_{μ_i} equals ε_r at the top right corner of cell (m_i, n_i). Since μ_i is linear inside the cell, $d_{\mu_i}(t) \leqslant \varepsilon_\mathrm{r}$ also holds for t with $\sigma_i(t) > m_i - 1$ and $\theta_i(t) > n_i - 1$. Analogously, for $i > 0$, $d_{\mu_i}(t)$ is at most ε_r for t with $\sigma_i(t) < 1$ and $\theta_i(t) < 1$. Hence, $d_\mu(t) \leqslant \varepsilon_\mathrm{r} \leqslant \varepsilon$ holds for t with $\sigma(t) \geqslant 1$ or $\theta(t) \geqslant 1$, and $\sigma(t) \leqslant m - 1$ or $\theta(t) \leqslant n - 1$.

To show that μ is locally correct, suppose for contradiction that values a, b exist such that $d_\mathrm{F}(P_\sigma[a, b], Q_\theta[a, b]) < d_\mu[a, b]$. If a, b are in between two consecutive events, we know that the submatching corresponds to one of the matchings μ_i. Since these are locally correct, $d_\mathrm{F}(P_\sigma[a, b], Q_\theta[a, b]) = d_\mu[a, b]$ must hold.

Hence, suppose that a and b are separated by at least one event of E_r. There are two possibilities: either $d_\mu[a, b] = \varepsilon_\mathrm{r}$ or $d_\mu[a, b] > \varepsilon_\mathrm{r}$. Since $d_\mu[a, b]$ includes a realizing event, $d_\mu[a, b] < \varepsilon_\mathrm{r}$ cannot hold. First, assume $d_\mu[a, b] = \varepsilon_\mathrm{r}$ holds. If $d_\mathrm{F}(P_\sigma[a, b], Q_\theta[a, b])$ is less than ε_r, then a matching exists that does not use the events between a and b and has a lower maximum. Hence, the free space admits a bimonotone path from point $(\sigma(a), \theta(a))$ to point $(\sigma(b), \theta(b))$ at a lower value than ε_r. This implies that all events between a and b can be omitted, contradicting that E_r is a minimal realizing set.

Now, assume $d_\mu[a, b] > \varepsilon_\mathrm{r}$. Let t' denote the highest t for which $\sigma(t) \leqslant 1$ and $\theta(t) \leqslant 1$ holds, that is, the point at which the matching leaves cell $C(0, 0)$. Similarly, let t'' denote the lowest t for which $\sigma(t) \geqslant m - 1$ and $\theta(t) \geqslant n - 1$ holds. Since $d_\mu(t) \leqslant \varepsilon_\mathrm{r}$ holds for any t with $t' \leqslant t \leqslant t''$, $d_\mu(t) > \varepsilon_\mathrm{r}$ can hold only for $t < t'$ or $t > t''$. Suppose that $d_\mu(a) > \varepsilon_\mathrm{r}$ holds. Then $a < t'$ holds and μ is linear between a and t'. Therefore, $d_\mu(a) > d_\mu(t)$ holds for any t with $a < t < t'$. Analogously, if $d_\mu(b) > \varepsilon_\mathrm{r}$ holds, then $d_\mu(b) > d_\mu(t)$ holds for any t with $t'' < t < b$. Hence, $d_\mu[a, b] = \max\{d_\mu(a), d_\mu(b)\}$ must hold. This maximum is a lower bound on the Fréchet distance, contradicting the assumption that $d_\mu[a, b]$ is larger than the Fréchet distance. Matching μ is therefore locally correct. □

5.3 Computation

The existence proof directly results in a recursive algorithm, which is given in Algorithm 5.1. Figure 5.5 and all other illustrated locally correct matchings in this chapter have been computed with this algorithm. In this section we prove the following theorem.

Theorem 5.7. *Algorithm 5.1 computes a locally correct Fréchet matching of two open polygonal curves P and Q with m and n edges in $O((m+n)mn \log mn)$ time.*

The crucial step here is to find the event e_r efficiently (Line 6). This is done in three steps. First, we compute the lowest value of ε for which the free-space diagram is feasible. Second, we compute a realizing set E. Finally, we find a significant event in E, that is, an event in some minimal realizing set. Below, we provide the details for each of these three steps.

Recall the following definitions and notations from Section 2.4.2. For $i \in \{0, \dots, m\}$ and $j \in \{0, \dots, n\}$, $L^\mathrm{F}_{i,j}$ denotes the *door* (i.e., the interval of free space) on $L_{i,j}$, the

Algorithm 5.1 ComputeLCFM(P, Q)

```
Require: $P$ and $Q$ are open polygonal curves with $m$ and $n$ edges
Ensure: A locally correct Fréchet matching for $P$ and $Q$
 1: if $m = 0$ or $n = 0$ then
 2:     return $(\sigma, \theta)$ where $\sigma(t) = t \cdot m$, $\theta(t) = t \cdot n$
 3: else if $m = n = 1$ then
 4:     return $(\sigma, \theta)$ where $\sigma(t) = \theta(t) = t$
 5: else
 6:     Find event $e_\mathrm{r}$ of a minimal realizing set
 7:     Split $P$ into $P_1$ and $P_2$ according to $e_\mathrm{r}$
 8:     Split $Q$ into $Q_1$ and $Q_2$ according to $e_\mathrm{r}$
 9:     $\mu_1 \to$ ComputeLCFM$(P_1, Q_1)$
10:     $\mu_2 \to$ ComputeLCFM$(P_2, Q_2)$
11:     return concatenation of $\mu_1$, $e_\mathrm{r}$, and $\mu_2$
```

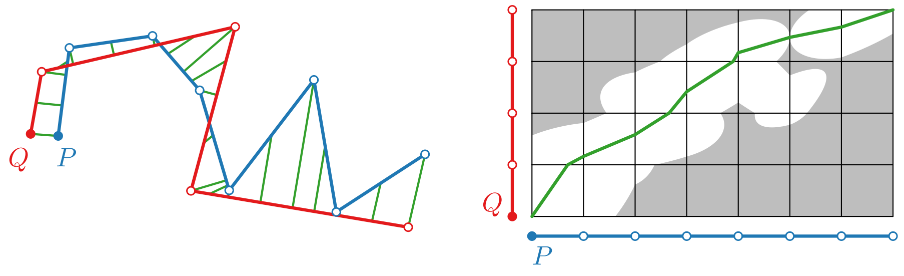


Figure 5.5 Locally correct matching produced by Algorithm 5.1. Free-space diagram for $\varepsilon = d_\mathrm{F}(P, Q)$.

left boundary of cell $C(i,j)$; $L^{\mathrm{R}}_{i,j}$ denotes the *reach-door*, that is, the intersection $L^{\mathrm{F}}_{i,j} \cap \mathrm{reach}(P,Q)$ between the door and the reachable space. Analogously, $B^{\mathrm{F}}_{i,j}$ and $B^{\mathrm{R}}_{i,j}$ are defined for the bottom boundary.

Computing the Fréchet distance. First, we compute the lowest value of ε for which the free-space diagram is feasible. That is, we compute the minimal value of ε such that the free space admits a bimonotone path from cell $C(0,0)$ to cell $C(m-1,n-1)$. This corresponds to computing a "modified" Fréchet distance, one in which we ignore the type-A events. With only minor modifications to the decision algorithm, we may apply the algorithm by Alt and Godau [10]. $L^{\mathrm{R}}_{0,0}$ and $B^{\mathrm{R}}_{0,0}$ are treated as nonempty intervals including $(0,0)$, the origin of the free-space diagram. Essentially, this ensures that any free space on the top and right side of cell $C(0,0)$ is reachable. $L^{\mathrm{F}}_{m,n}$ and $B^{\mathrm{F}}_{m,n}$ are treated as nonempty intervals including (m,n), the destination of the free-space diagram. This ensures that (m,n) is reachable if any point on the bottom or left side of cell $C(m-1,n-1)$ is reachable. These changes have no effect on the execution time of the algorithm. Thus, it runs in $O(mn \log mn)$ time.

Computing a realizing set. In the second step, we compute some (possibly nonminimal) realizing set E. Recall that this is a subset of $\mathcal{E}$ (all concurrent realizing events) such that the free space admits a bimonotone path from cell $C(0,0)$ to cell $C(m-1,n-1)$ without using events in $\mathcal{E}\backslash E$. We present a simple $O(mn)$-time algorithm to find a realizing set for a given value of ε. This algorithm is again a modified version of the decision algorithm by Alt and Godau [10]: it keeps track of any realizing events that are encountered. We observe that the occurrence of a *singleton reach-door*—a reach-door that consists of only a single point—corresponds directly to a realizing event. Although not all realizing events have to give rise to a singleton reach-door, a realizing set can be defined using only events that cause a singleton reach-door. This is formalized in the following lemma.

Lemma 5.8. *If $L^{R}_{i,j}$ or $B^{R}_{i,j}$ is a singleton, then a realizing event ends at $L_{i,j}$ or $B_{i,j}$ respectively. The set of all events that cause a singleton reach-door is a realizing set.*

Proof. The reach-door $L^{\mathrm{R}}_{i,j}$ is determined by the maximum value in the free space of the boundary, $L^{\mathrm{F}}_{i,j}$, and the minimum in some door $L^{\mathrm{F}}_{i',j}$ with $i' \leqslant i$. If $L^{\mathrm{R}}_{i,j}$ is a singleton, then this minimum and maximum coincide and thus this corresponds to an event of type B ($i' = i$) or type C ($i' < i$). The argument for a reach-door $B^{\mathrm{R}}_{i,j}$ is analogous. What remains is to argue that the corresponding events form a realizing set.

Let π be some bimonotone path in the free space between the first and last cell. Path π passes through a number of realizing events. We are done if all these events end at a boundary for which the reach-door is a singleton. So, assume that e is the last event along π that ends on $L_{i,j}$ for which $L^{\mathrm{R}}_{i,j}$ is not a singleton. Note that $L^{\mathrm{R}}_{i,j}$ cannot be empty as π passes through it. Event e must be of type C, as a type-B event implies that $L^{\mathrm{F}}_{i,j}$ is a singleton and thus $L^{\mathrm{R}}_{i,j}$ would be as well. Hence, we let $L_{i',j}$ denote the boundary where event e starts and $i' < i$. Since realizing event e defines a passage from $L_{i',j}$ to $L_{i,j}$, the startpoint of e corresponds to the lowest point in $L^{\mathrm{F}}_{i',j}$; the endpoint of e is the maximal point in $L^{\mathrm{F}}_{i,j}$ and thus of $L^{\mathrm{R}}_{i,j}$. Therefore, any other reachable point on $L_{i,j}$ implies that there is some boundary $B_{i^*,j}$ such that $i' \leqslant i^* < i$ and $B^{\mathrm{R}}_{i,j}$ is nonempty. Hence, there is a bimonotone path π' in the free space up through $B_{i^*,j}$ to $L_{i,j}$. In particular, π' does not pass through e. Hence, the concatenation of π' with the subpath of π starting at $L_{i,j}$ is a bimonotone path in the free space between the first and last cell. Any event along π' that does not end at a boundary with a singleton reach-door must end at some boundary $L_{x,y}$ or $B_{x,y}$ with $x < i$ and $y < j$. Therefore, repeating the replacement above must terminate as there are only a finite number of boundaries. This proves that there is some bimonotone path between the first and last cell that passes only through events that end on a boundary with a singleton reach-door. Thus, these events form a realizing set. □

From the above, we learn how to compute a realizing set: we simply record occurrences of singleton reach-doors. However, this gives us only the boundary on which the event ends; we also need to know the boundary on which the event starts. Assume a singleton reach-door is found on boundary $L_{i,j}$. Let $h < i$ be the largest value such that $B^{\mathrm{R}}_{h,j}$ is nonempty: all bottom boundaries between h and i are not reachable. Then the singleton is caused by the lower endpoint of $L^{\mathrm{F}}_{g^*,j}$ maximized over all g^* with $h < g^* \leqslant i$. This means that the event that causes the singleton starts at $L_{g^*,j}$ and ends at $L_{i,j}$. We refer to g^* as the *event index*. For each row we maintain this event index. After computing $L^{\mathrm{R}}_{i,j}$ but before checking for a singleton, the event index is updated to j if either $B^{\mathrm{R}}_{i-1,j}$ is nonempty or the lower endpoint of $L^{\mathrm{F}}_{g^*,j}$ is less than or equal to the lower endpoint of $L^{\mathrm{F}}_{i,j}$. If this is not the case, g^* maintains its value. Observe that we may replace the check for a singleton interval by computing the value of the critical event between $L_{i,j}$ and $L_{g^*,j}$ and comparing it to ε. This is slightly more numerically stable. Columns and horizontal boundaries are dealt with analogously. Maintaining the event indices incurs no asymptotic overhead on the basic decision algorithm. Hence, this modified algorithm that finds a realizing set also runs in $O(mn)$ time.

Finding a significant event. In this last step, we find a significant event e_{r} in the realizing set E, that is, e_{r} must be contained in a *minimal* realizing set. We assume that the events in E end at different cell boundaries. If events end at the same boundary, then these occur in the same row (or column) and it suffices to consider only the event that starts at the rightmost column (or highest row). The algorithm described above computes exactly those events.

To find a significant event in E, we proceed as follows. Fix the order of events in E. Let e_k denote the kth event and let $E_k = \{e_1, \ldots, e_k\}$ denote the first k events of E. We use a binary search on E to find the r such that E_r contains a realizing set, but E_{r-1} does not. This implies that event e_r is contained in a minimal realizing set. Note that r is unique due to monotonicity. What remains is to describe an algorithm that checks whether E_k is a realizing set.

To determine whether some E_k is a realizing set, we check whether the free-space diagram is feasible even without "using" the events of $\mathcal{E} \backslash E_k$. Again, we modify the decision algorithm by Alt and Godau [10]. We associate the index of each event in E with the boundaries at which the event ends in the free-space diagram. When $L^{\mathrm{R}}_{i,j}$ is computed, we check whether $L^{\mathrm{R}}_{i,j}$ is a singleton and an index k' is associated with $L_{i,j}$.

If this is not the case, then no realizing event ends at $L_{i,j}$ or it is not required to reach it. If the reach-door is a singleton and k' exists, event $e_{k'}$ is required to reach $L_{i,j}$ (as argued in the previous paragraph). We then check whether this event may be used by comparing k and k'. If $k' \leqslant k$, no action is taken; otherwise, the event may not be used and $L^{\mathrm{R}}_{i,j}$ is replaced with an empty interval. In this modified algorithm, cell $C(m-1, n-1)$ is reachable (that is, $L^{\mathrm{R}}_{m-1,n-1}$ or $B^{\mathrm{R}}_{m-1,n-1}$ is nonempty) if and only if E_k is a realizing set. The additional check takes only constant time per cell boundary. Thus, we decide in $O(mn)$ time whether E_k is a realizing set.

To obtain an algorithm that is numerically more stable, we again maintain the event index g^* for each row and column. Assume that $L_{i,j}$ has some associated event e that starts on $L_{g,j}$. Event e is necessary to obtain $L^{\mathrm{R}}_{i,j}$ if and only if $g = g^*$. Therefore, we can replace checking whether $L^{\mathrm{R}}_{i,j}$ is a singleton with an equality check of two integers.

Analysis. As detailed above, computing e_{r} (Algorithm 5.1, Line 6) consists of three steps. The first and second step—computing the Fréchet distance and a realizing set—take $O(mn \log mn)$ time combined. The third step performs a binary search on the realizing set and thus depends on its size. As argued, it contains at most $O(mn)$ events. However, concurrent events can be considered degenerate. To analyze this step more accurately, we therefore use K to denote the maximum number of elements in the realizing set. The third step then takes $O(mn \log K)$ time. Splitting the curves P and Q according to e_{r} and concatenating the matchings can be done in $O(m+n)$ time and is thus subsumed under the previous steps. Each recursion step splits the problem into two smaller problems, and the recursion ends when $mn \leqslant 1$. This results in an additional factor $O(m+n)$. Thus, the total execution time is $O((m+n)mn \log mn)$.

Substituting other methods. We used Alt and Godau's method [10] to compute the Fréchet distance. However, it can be substituted by any other algorithm, provided that the modification can be made to "ignore" the first and last cell. In general, this leads to an algorithm that runs in $O(n(T(m,n) + mn \log K))$ time, where $T(m,n)$ indicates the computation time of the (modified) algorithm used to compute the Fréchet distance. Whereas earlier the degeneracy was subsumed under the computation of the Fréchet distance, it is now a relevant factor if $T(m,n) = o(mn \log mn)$.

As an example, we may substitute the algorithm described in Chapter 4). Similar to Alt and Godau's algorithm, this algorithm can easily be adapted by treating four doors in the free-space diagram as open. For very degenerate curves—$K = \Omega(m+n)$—this substitution does not lead to an improved execution time. However, there is a pitfall here: the improved algorithm is faster only if $m = \Omega(n/\log n)$, assuming $m \leqslant n$. Therefore, we proceed as follows. If $m \geqslant \beta n/\log n$ or $n \geqslant \beta m/\log m$ for some constant $\beta > 0$, we use the algorithm of Chapter 4; otherwise, we simply use the modified version of Alt and Godau's algorithm.

Vertex sampling. Ideally, the computed Fréchet matching depends only on the shape of the given curves. However, locally correct Fréchet matchings are not unique and adding extra vertices may alter the result, even if these do not modify the shape. This is illustrated in Figure 5.6. In this figure, the columns of the free-space diagrams are stretched

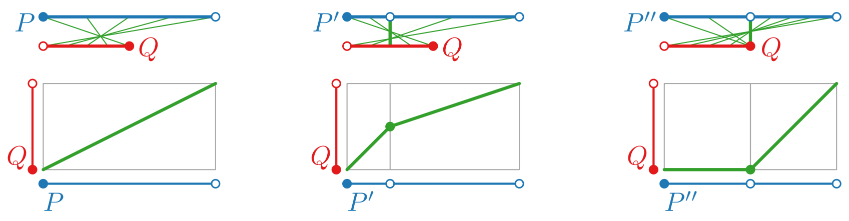


Figure 5.6 Different sampling may result in different matchings. In the free-space diagrams, columns are stretched to represent edge length.

to correspond to edge length. This makes the diagram invariant under sampling of P, thus emphasizing the difference in matching. Increasing the sampling further and further seems to result in a matching that decreases the matched distance as much as possible within a cell. However, since cells are rectangles, there is a slight preference for taking longer diagonal paths.

Further restrictions. Two curves may still have many locally correct Fréchet matchings: the algorithm computes just one of these. For most applications, we expect that it is desirable to restrict to locally correct matchings. However, it is often desirable to find the "best" Fréchet matching, though what defines "best" likely depends on the intended application. For example, the curves in Figure 5.7 admit two matchings, both of which may be considered better: this depends on what the curves represent and on the purpose of the matching. Corresponding to these two examples, we mention two possible criteria to further restrict locally correct Fréchet matchings.

The first criterion is the "length" of the matching, measured by its path length in the free-space diagram [32]. Note that slope constraints [32] or speed limits [120] indirectly restrict the length of a matching. As we showed in the introduction, a strict slope constraint may in some cases lead to unintuitive matchings (see Figure 5.2). Using the shortest locally correct Fréchet matching may provide another option for applications in which length or slope constraints are desirable. An alternative to the shortest locally correct Fréchet matching would be to consider the computation of a locally correct matching that adheres to length or slope constraints.

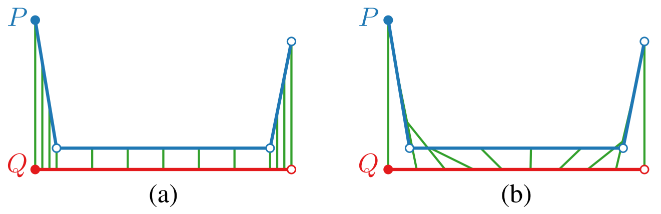


Figure 5.7 Two locally correct Fréchet matchings. (a) Matching that decreases distances as quickly as possible. (b) Shortest matching.

The second criterion again considers the matched distances. Local correctness enforces that local maxima in matched distances are avoided whenever possible, by considering subcurves that are induced by the matching. We may strengthen this idea by requiring that large distances are decreased as fast as possible, while maintaining local correctness. These "locally optimal" Fréchet matchings would require a steepest-descent method to find a matching. An important question is how to parameterize the descent. Using the L_2 norm for the descent results in nonlinear matchings with possible algebraic issues. Recently, Rote [153] has shown that using L_∞ yields a linear "locally optimal" matching (called lexicographic Fréchet matching in [153]). Assuming nondegenerate curves ($K = 1$), these are computable in $O(N^3 \log N)$ time where N is the combined complexity of the two curves [153].

5.4 Locally correct discrete Fréchet matchings

Here we study the discrete variant of Fréchet matchings. In this variant only the vertices of curves are matched. The discrete Fréchet distance can be computed in $O(mn)$ time via dynamic programming [75]. Here, we extend this simple algorithm to show that a locally correct discrete Fréchet matching can also be computed in $O(mn)$ time.

Grids. Since we are interested only in matching vertices of the curves, the free-space diagram turns into a grid. Suppose we have two curves P and Q with m and n edges respectively. These convert into a grid G of nonnegative values with $m + 1$ columns and $n + 1$ rows. Every column corresponds to a vertex of P, every row to a vertex of Q. Any node of the grid $G[i, j]$ corresponds to the pair of vertices (p_i, q_j). Its value is the distance between the vertices: $G[i, j] = \|p_i - q_j\|$. Analogous to free-space diagrams, we assume that $G[0, 0]$ is the bottomleft node and $G[m, n]$ the topright node.

Matchings. A bimonotone path π is a sequence of grid nodes $\pi(1), \dots, \pi(k)$ such that, for all $i \in \{2, \dots, k\}$, node $\pi(i)$ is the above, right, or above/right diagonal neighbor of $\pi(i - 1)$. In the remainder of this section a path refers to a bimonotone path unless indicated otherwise. A bimonotone discrete matching of the curves corresponds to a path π such that $\pi(1) = G[0, 0]$ and $\pi(k) = G[m, n]$. We call a path π locally correct if for all $1 \leqslant t_1 \leqslant t_2 \leqslant k$, $\max_{t_1 \leqslant t \leqslant t_2} \pi(t) = \min_{\pi'} \max_{1 \leqslant t \leqslant k'} \pi'(t)$, where π' ranges over all paths starting at $\pi'(1) = \pi(t_1)$ and ending at $\pi'(k') = \pi(t_2)$.

Algorithm. To compute a locally correct discrete Fréchet matching, the algorithm needs to compute a locally correct path from $G[0, 0]$ to $G[m, n]$ in a grid G of nonnegative values. To this end, the algorithm incrementally constructs a tree T on the grid. Tree T is rooted at $G[0, 0]$ and each path in T is locally correct. This is summarized in Algorithm 5.2. We define a *growth node* as a node of T that has a neighbor in the grid that is not yet part of T: a new branch may sprout from such a node. The growth nodes form a sequence of horizontally or vertically neighboring nodes. A *living node* is a node of T that is not a growth node but is an ancestor of a growth node. A *dead node* is a node of T that is neither a living nor a growth node, that is, it has no descendant that is a growth node. Every pair of nodes in this tree has a *nearest common ancestor* (NCA). When we add a

Algorithm 5.2 COMPUTEDISCRETELCFM(P, Q)

```
Require: $P$ and $Q$ are curves with $m$ and $n$ edges
Ensure: A locally correct discrete Fréchet matching for $P$ and $Q$
 1: Construct grid $G$ for $P$ and $Q$
 2: Let $T$ be a tree consisting only of the root $G[0, 0]$
 3: for $i \leftarrow 1$ to $m$ do
 4:     Add $G[i, 0]$ to $T$
 5: for $j \leftarrow 1$ to $n$ do
 6:     Add $G[0, j]$ to $T$
 7: for $i \leftarrow 1$ to $m$ do
 8:     for $j \leftarrow 1$ to $n$ do
 9:         ADDTOTREE$(T, G, i, j)$
10: return path in $T$ between $G[0, 0]$ and $G[m, n]$
```

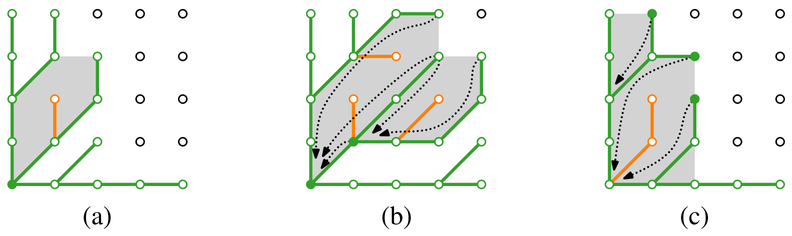


Figure 5.8 (a) Face (gray) of tree T with its unique sink (solid dot). An orange line represents a dead path. (b) Two adjacent faces with some shortcuts indicated. (c) A tree with three faces. Solid dots indicate growth nodes with a growth node as parent. These nodes are incident to at most one face and have their shortcut indicated.

new node to T, we have to decide on a growth node to be its parent such that paths to the new node are locally correct. To this end, we compare the maximum values encountered after the NCAs of the growth nodes. We provide more details on this procedure later in this section. A *face* of T is the area enclosed by the segment between two horizontally or vertically neighboring growth nodes (without one being the parent of another) and the paths to their NCA. The unique *sink* of a face is the node of the grid that is in the lowest column and row of all nodes on the face. Figure 5.8(a–b) shows some examples.

Shortcuts. To avoid repeatedly walking along the tree to compute maxima, we maintain up to two *shortcuts* from every node in the tree. The segment between the node and its parent is incident to up to two faces of the tree. The node maintains shortcuts to the sink of these faces, associating the maximum value encountered on the path between the node and the sink (excluding the value of the sink). Figure 5.8(b) illustrates some shortcuts. With these shortcuts, the maximum up to the NCA of two (potentially diagonally) neighboring growth nodes is computed in constant time.

Algorithm 5.3 ADDTOTREE(T, G, i, j)

Require: G is a grid of nonnegative values; any path in tree T is locally correct
Ensure: node $G[i, j]$ is added to T and any path in T is locally correct

```
1: $parent(G[i,j]) \leftarrow$ candidate parent with lowest maximum value to NCA
2: if $G[i-1, j-1]$ is dead then
3:     Remove the dead path ending at $G[i-1, j-1]$ and extend shortcuts
4: Make shortcuts for $G[i-1, j]$, $G[i, j-1]$, and $G[i, j]$ where necessary
```

Note that a node g of the tree that has a growth node as parent is incident to at most one face (see Figure 5.8(c)). We need the "other" shortcut only when the parent of g has a living parent. Therefore, the value of this shortcut can be obtained in constant time by using the shortcut of the parent. When the parent of g is no longer a growth node, then g obtains its own shortcut.

Extending the tree. Algorithm 5.3 summarizes the steps required to extend the tree T with a new node. Node $G[i, j]$ has three *candidate parents*, $G[i-1, j]$, $G[i-1, j-1]$, and $G[i, j-1]$. Each pair of these candidates has an NCA. For the actual parent of $G[i, j]$, we select the candidate c such that for any other candidate c', the maximum value from c to their NCA is at most the maximum value from c' to their NCA—both excluding the NCA itself. We must be consistent when breaking ties between candidate parents. To this end, we use the preference order of $G[i-1, j] \succ G[i-1, j-1] \succ G[i, j-1]$. Since paths in the tree cannot cross, this order is consistent between two paths at different stages of the algorithm. Note that a preference order that prefers $G[i-1, j-1]$ over both other candidates or vice versa results in an incorrect algorithm.

When a dead path is removed from the tree, adjacent faces merge and a sink may change. Hence, shortcuts have to be extended to point towards the new sink. By using a charging scheme in the analysis, we show that the execution time remains $O(mn)$. Figure 5.9 illustrates the incoming shortcuts at a sink and the effect of removing a dead path on the incoming shortcuts. Note that the algorithm does not need to remove dead paths that end in the highest row or rightmost column.

Finally, $G[i-1, j]$, $G[i, j-1]$, and $G[i, j]$ receive shortcuts where necessary. $G[i-1, j]$ or $G[i, j-1]$ needs a shortcut only if its parent is $G[i-1, j-1]$. $G[i, j]$ needs two shortcuts if $G[i-1, j-1]$ is its parent, only one shortcut otherwise.

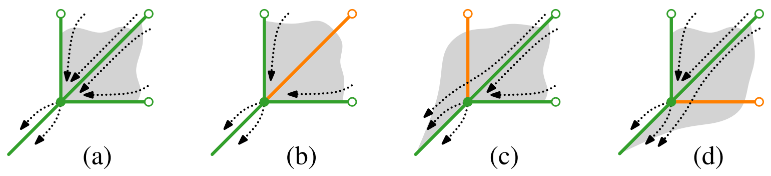


Figure 5.9 (a) Each sink has up to four sets of shortcuts. (b–d) Removing a dead path (orange) extends at most one set of shortcuts.

Correctness. To prove correctness of Algorithm 5.2, we require a stronger version of local correctness. A path π is *strongly locally correct* if for all paths π' with the same endpoints $\max_{1<t\leqslant k} \pi(t) \leqslant \max_{1<t'\leqslant k'} \pi'(t')$ holds. Note that the first node is excluded from the maximum. Since $\max_{1<t\leqslant k} \pi(t) \leqslant \max_{1<t'\leqslant k'} \pi'(t')$ and $\pi(1) = \pi'(1)$ imply $\max_{1\leqslant t\leqslant k} \pi(t) \leqslant \max_{1\leqslant t'\leqslant k'} \pi'(t')$, a strongly locally correct path is also locally correct. Lemma 5.9 below implies the correctness of Algorithm 5.2.

Lemma 5.9. *Algorithm 5.2 maintains the following invariant: any path in T is strongly locally correct.*

Proof. To prove this lemma, we strengthen the invariant.

Invariant. We are given a tree T such that every path in T is strongly locally correct. In constructing T, any ties were broken using the preference order.

Initialization. Tree T is initialized such that it contains two types of paths: either between grid nodes in the first column or in the first row. In both cases there is only one path in the grid between the endpoints of the path. Therefore, this path must be strongly locally correct. Since every node has only one candidate parent, T adheres to the preference order.

Maintenance. The algorithm extends T to T' by including node $g = G[i, j]$. This is done by connecting g to one of its candidate parents ($G[i-1, j]$, $G[i-1, j-1]$, or $G[i, j-1]$), the one that has the lowest maximum value along its path to the NCA. We must now prove that any path in T' is strongly locally correct. The invariant implies that only paths that end at g could falsify this statement. We prove this via contradiction.

Assuming T' is not strongly locally correct, there must be an invalidating path that ends at g. This path must use one of the candidate parents of g as its before-last node. We distinguish three cases on how this path is situated compared to T'. The last case, however, needs two subcases to deal with candidate parents that have the same maximum value on the path to their NCA. Hence, we have four cases; these cases are illustrated in Figure 5.10. First, we introduce some common notation.

The invalidating path must diverge from the path in T to g. For each case, we consider the path π_{i} starting at the node before the first node that is different and end at a candidate parent of g in the invalidating path. Note that π_{i} need not be disjoint of the paths in T'. Slightly abusing notation, we also use a path π' to denote its maximum value excluding the first node: $\max_{1<t\leqslant k'} \pi'(t)$. We use p to denote the parent of g in T', that is, the candidate parent with the lowest maximum value to the NCA.

Case (a). Path π_{i} ends at p. Path π is the path in T' between the first and last vertex of π_{i}. Since (π_{i}, g) is the invalidating path, we know that $\max\{\pi_{\mathrm{i}}, g\} < \max\{\pi, g\}$ holds. This implies that $\pi_{\mathrm{i}} < \pi$ holds. In particular, this means that π, a path in T, is not strongly locally correct: a contradiction.

Case (b). Path π_{i} does not end at p and $\pi_{\mathrm{i}}(1)$ is not a descendant of the NCA of p and the last node of π_{i}. Path π_1 is the path in T from $\pi_{\mathrm{i}}(1)$ to this NCA. Paths π_2 and π_3 are paths in T that start at this NCA and end at p and the last node of π_{i} respectively. Since the endpoint of π_2 was chosen as parent over the endpoint of π_3, we know that $\pi_2 \leqslant \pi_3$ holds. Furthermore, since (π_{i}, g) is the invalidating path, we know that $\max\{\pi_{\mathrm{i}}, g\} < \max\{\pi_1, \pi_2, g\}$ holds. These two inequalities imply $\max\{\pi_{\mathrm{i}}, g\} <$

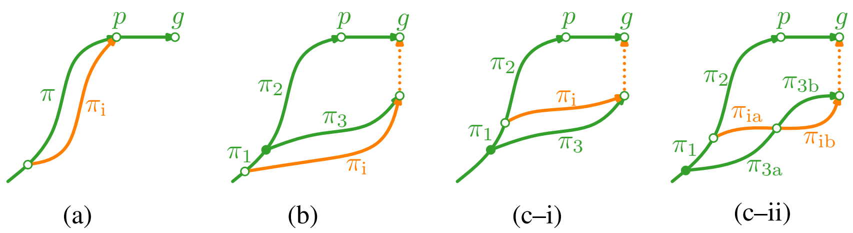


Figure 5.10 Four cases of the invalidating path (in orange) for the proof of Lemma 5.9.

$\max\{\pi_1, \pi_3, g\}$ holds. This in turn implies that $\pi_{\mathrm{i}} < \max\{\pi_1, \pi_3\}$ must hold. Since (π_1, π_3) is a path in T and the inequality implies that it is not strongly locally correct, we again have a contradiction.

Case (c). Path π_{i} does not end at p and the first node of π_{i} is a descendant of the NCA of p and the last node of π_{i}. Let π_1 be the path from this NCA to $\pi_{\mathrm{i}}(1)$. Path π_2 starts at $\pi_{\mathrm{i}}(1)$ and ends at p. Path π_3 starts at $\pi_1(1)$ and ends at the last node of π_{i}. In this case, we must explicitly consider the possibility of two paths having equal values. Hence, we distinguish two subcases.

Case (c–i). In the first subcase, we assume that the endpoint of π_2 was chosen as parent since its maximum value is strictly lower: $\max\{\pi_1, \pi_2\} < \pi_3$ holds. Since (π_{i}, g) is the invalidating path, we know that $\max\{\pi_{\mathrm{i}}, g\} < \max\{\pi_2, g\}$ holds. Since $\pi_2 \leqslant \max\{\pi_1, \pi_2\}$ always holds, we obtain that $\max\{\pi_{\mathrm{i}}, g\} < \max\{\pi_3, g\}$ must hold. This in turn implies that $\pi_{\mathrm{i}} < \pi_3$ holds. Similarly, since $\pi_1 \leqslant \max\{\pi_1, \pi_2\}$, we know that $\pi_1 < \pi_3$ must hold. Combining these last two inequalities yields $\max\{\pi_1, \pi_{\mathrm{i}}\} < \pi_3$. Since π_3 is a path in T and the inequality implies that it is not strongly locally correct, we again have a contradiction. (Note that with $\max\{\pi_1, \pi_2\} \leqslant \pi_3$, we can at best derive $\max\{\pi_1, \pi_{\mathrm{i}}\} \leqslant \pi_3$ which is not strong enough to contradict the invariant on T.)

Case (c–ii). In the second subcase, we assume that the endpoint of π_2 was chosen as parent based on the preference order: the maximum values are equal, thus $\max\{\pi_1, \pi_2\} = \pi_3$ holds. If π_{i} does not intersect π_3 before their common last node, we must conclude that $\pi_{\mathrm{i}} > \pi_3$: otherwise, the preference order would be violated at this common last node. In particular, this implies that π_{i} cannot be an invalidating path. If π_{i} does intersect π_3 before their common last node, we proceed as follows. We partition π_{i} into π_{ia} and π_{ib}: the split is based on the first node that π_{i} and π_3 have in common. At the same node, we also partion π_3 into $\pi_{3\mathrm{a}}$ and $\pi_{3\mathrm{b}}$. We now obtain two more cases, $\pi_{3\mathrm{a}} < \max\{\pi_1, \pi_{\mathrm{ia}}\}$ and $\pi_{3\mathrm{a}} \geqslant \max\{\pi_1, \pi_{\mathrm{ia}}\}$. In the former case, we obtain that $\max\{\pi_{3\mathrm{a}}, \pi_{\mathrm{ib}}\} < \max\{\pi_1, \pi_2\}$ holds and thus $(\pi_{3\mathrm{a}}, \pi_{\mathrm{ib}}, g)$ is also an invalidating path. Since this path starts at the NCA of π_2 and π_3, this is already covered by case (b). In the latter case, we have that either path $\pi_{3\mathrm{a}}$—which is in T—is not strongly locally correct (contradicting the invariant) or there is equality between the two paths $(\pi_1, \pi_{\mathrm{ia}})$ and $\pi_{3\mathrm{a}}$. In case of equality, we observe that $(\pi_1, \pi_{\mathrm{ia}})$ and $\pi_{3\mathrm{a}}$ arrive at their endpoint in the same order as π_2 and π_3 arrive at g. Thus T does not adhere to the preference order to break ties. This contradicts the invariant.

In all cases, we find that the assumption of an invalidating path contradicts the invariant. Therefore, we conclude that Algorithm 5.2 maintains the required invariant: all paths in T are strongly locally correct. □

Execution time. Most steps in the algorithm can be done easily in $O(1)$ time; the only exception is the removal of a dead path (Algorithm 5.3, Line 3). When a dead path π_{d} is removed, we may need to extend a list of incoming shortcuts at $\pi_{\mathrm{d}}(1)$, the node that remains in T. Let k denote the number of nodes in π_{d}. The first node of this path, $\pi_{\mathrm{d}}(1)$, remains a living node; thus there are $k-1$ dead nodes along the path. The lemma below relates the number of extended shortcuts to the size of π_{d}. The main observation is that the path requiring extensions starts at $\pi_{\mathrm{d}}(1)$ and ends at either $G[i-1, j]$ or $G[i, j-1]$, since $G[i, j]$ has not yet received any shortcuts.

Lemma 5.10. *A dead path π_{d} with k nodes results in at most $2k-1$ extensions.*

Proof. Since π_{d} is a path with k nodes, it spans at most k columns and k rows. When a dead path is removed, its endpoint is $G[i-1, j-1]$. Let π_{e} denote the path of T that requires extensions. Both paths start at the same node: $\pi_{\mathrm{d}}(1) = \pi_{\mathrm{e}}(1)$. The endpoint of π_{e} is either $G[i-1, j]$ or $G[i, j-1]$, since $G[i, j]$ has not yet received shortcuts when the dead path is removed. If the endpoint of π_{e} is not the parent of $G[i, j]$, then it has at most one child; it is a growth node and thus any of its descendants are also growth nodes. Hence, these descendants have a parent that is a growth node and thus do not have shortcuts that need to be extended. Figure 5.11 illustrates these situations. Hence, we know that π_{e} spans either $k+1$ columns and k rows or vice versa; the maximum number of nodes in π_{e} is $2k$, since it must be bimonotone. Since $\pi_{\mathrm{e}}(1)$ does not have a shortcut to itself, there are at most $2k-1$ incoming shortcuts from π_{e} at $\pi_{\mathrm{d}}(1)$. □

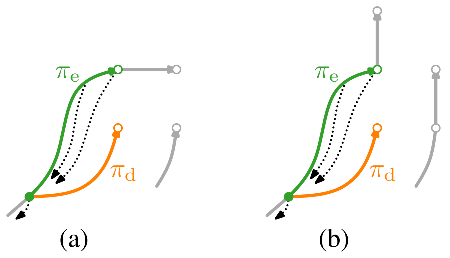


Figure 5.11 (a) Dead path π_{d} and path π_{e} that requires extensions. (b) Endpoint of π_{e} has one child. None of its descendants has a shortcut to $\pi_{\mathrm{e}}(1)$.

Hence, we can charge every extension to one of the $k-1$ dead nodes (all but $\pi_{\mathrm{d}}(1)$). Since these nodes are removed from T, a node gets at most 3 charges. Due to the existing shortcuts, each extension can be done in constant time. Therefore, the total execution time of the algorithm is $O(mn)$.

The dynamic program to compute the discrete Fréchet distance can be processed on a per-row or per-column basis, thus requiring only $O(\min\{m,n\})$ additional memory. This can also be done in our algorithm to compute a locally correct discrete Fréchet matching. However, we maintain tree T to store the locally correct paths. Naively speaking, this tree spans the entire grid, thus requiring $O(mn)$ space. By appropriately choosing per-row or per-column processing, T has at most $O(\min\{m,n\})$ growth nodes: any leaves that do not have an unprocessed neighbor are part of a dead branch and are thus removed from T. Any living node with exactly one child is never used, as it cannot be the sink of a face, nor is it ever the root of a dead branch. Therefore, such nodes can easily be removed without additional overhead: this yields a compressed tree T in which all living nodes (that are not growth nodes) have at least two children. Thus we conclude that the compressed tree contains at most $O(\min\{m,n\})$ nodes.

We summarize the findings of this section in the following theorem.

Theorem 5.11. *Algorithm 5.2 computes a locally correct discrete Fréchet matching of two polygonal curves P and Q with m and n edges in $O(mn)$ time and $O(\min\{m,n\})$ additional space.*

5.5 Conclusion

We set out to find "good" matchings between two curves. To this end, we introduced the local correctness criterion for Fréchet matchings. We have proven that at least one locally correct Fréchet matching exists for any two polygonal curves. This proof resulted in an $O((m+n)mn\log mn)$ recursive algorithm for two curves of complexity m and n. Furthermore, we considered computing a locally correct matching using the discrete Fréchet distance. By maintaining a tree with shortcuts to encode locally correct partial matchings, we have shown how to compute such a matching in $O(mn)$ time and $O(\min\{m,n\})$ space.

Future work. Computing a locally correct discrete Fréchet matching takes $O(mn)$ time, just like the basic dynamic program to compute only the discrete Fréchet distance. Recently, Agarwal *et al.* [3] have shown how to compute the discrete Fréchet distance in $O(mn\log\log n/\log n)$ time. This raises the question whether a similar improvement can be made to compute a locally correct discrete Fréchet matching.

The time required to compute a locally correct (continuous) Fréchet matching is a linear factor higher than the time to compute the Fréchet distance with Alt and Godau's algorithm [10]. An interesting question is whether this gap in computation can be reduced. Our algorithm for discrete matchings constructs a tree of locally correct paths in a single sweep of the parameter space. This suggests that going away from the "decision-and-search" paradigm for the continuous case may yield improvements here. Recently, Buchin *et al.* [38] have given such an algorithm to compute the Fréchet distance in $O(mn\log^2 mn)$ time. However, proceeding in the exact same way as for the discrete case is not sufficient: much of the information in the algorithm of Buchin *et al.* is maintained implicitly, to be constructed and retrieved only when it is actually needed. Moreover, along a single boundary of a cell, the locally correct Fréchet matchings may vary: the tree encoding the various matchings might become too big to maintain efficiently.

Finally, it would be interesting to further investigate criteria of restricting matchings. On the one hand, this includes further restrictions to locally correct Fréchet matchings, such as length and descent constraints (as discussed in Section 5.3). On the other hand, this also includes the benefit of local correctness for other matching-based similarity measures, such as the geodesic width [74].

Part II

Schematization Algorithms

Chapter 6

Schematization via Map Matching

In this chapter we investigate algorithmic properties of computing a $\mathcal{C}$-oriented schematization S with low complexity and low Fréchet distance for a given simple polygon P. As with simplification, we may distinguish two variants: the first asks to minimize the complexity of S constrained by a maximal Fréchet distance between S and ∂P, the boundary of P; the second asks to minimize the Fréchet distance constrained by a maximal complexity. In this chapter we attempt to solve this problem via map matching. *Map matching* asks to find a path (or cycle) in a given embedded graph that minimizes the Fréchet distance to a given open (or closed) polygonal curve. Because we are interested in computing a $\mathcal{C}$-oriented schematization, we assume that the graph is a plane graph that uses only edges oriented according to $\mathcal{C}$. We then solve the map-matching problem by computing a simple cycle in the graph that resembles ∂P. By construction, this cycle represents a $\mathcal{C}$-oriented schematization of the input polygon. Figure 6.1 illustrates this idea.

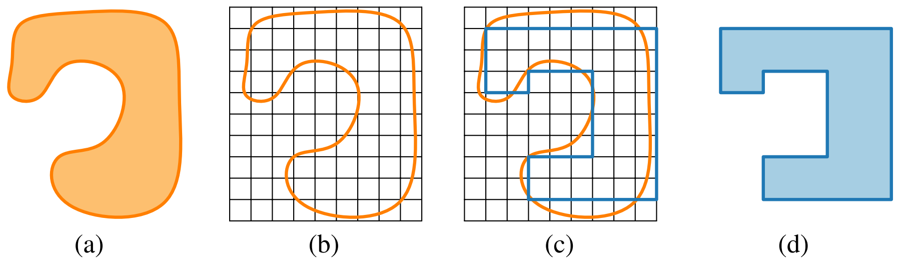


Figure 6.1 Schematization via map matching. (a) An input polygon. (b) Its boundary overlaid on a $\mathcal{C}$-oriented (rectilinear) graph G. (c) Compute a simple cycle in G that resembles the input. (d) This cycle is a $\mathcal{C}$-oriented schematization of the input.

While algorithms for map matching exist [9, 176], these methods cannot guarantee a simple path or cycle. One of the main applications of map matching is finding a driven route based on a road network and a sequence of GPS positions [28, 131, 176]. Driven routes may have selfintersections, for example, at bridges or cloverleafs. For schematization, however, this would result in a map that is not topologically valid. Hence, we seek a map-matching algorithm that considers only simple paths or cycles. We refer to this problem as *simple map matching*.

Unfortunately, this problem turns out to be NP-complete, as we prove in Section 6.1. In Section 6.2, we discuss the implications of this proof on approximability and fixed-parameter tractability and on using variants of the Fréchet distance. In Section 6.3, we explore options of solving map-matching instances. We provide an ILP formulation in an attempt to leverage efficient solvers. However, the resulting ILPs do not solve in reasonable time. The techniques used to develop the ILP formulation lead to a straightforward and comparatively efficient brute-force algorithm. We illustrate and discuss some results of this method.

Related work. A large number of papers are concerned with map-matching problems, e.g. [9, 28, 98, 101, 131, 176]; for an extensive overview, we refer to the review by Quddus *et al.* [141]. Map matching to find driven routes may include timestamps of the GPS measurements and may differentiate between road types. For schematization, we use the purely geometric formulation given above, without a time dimension or differentiation. We focus on establishing the computational complexity of variants under the Fréchet distance. Alt *et al.* [9] describe an algorithm that decides whether a path exists in $O(mn \log n)$ time, where m and n are the number of vertices of the polygonal curve and the plane graph respectively. Though "U-turns" can be avoided, no general simplicity guarantees are possible. Similarly, the decision problem for the weak Fréchet distance can be solved in $O(mn)$ time [28]. Wylie [180] proves that simple map matching using the discrete Fréchet distance is NP-hard. The same result also follows directly from our proofs. Studying a slightly different problem, Sherette and Wenk [160] show that it is NP-hard to determine the existence of a simple curve on a 2D surface with holes or in 3D. However, their proof does not extend to the case where the input curve is simple as well. Previous to the results presented in this chapter, the complexity of simple map matching under the Fréchet distance was unknown. For an overview of related work on schematization and simplification, we refer to Chapter 7.

6.1 Simple map matching is NP-complete

First, we consider the simple map-matching problem, without considering the number of bends (either as a constraint or as an optimization criterion). We prove that this problem is NP-complete, as formalized in the following theorem.

Theorem 6.1. *Let G be a plane graph and let P be a simple closed polygonal curve. It is NP-complete to decide whether G contains a simple cycle C with $d_F(P, C) \leqslant 1$.*

The problem is in NP since the Fréchet distance can be computed in polynomial time [10] and it is straightforward to check simplicity. In this section we prove that the problem is also NP-hard. We implicitly assume that the mentioned graphs are plane graphs.

Planar 3SAT. We give a reduction from planar 3SAT. For any planar 3SAT formula F, we show how to construct graph G and simple closed curve P such that G contains a simple cycle C with $d_{\mathrm{F}}(P, C) \leqslant 1$ if and only if F is satisfiable. Lichtenstein [117] showed that planar 3SAT is NP-hard. Knuth and Raghunathan [113] proved that it remains NP-hard, even if a specific rectilinear embedding is given (see Figure 6.2(a)). In this embedding all variables and clauses are represented as disjoint rectangles; the variables lie on a single horizontal line; and all links relating variables to clauses are strictly vertical. However, this is not exactly the embedding that we use to construct graph G and closed curve P. Our reduction requires that the clauses have a small fixed dimensions and hence cannot be stretched horizontally. We observe that a single bend for the two "outer" edges of a clause is sufficient to ensure this (see Figure 6.2(b)).

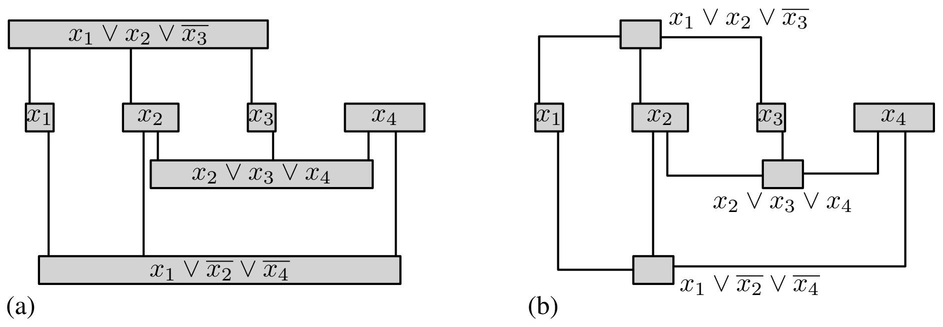

$x_1 \vee x_2 \vee \overline{x_3}$
x_1
x_2
x_3
x_4
$x_2 \vee x_3 \vee x_4$
$x_1 \vee \overline{x_2} \vee \overline{x_4}$
(a)
$x_1 \vee x_2 \vee \overline{x_3}$
x_1
x_2
x_3
x_4
$x_2 \vee x_3 \vee x_4$
$x_1 \vee \overline{x_2} \vee \overline{x_4}$
(b)

Figure 6.2 (a) Rectilinear embedding of a 3SAT formula [113]. (b) The same embedding with bends and fixed dimensions for clauses.

As is common in reductions from planar 3SAT, we define a number of *gadgets*. We define *variable gadgets*, *clause gadgets*, and *propagation gadgets* which represent the variables, clauses, and edges of F respectively. Based on these, we construct a simple map-matching instance—a graph and a simple closed curve—which represents F. That is, the constructed instance admits a simple cycle if and only if F is satisfiable. The result of this construction is illustrated in Figure 6.3. The reduction is split into two parts. In Section 6.1.1 we present the reduction based on a specification of the gadgets. In Section 6.1.2 we provide the details for each gadget.

6.1.1 Proof with gadget specifications

Each gadget specifies part of the graph and part of the simple closed curve. We call these parts the *local graph* and *local curve* of a gadget. The gadgets interact via vertices and edges shared by their local graphs. There is no interaction based on the local curve: it is

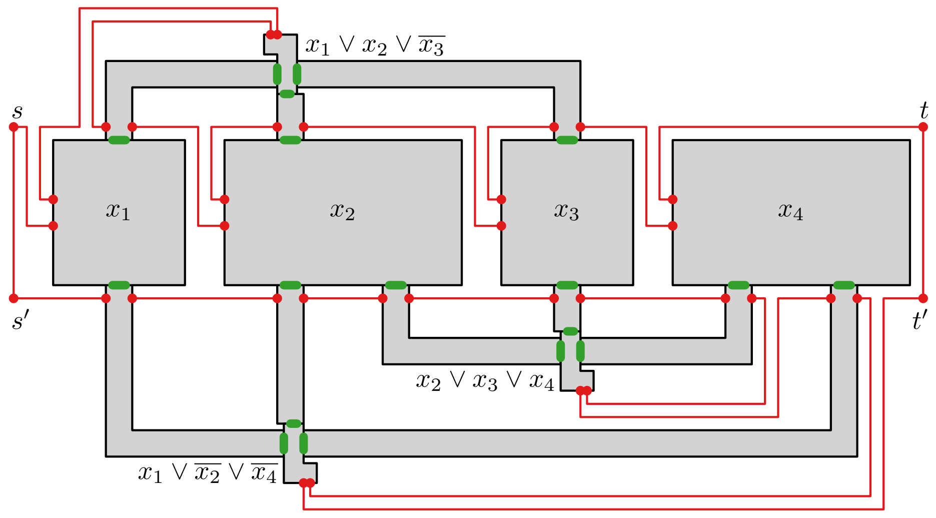

$x_1 \vee x_2 \vee \overline{x_3}$
s
t
x_1
x_2
x_3
x_4
s'
t'
$x_2 \vee x_3 \vee x_4$
$x_1 \vee \overline{x_2} \vee \overline{x_4}$

Figure 6.3 Overview of the construction for the formula in Figure 6.2. Each gray block represents a gadget. The red lines connect the various gadgets to obtain a simple closed curve. The green bars indicate places where gadgets interact.

used only to force choices in using edges of the local graph to find a simple cycle with a Fréchet distance of at most 1.

For now, we abstract from the details of the local graph and curve. We first give only specifications for the gadgets: we describe their desired behavior. Based on this specification, we complete the reduction. In Section 6.1.2, we give a construction for each gadget that adheres to its specification.

If a cycle exists in the complete graph, some simple path in the local graph must have a Fréchet distance of at most 1 to the local curve. Using a certain path in the local graph "claims" its vertices and edges: the shared vertices and edges can no longer be used by another gadget as this would result in a nonsimple cycle. This results in *pressure* on the other gadget to use a different path. A gadget has a number of pressure *ports*. These ports correspond to a sequence of edges (and their vertices) in the local graph that may be shared with another gadget. A port may *receive* pressure, indicating that the shared edges and vertices may not be used in the gadget. Similarly, it may *give* pressure, indicating that the shared edges and vertices may not be used by other gadgets. All interaction between gadgets goes via these ports.

The local curves must be joined carefully, ensuring that the complete curve is a simple curve. Each gadget has two curve *gates* that correspond to the endpoints of the local curve. Later, we show how to connect these gates such that the complete curve is indeed simple. The complete graph is the union of all local graphs and some additional edges used to connect the gates.

In the following paragraphs, we give the specifications for the three gadgets. Each specification consists of the following items:

- its behavior in terms of its ports;
- a rectilinear bounding polygon that contains the local graph and local curve;
- the placement of its two gates and its ports.

Regardless of the gadget, we require that all coordinates are an integer multiple of a half. We give specifications for clauses and edges that are placed above the variables in the embedded formula. Gadgets for clauses and edges below are defined analogously, by a rotation over 180 degrees.

For each gadget we also give a visual representation on an integer grid. The bounding polygon is given with a black outline; the ports are represented with thick green lines; the gates are represented with red dots.

Propagation gadget. Each edge of the embedded formula is represented by a propagation gadget. The propagation gadget is shaped like a “thick edge”. If the edge has a bend, then the gadget also has a bend. A visual specification of the gadget is given in Figure 6.4.

The propagation gadget has two ports representing the endpoints of the edge. The gadget does not admit a path if both ports receive pressure. If one port receives pressure, the other gives pressure: as the name suggests, the gadget propagates pressure.

The gadget has a bounding polygon that represents the edge with thickness 4. If there is no bend, the bounding polygon is a (vertical) rectangle of width 4 and height h. If there is a bend, the bounding polygon has two dimensions: width w and height h. The bounding polygon is the union of two rectangles: a vertical rectangle of width 4 and height h and a horizontal rectangle of width w and height 4. For a right bend, these rectangles coincide at the lefttop corner; for a left bend, they coincide at the righttop corner. We constrain propagation gadgets to have height h at least 7. If it contains a bend, the width w is at least 6. In other words, in our construction of the gadget (Section 6.1.2), we may assume that smaller dimensions are not required.

The gates are located at distance 2 above the bottom side of the vertical rectangle. The ports are at the endpoints of the thick edge, the sides with fixed width 4. That is, one of

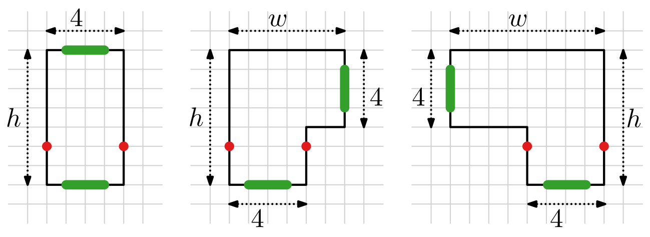


Figure 6.4 Specification of a propagation gadget with and without a bend. The bounding polygon is given in black; the red dots represent gates; the green segments indicate ports.

the ports is located on the bottom side of the vertical rectangle. If there is no bend, the other port is on the topside of the vertical rectangle. If there is a right or left bend, the other port is on the right or left side of the horizontal rectangle respectively. All ports have width 2 and are centered on their respective sides of the bounding polygon. On the sides that contain a port, the local graph has no edges or vertices except for those representing the port. Also, the local curve does not overlap these sides. This ensures that interaction between gadgets occurs only via ports and that the local curves do not intersect.

Clause gadget. The clause gadget has a fixed specification. A visual specification of this gadget is given in Figure 6.5.

A clause gadget has three ports. It admits a path if and only if it does *not* receive pressure on one of its ports. That is, any path—through the local graph between the gates with a Fréchet distance of at most 1 to the local curve—causes pressure on at least one of its ports; and for each port there is a path that causes pressure only on that port. The lack of external pressure on a port corresponds to the state of the corresponding variable being such that the clause is satisfied. If none of the variables has a state such that the literal is true (i.e., the clause is false), then all ports receive pressure and the clause gadget does not admit a path.

The bounding polygon has fixed dimensions. It is the union of a vertical rectangle with width 3 and height 9 and a horizontal rectangle of height 3 and width 5. These rectangles coincide on their righttop corner.

The gates are located at the top side of the horizontal rectangle and are at distance 1 and 3 from its left endpoint. The vertical rectangle contains the three ports: one of width 1, centered on its bottom side; one on its left and one on its right side, both have width 2 and are located at distance 2 above the bottom side.

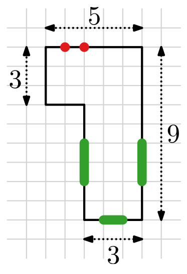


Figure 6.5 Specification of a clause gadget. It has fixed dimensions and requires three ports.

Variable gadget. The specification of a variable gadget depends on the number of occurrences in clauses: literals of the given variable or, alternatively, incident edges in the embedding of F. Let k^+ and k^- denote the number of occurrences in gadgets above or below the variables respectively. Let $k = \max\{k^+, k^-\}$ denote the maximum of these two values. We assume that $k > 0$; a variable with $k = 0$ does not occur in the formula and can be safely omitted. A visual specification of the gadget is given in Figure 6.6.

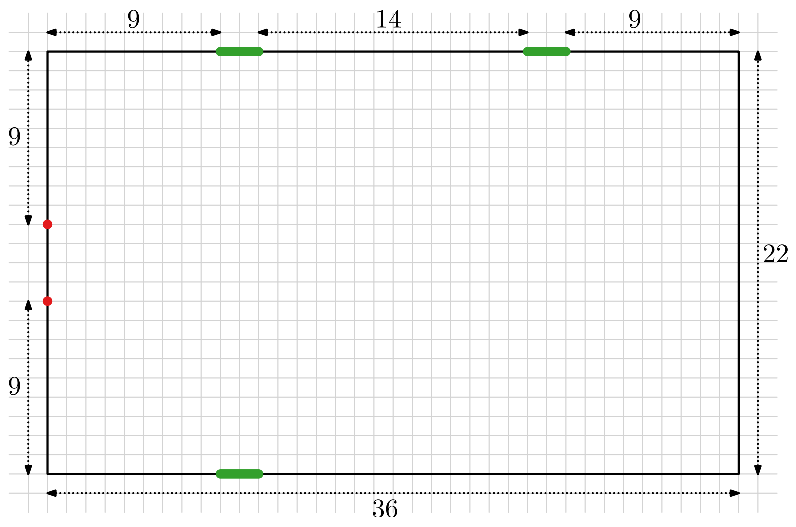


Figure 6.6 Specification of a variable gadget with $k^+ = 2$ and $k^- = 1$.

The variable gadget admits two states: one corresponds to *true*, the other to *false*. That is, each path in the local graph that has a Fréchet distance of at most 1 to the local curve must correspond to one of these states. The gadget has a number of ports along its top and bottom boundary, one for each occurrence of the variable. These ports give or receive pressure depending on the state. A port that corresponds to a positive literal gives pressure only in the false state. A port that corresponds to a negative literal gives pressure only in the true state. In other words, the port gives pressure if the state of the variable does not satisfy the corresponding clause.

The bounding polygon of the variable gadget is a rectangle that depends on the value of k. It has a width of $4 + 16k$. The height is fixed at 22.

The gates are both located on the left side, at a distance 9 and 13 from the top side. All ports have width 2. The i^{th} port (either above or below) is placed at distance $9+16(i-1)$ from the left side. That is, the first port is placed at distance 9, the second at 25, etc. Hence, the rightmost port ($i = k$) is at distance 9 from the right side.

Construction with gadgets. With the gadgets defined above, we now construct the complete graph G and simple closed curve P based on the structure given by the (modified) embedded formula F. Figure 6.3 illustrates this construction. Whenever we place a gadget, we argue that it does not overlap other gadgets that have already been placed.

First, we construct each variable gadget and place these on a horizontal line with a distance of 6 between consecutive variables. The order of the variables is given by the embedding of formula F. No variables overlap due to the horizontal spacing.

For the placement of clause gadgets, we introduce *dominance* between two clauses either both above or both below the variables. We present the placement for clauses above; placement for clauses below is analogous. Clause c *dominates* clause c' if there is a vertical line that intersects both c and c' (or the horizontal part of an incident edge) in the embedding of F and c is above c'. This induces a partial order on the clauses. Let $\mathrm{dom}(c)$ denote the clauses that are dominated by c. The *dominance number* of a clause is 0 if $\mathrm{dom}(c)$ is empty; otherwise, it is $1 + k$ where k is the maximum dominance number of a clause in $\mathrm{dom}(c)$. We compute the dominance number of all clauses in polynomial time by traversing the given embedding. Now, we construct and place the clause gadgets using their dominance number. Each clause has exactly one edge without a bend in F: the middle literal. Assume this middle literal is the i^{th} incident edge from above for the corresponding variable. We place the gadget such that its bottom port is above the i^{th} port of the variable. Moreover, we position it such that this port is at height $7 + 14k$ above the top of the variable gadget where k is the dominance number of the clause. Since a clause gadget has height 9, the vertical distance between gadgets with a different dominance number is at least 5. As $k \geqslant 0$, the gadgets cannot overlap a variable gadget. The distance between two consecutive ports is at least 14. Hence, no two clause gadgets overlap.

We now construct the propagation gadgets. For each middle edge (without a bend), we construct the propagation gadget without bend of the required length and place it between the clause and the variable. By placement of the clauses, the ports match up correctly: one port of the propagation gadget coincides with the bottom port of the clause gadget; the other coincides with the top port of the variable gadget. By the positioning of the clause gadgets, any propagation gadget has height at least 7 and thus adheres to the minimal height requirement. For edges with a bend, we construct propagation gadgets with a bend. Again, assume the edge is the i^{th} incident edge from above for the variable. We place it such that the bottom port coincides with the i^{th} port on the top side of the variable. We give it the proper height and width such that the other port of the propagation gadget coincides with the port of the corresponding clause. Due to the spacing of the ports, the vertical parts of propagation gadgets cannot overlap nor do these parts overlap clause or variable gadgets (except at the desired ports). Also, a propagation gadget does not overlap a horizontal part of another propagation gadget nor does it overlap a clause gadget: such overlaps would contradict the placement of clauses via the dominance number.

From the above, we know how to compose the various gadgets in polynomial time. We obtain a graph G, but we do not yet have a simple closed curve: each gadget has its own local curve. What remains is to argue that we can "stitch" these local curves together to create a single simple closed curve. All interaction between gadgets occurs via ports: the order of the gadgets along the curve is not used to convey information. We thus proceed as follows. First, we define two points s and t. These are at the same height as the gates of the propagation gadgets above the variables. Point s is 6 steps to the left of the leftmost variable gadget; t is 2 steps to the right of the rightmost. We create a simple curve from s to t that includes the local curves of the propagation and clause gadgets above the variables as well as the variable gadgets themselves. To this end, we move from s to t to construct the curve; at certain *events*, we include the various gadgets. We distinguish three events.

(a) At distance 4 before the left boundary of a variable gadget, we traverse the gadget as follows. First, we go straight down to the height of its lower gate and then four steps to the right to connect to the gate. After traversing the gadget, the curve exits at its top gate. We take two steps back to the left and go back up to the height of s.

(b) At distance 4 before the left boundary of a propagation gadget that has a right bend, we include the corresponding clause gadget. To this end, we go straight up to a height of 4 above the topside of the clause gadget. Then we go to the right and finally four steps down to connect to the right gate of the clause. After traversing the clause gadget, the curve exists on the left gate. We go up two steps and then go back to the start of the event, maintaining a distance of 2 to the curve used to arrive at the right gate of the clause gadget.

(c) When we reach the left gate of a propagation gadget, we traverse it and continue from its right gate.

We stop when point t is reached. Due to the spacing between the gadgets and ports, this traversal does not intersect any gadgets and the described events do not coincide nor do they occur within a propagation gadget. We have now constructed a simple curve from s to t that contains the local curves of the variable gadgets as well as the propagation and clause gadgets above the variables. We add to graph G the parts of the constructed curve that do not originate from a local curve of some gadget. For the propagation and clause gadgets below the variables, we use a similar procedure going back from point t' to s'. However, the variable gadgets are not included. By adding the vertical connection between s and s' as well as between t and t' to the curves and graph, we obtain the final graph G and simple closed curve P. An example of the traversal is given in Figure 6.3.

Proving the theorem. We now have a graph G and a simple closed curve P. Let n denote the number of variables, and m the number of clauses in formula F. For the reduction, the complexity of G and P must be polynomial in n and m. Moreover, the coordinates used to represent the vertices of G and P must be polynomial, such that each coordinate does not use more than $O(\log nm)$ bits.

The width of the entire construction is at most $8 + \sum_{i=1}^{n}(4 + 16k_i) + 6(n-1)$ where k_i is the number of occurrences of the i^{th} variable. As $\sum_{i=1}^{n} k_i = 3m$, we find that the width of the construction is bounded by $2 + 10n + 48m = O(n + m)$. Since the maximal dominance of a clause is $m - 1$, the height of the entire construction is bounded by $22 + 2(7 + 14m + 9 + 4) = 49 + 28m = O(m)$. Hence, the space occupied is $O(nm + m^2)$. The required coordinates easily fit into $O(\log nm)$ bits each, as all coordinates are required to be an integer multiple of a half. Clause and propagation gadgets have constant complexity; the complexity of a variable gadget is $O(k_i)$. The traversal to create a simple closed curve adds constant complexity per event, thus $O(n + m)$ in total. Hence, the constructed graph G and simple curve P have a polynomially bounded complexity. To conclude the proof of Theorem 6.1, we must argue that G has a simple cycle C with $d_{\mathrm{F}}(P, C) \leqslant 1$ if and only if F is satisfiable.

Assume that F is satisfiable and consider some satisfying assignment. We must now argue the existence of a simple cycle C. For each variable gadget, we choose a local path that corresponds to the true state of a variable that is assigned the value true; we choose a local path that corresponds to the false state otherwise. This gives pressure on a number of propagation gadgets: we choose the only remaining admissible path for these, causing pressure on the corresponding clauses. For the other propagation gadgets, we choose the path such that it gives pressure at the variable and may receive pressure at the clause. Since the truth values of the variables originate from a satisfying assignment, we know that at most two ports of any clause receive pressure. Hence, by its specification, the clause admits a local simple path as well. We concatenate the local paths with the paths that are used to stitch together the local curves of the various gadgets to obtain a simple cycle C. By construction, $d_\mathrm{F}(P, C)$ is at most 1.

Now, assume that G contains a simple cycle C with $d_\mathrm{F}(P, C) \leqslant 1$. By construction, cycle C traverses all gadgets and contains exactly one subpath for each gadget. This subpath ends at the gates of the gadget and the Fréchet distance between this subpath and the local curve is at most 1. For a variable, this local path corresponds to either the true or false state. This directly yields the truth values of the variables. Each clause gadget also has a local path and hence one or more of its ports give pressure. Since the propagation gadgets have a local path, the pressure from the clauses results in pressure on a variable gadget. This pressure ensures that a variable that receives pressure from a clause is in a state satisfying the clause. Hence, the truth values found from the variables yield a satisfying assignment for formula F.

This proves the theorem. However, we have worked only with the specification of the gadgets. In the next section, we implement the gadgets accordingly.

6.1.2 Gadget implementation

In the previous section we have proven Theorem 6.1 with the specifications of three gadgets. In this section we show how to implement these gadgets according to their specification by constructing a local graph and local curve.

We illustrate each of the constructions as follows. We give the local graph in thick light-blue lines and the local curve as a red line. Both are placed on top of the visual specification of a gadget, given in the previous section. We give various path choices using thinner dark-blue lines. A choice of path leads to pressure on the ports. Ports that give pressure are indicated with an outward arrow. As with the specifications, the constructions are visualized on an integer grid of thin gray lines. Recall that all coordinates must be an integer multiple of a half. Hence, we ensure that all vertices are placed on the integer grid, exactly halfway an edge of the grid, or exactly in the center of a cell.

Propagation gadget. To propagate pressure from one port to the other, we construct a local graph that admits only two paths with a Fréchet distance of at most 1 to the local curve. Each of the choices visits exactly one of the ports. The construction is illustrated in Figure 6.7 for gadgets without and with a right bend at minimal size: height $h = 7$ and, for gadgets with a bend, width $w = 6$. A gadget with a left bend is constructed symmetrically to one with a right bend. However, specification requires any larger height and width. This is achieved by stretching parts of the local graph and local curve (see Figure 6.7). This does not increase the number of edges in the local graph and curve nor does it change the Fréchet distance between an admissible path and the local curve.

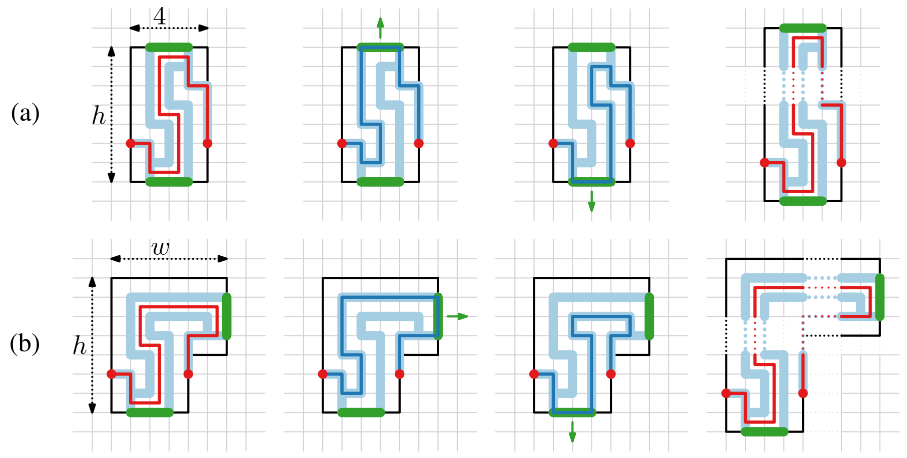


Figure 6.7 Construction of the local graph and local curve for a propagation gadget. Last column illustrates stretched parts (dotted) to obtain arbitrary dimensions. (a) Without a bend. (b) With a right bend.

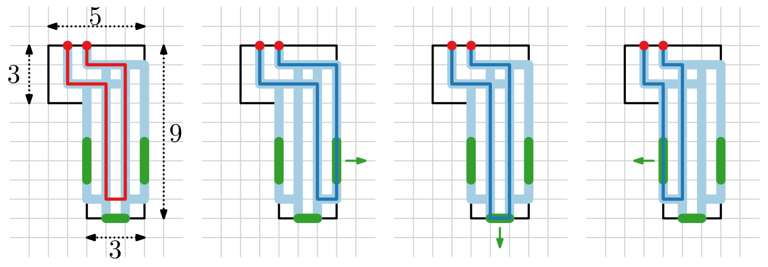


Figure 6.8 A clause gadget with its path choices. Each choice gives pressure on at least one port.

Clause gadget. The clause gadget is illustrated in Figure 6.8. It is constructed such that it admits three paths, each of which passes through a different port. Most importantly, it does not admit a path that passes through none of the ports. Therefore, if all ports receive pressure, no admissible path remains. However, it also admits paths that pass through multiple ports. In particular, it admits a path that puts pressure on the left and bottom port, on the bottom and right port, and on all three ports. Though not required, these paths also adhere to the specification.

Variable gadget. A variable gadget requires two states: true and false. The pressure on its ports must be consistent with its state and the literal in the corresponding clause. If the literal is positive, then the port should give pressure if the variable is false. If the literal is negative, then the port should give pressure if the variable is true. In short, a port should give pressure if the variable state does not satisfy the clause. To obtain a consistent

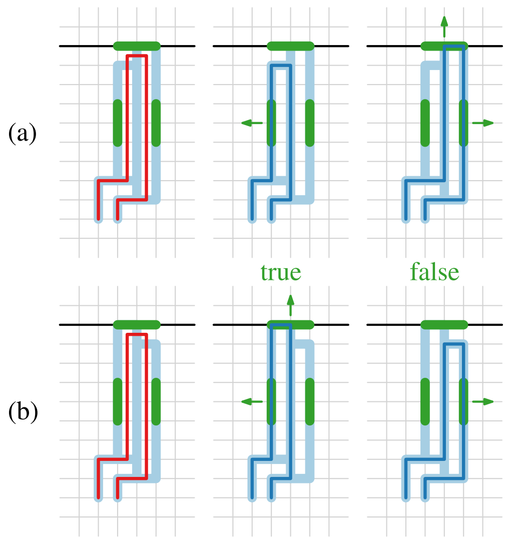


Figure 6.9 A literal block is shaped depending on whether the literal is positive (a) or negative (b). Regardless, it has two choices: counterclockwise (“true”) or clockwise pressure (“false”).

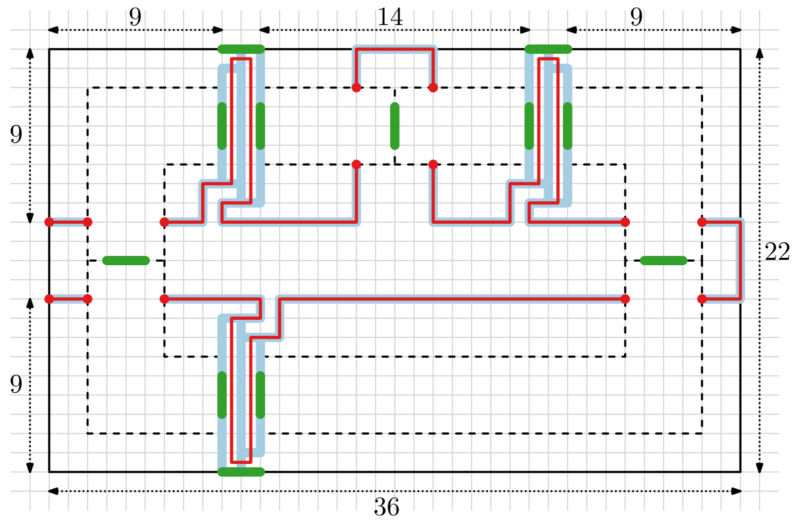


Figure 6.10 Literal blocks are connected via propagation gadgets (dashed) within the variable gadget. This creates the circular pressure that represents the truth value.

state, we insert a circular pressure within the variable gadget. We associate true with a counterclockwise pressure and false with a clockwise pressure.

At each port, a *literal block* translates the circular pressure into pressure on the port if necessary. This block is illustrated in Figure 6.9. Its shape depends on whether the literal is positive or negative. The only difference is which of the vertical "bars" is raised to touch the port at the top. It also features two "internal ports" that are used to create the circular pressure that represents the value of the variable. Observe that a literal block in isolation also admits a path that puts pressure on both internal ports. However, due to the circular pressure within the variable gadget, this path is never used.

We must connect the literal blocks via the internal ports in a circular fashion; see the sketch in Figure 6.10. To create the circular pressure, we reuse the construction of propagation gadgets. The local curves of the literal blocks end on the inside of the circular connection. Hence, we use two instances of a propagation gadget between two consecutive literal blocks. A complete variable gadget is illustrated in Figure 6.11.

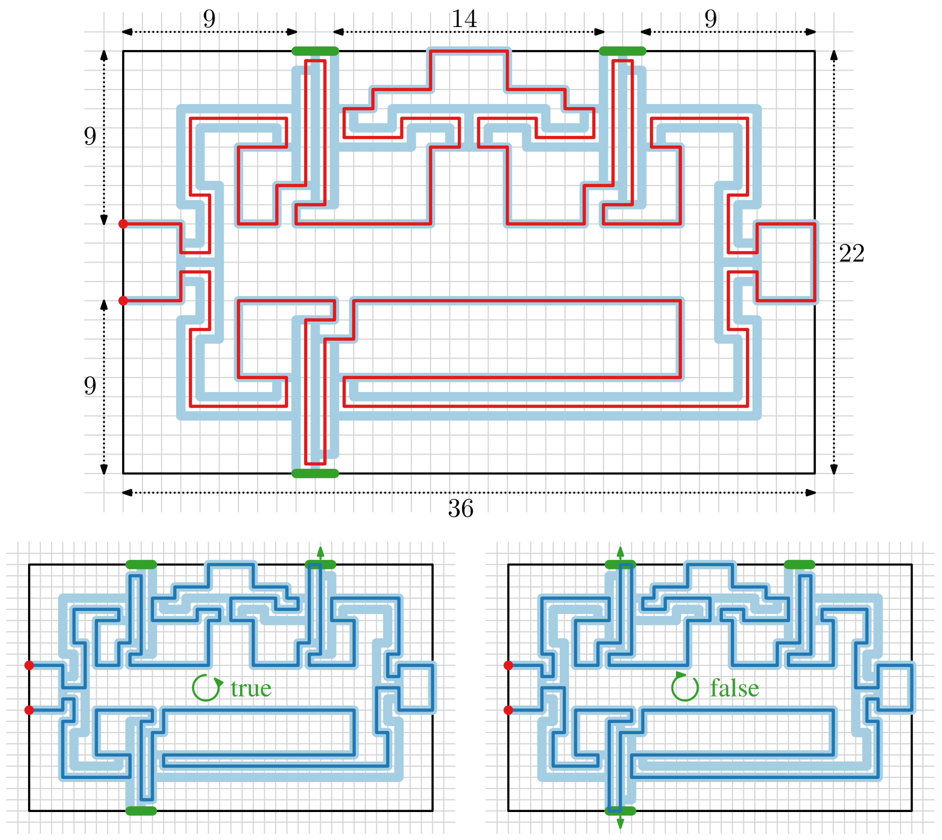


Figure 6.11 A complete variable gadget and its two states: true and false.

6.2 Implications

Theorem 6.1 and its proof have a number of implications on related problems. We briefly discuss them here.

Inapproximability. For an unsatisfiable planar 3SAT formula, the minimal Fréchet distance of a simple cycle in the constructed graph is significantly larger than 1. Suppose that this minimal Fréchet distance is strictly greater than c. Any c-approximation algorithm for simple map matching is able to decide satisfiability of planar 3SAT formulas. Thus, unless P=NP, no c-approximation algorithm can have polynomial execution time.

To investigate the value of c, we consider when constructions no longer adhere to their specification. That is, we wish to know the smallest c such that a gadget or construction admits a path with Fréchet distance c that does not adhere to its specification. A propagation gadget admits only two simple paths between its gates: they remain functional for higher values of c. For $c \geqslant 6$, a clause gadget admits a path that does not pass through any ports, violating its specification: the clause is considered "satisfied" though none of its literals are true (see Figure 6.12(a)). For $c \geqslant \sqrt{0.5^2 + 6.5^2} \approx 6.51$, a literal block within a variable gadget admits a path that does not pass through either of the internal ports: this breaks the circular pressure that represents the truth value of the variable (see Figure 6.12(b)). One more problem may occur: two constructions that occur consecutively along the simple curve may together admit a "shortcut" that violates the specification of either or both constructions. The parts connecting the various gadgets (red in Figure 6.3) always connect gates that have a distance larger than 6. Hence, these do not cause a problem. Only the connections within a variable gadget remain: between the literal blocks and internal propagation gadgets (see Figure 6.13). Here, problems indeed arise at the connections for $c < 6$. The smallest c that admits a path that does not adhere to the specification is $\sqrt{20} \approx 4.47$. Thus the construction shows that it is NP-hard to approximate the simple map-matching problem within a factor of 4.47.

However, it is straightforward to modify the construction to increase the value of c, while ensuring that only a satisfiable formula yields a simple cycle with a Fréchet distance of at most 1. To this end, clause gadgets and literal blocks can be elongated. Within

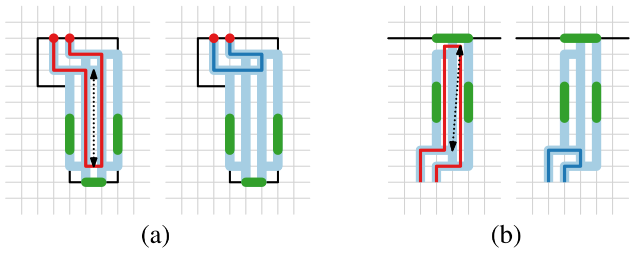

(a) (b)

Figure 6.12 Clause gadgets (a) and literal blocks (b) admit a path that does not adhere to the specification for a Fréchet distance significantly larger than 1.

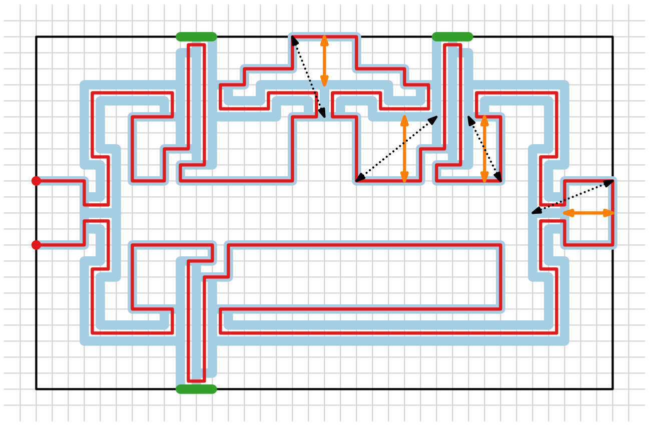


Figure 6.13 Several issues may arise in a variable gadget (dotted) for a Fréchet distance larger than 1. By increasing the distances indicated in orange, we increase the lowest value at which these problems occur.

a variable gadget, we increase the distance between the connector and the constructions to c (as indicated by the orange arrows in Figure 6.13). We must also increase the distance between consecutive ports of the same variable gadget to be at least c. Clause gadgets increase their height by $c - 6$; variable gadgets increase their height by at most $4c$ and their width by at most ck where k is the number of occurrences. The placement of the various gadgets is adjusted accordingly. Now, the upper bound on the total width is $O(c(nm + m^2))$; the total height is bounded by $O(cm)$ where n and m denote the number of variables and clauses respectively. Thus, for any value of c that is polynomially bounded in m and n, the size of the constructed instance is polynomial. We obtain the following result.

Corollary 6.2. *Let G be a plane graph and let P be a simple closed polygonal curve. Unless* P=NP*, no polynomial-time algorithm exists to approximate the minimal Fréchet distance $d_F(P, C)$ of any simple cycle C in G within any factor polynomially bounded in $|G|$ and $|P|$.*

Counting turns. With the inapproximability result above, we may wish to investigate the possibility of a fixed-parameter-tractable algorithm (FPT). That is, we would like to find an algorithm for which the execution time depends exponentially only on some parameter of small value (rather than the input size). A schematization typically has a low number of bends; we could therefore use the number of bends as a parameter. Suppose we desire a rectilinear schematization, that is, graph G contains only vertical and horizontal edges. Then any bend is either a left turn or a right turn. Every rectilinear polygon has a certain *turn profile*: the sequence of left and right turns in counterclockwise order along its boundary. The turn profile gives no information about edge lengths. As a result, many seemingly different polygons have the same turn profile (see Figure 6.14).

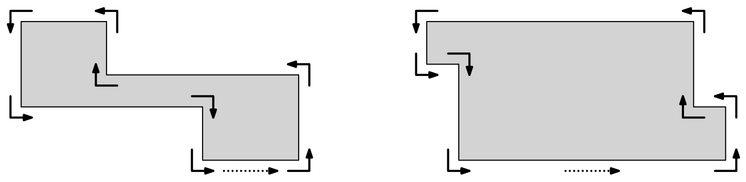


Figure 6.14 Two polygons with the same turn profile. Starting from dashed arrow, it is: L(eft), L, R(ight), L, L, L, R, L.

Consider the following approach. We choose a turn profile for the schematization and solve the simple map-matching problem for cycles that adhere to this turn profile. By optimizing over all turn profiles, we find the general solution to the simple map-matching problem. Unfortunately, even with a given turn profile, the problem remains NP-complete. Let us consider the various gadgets in the proof of Section 6.1. In all constructions, the turns made by the various admissible paths in the local graph are identical. Thus, we can easily decide the turn profile that a cycle in the complete graph must adhere to.

Corollary 6.3. *Let G be a plane graph, let P be a simple polygon and let T be a turn profile. It is NP-complete to decide whether G contains a simple cycle C that adheres to T with $d_F(\partial P, C) \leqslant 1$.*

This does not decisively prove that no FPT algorithm exists. However, it does show that one of the more obvious ways of obtaining an FPT is not feasible.

Variants. Finally, we observe that there are a number of variants of the problem that can be proven to be NP-complete via the same construction.

The strict monotonicity of the Fréchet distance is not crucial for the construction. Hence, the problem under the weak Fréchet distance is also NP-complete. Using the weak Fréchet distance does allow for more admissible paths in the local graph of a clause gadget. It now admits a path that results in pressure on both the left and right port (but not the bottom port). This path adheres to the specification and thus does not influence the proof.

We may sample the graph and curve on the integer grid. As the construction has polynomially bounded width and height, the resulting sampled graph and curve are also polynomially bounded in size. Hence, the problem is also NP-complete when applying the discrete (weak) Fréchet distance.

All interaction between and within gadgets is based on edges. Therefore, it is also NP-complete to determine the existence of an “edge-simple” cycle that uses each edge at most once (but vertices may be used more than once).

The proof straightforwardly extends to simple open curves and paths in the graph. Omitting the connection between s and s' (see Figure 6.3) yields a simple open curve, rather than a closed curve. Also, generalizations of the problem in which the graph is not a plane graph or the curve is not simple are NP-complete.

6.3 Solving simple map-matching problems

In this section we investigate schematization via map matching by considering the bend-minimization variant. That is, we study the following problem: given a plane graph G, a simple closed polygonal curve P and a value $\varepsilon > 0$, compute a simple cycle in G with a Fréchet distance of at most ε to P with a minimal number of bends (if it exists). As we have proven in the previous sections, it is unlikely that an efficient (approximation) algorithm exists. We now consider some options of computing exact solutions for comparatively small instances.

To facilitate this, we first transform the problem into a conceptually simpler problem in Section 6.3.1. This transformation avoids the need for computing or verifying Fréchet distances afterwards. This leads to a simple brute-force algorithm as described in Section 6.3.2. In Section 6.3.3 we show that this transformed problem can be solved via an *integer linear program* (ILP). Though this is also an NP-hard problem, there are methods to solve ILPs that are efficient in practice. By formulating a simple map-matching problem as an ILP, we may be able to leverage these methods. We present some experimental results of the brute-force and ILP methods in Section 6.3.4.

6.3.1 Interval graph

For our transformation, we wish to compute a structure that allows us to rid ourselves of further Fréchet distance computations. To this end, we compute for each edge in G the subcurves (“intervals”) of P that have a Fréchet distance at most ε to the edge . The structure describes exactly which subcurve of an edge (u, v) can precede a subcurve of another edge (v, w). Thus, by traversing this structure, we may construct cycles that have a Fréchet distance of at most ε to P. We refer to this structure as the *interval graph*.

We assume that the plane graph G is given as a (discrete) set of vertices $V \subset \mathbb{R}^2$ and *directed* edges $E \subset V \times V$. As the given graph is typically undirected, we represent an undirected edge between u and v as two directed edges, (u, v) and (v, u). We treat the simple closed curve P as a function $P\colon S^1 \to \mathbb{R}^2$ where S^1 denotes the unit circle. We assume that the diameter of P is strictly greater than 2ε, i.e., no single point has a Fréchet distance of at most ε to P. Such cases can be considered degenerate: the value of ε is too large to find a meaningful schematization in G. Moreover, we assume that all vertices of G lie within distance ε of some point on P. Vertices that lie further away cannot be used in a cycle; these can easily be discarded beforehand.

Sectors and sector points. To construct the interval graph, we first determine all points s on S^1 such that the distance between $P(s)$ and some vertex $v \in V$ is exactly ε. We refer to these points as *sector points* and we denote them, in sequence along S^1, by $\mathcal{S} = \langle s_1, \ldots s_k \rangle$. These points are correspond to the intersections of P with the circles of radius ε centered at the vertices of G. For vertex v, we denote by $\mathcal{S}_v$ the subsequence of $\mathcal{S}$ corresponding to intersections between P and the circle centered at v. As a circle and a line segment intersect at most twice, we know that $\mathcal{S}_v$ contains at most $2|P|$ sector points; the size of $\mathcal{S}$ is at most $2|P||V|$. The sector points partition S^1 into *sectors*.

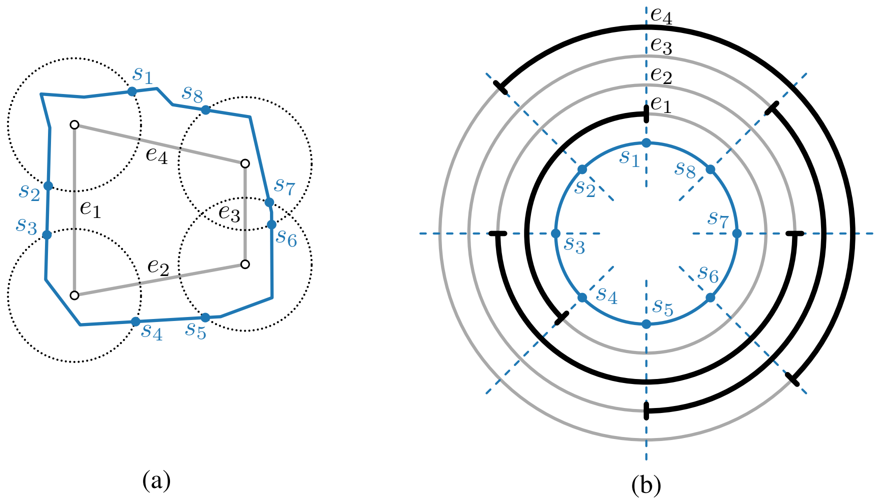


Figure 6.15 (a) The intersections of P with circles centered at vertices of G define a sequence of sector points, $\mathcal{S} = \langle s_1, \ldots, s_8 \rangle$. (b) The domain of P is the unit circle. Each edge has a set of intervals with Fréchet distance exactly ε to a subcurve of P.

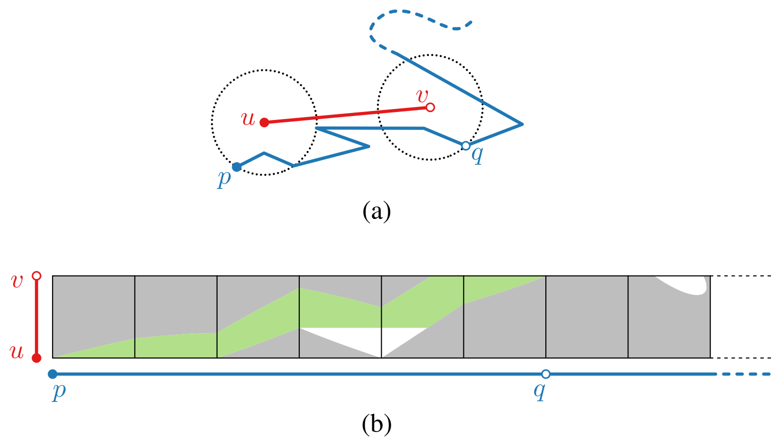


Figure 6.16 (a) An edge (u, v) and two sector points, p and q, that define a maximal subcurve (and hence an interval) with a Fréchet distance of exactly ε. (b) The free-space strip can be traversed to find the interval. The green area represents the reachable space, starting at (p, u). Its rightmost point determines the interval.

Intervals. In the second step we compute, for each edge $e \in E$, the subcurves of P with a Fréchet distance of at most ε to e. An edge may have an infinite number of such subcurves. Therefore, we restrict ourselves to maximal subcurves. We refer to a (contiguous) subdomain of S^1 that corresponds to a maximal subcurve as an *interval*. An interval is not contained in another, though intervals may overlap. An example to illustrate sectors, sector points and intervals is given in Figure 6.15.

Any interval for an edge $(u, v) \in E$ must start at a sector point in $\mathcal{S}_u$ and end at a sector point in $\mathcal{S}_v$. If an interval does not start at one of these points, it does not correspond to a maximal subcurve. Hence, an interval spans a number of (complete) sectors and represents a subcurve that has a Fréchet distance of exactly ε. We can compute the intervals for each edge as follows. We treat the vertices in $\mathcal{S}_u$ in order while maintaining a linked list L of current intervals. At each of these sector points s, we compute a candidate interval that starts at s with a Fréchet distance of exactly ε to (u, v). We consider the *free-space strip* of e and P starting at s: this is a free-space diagram for e and P, where P is "cut" at $P(s)$ into an open polygonal curve. This is illustrated in Figure 6.16. Using the dynamic programming techniques of Alt and Godau [10], we can traverse the free-space strip to compute the candidate interval in $O(n)$ time. This procedure gives us the sector point in $\mathcal{S}_v$ at which the candidate interval ends (if any exists). The candidate interval I may contain or be contained in intervals of L. If I is contained in some interval of L, it must be contained in the last, since we process the vertices in $\mathcal{S}_u$ in order. Thus, we can check in constant time whether this is the case. If it is contained, we discard I and continue with the next sector point. If it is not contained, we append it to L and remove any intervals in L that are contained in I. If any interval is contained in I, in particular the first must also be contained in I. We remove the first element of L until it is no longer contained in I. Hence, processing one vertex takes $O(|P|)$ time. After iterating over all points in $\mathcal{S}_u$, we found all intervals of (u, v). This takes $O(|P|^2)$ time per edge and thus $O(|P|^2|V|)$ time in total.

Transitions and transition points. As a last step in building the interval graph, we determine which interval of edge (v, w) can follow which interval of edge (u, v) in a cycle in G with a Fréchet distance of at most ε to P. We refer to these as *transitions* between intervals. To model transitions, we introduce *transition points* for all intervals. For each sector point that is contained in an interval (including its endpoints), we introduce a transition point. Consider a potential transition between two transition points, t and t', that correspond to the same sector point s. Moreover, assume that t is part of an interval of an edge (u, v) and that t' is part of an interval of an edge (v, w) such that $u \neq w$. A transition from t to t' is *admissible* if and only if $\|P(s) - v\| \leqslant \varepsilon$. If a transition is admissible, two intervals can be glued together at sector point s: the result has a Fréchet distance of at most ε to the sequence of two edges (see Figure 6.17). This follows from the fact that each interval has some matching between the edge and the subcurve of P and that cells are convex [10]: the two matchings can be combined at s into a new matching for the combined edges. If t is the last transition point or t' is the first transition point of its interval, then $\|P(s) - v\| = \varepsilon$ must hold: the transition is always admissible. In our interval graph, we store all transition points and admissible transitions.

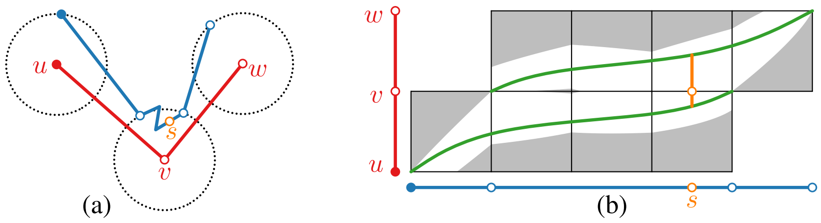


Figure 6.17 (a) Two edges (u, v) and (v, w) and part of P. (b) Their corresponding free-space strips. At any point on P with distance at most ε to v, the two matchings (green) can be combined (orange) to create a matching for the two edges combined in sequence.

Correctness. To prove correctness of this transformation, we argue two implications: any cycle of admissible transitions in the interval graph corresponds to a cycle in graph G with a Fréchet distance of at most ε to P and vice versa.

Suppose we have a cycle of admissible transitions $\langle t_0, \ldots, t_{k-1}, t_k = t_0\rangle$, such that the transition that leaves an interval is not before the transition that arrives at an interval. Let Q be the polygonal closed curve that represents the corresponding cycle in G. We must show that $d_{\mathrm{F}}(P, Q) \leqslant \varepsilon$. Each transition t_i matches a point p_i along P to a vertex q_i of Q (and thus of G); by definition, $\|p_i - q_i\| \leqslant \varepsilon$ holds. Because the transitions make a cycle in the interval graph, the interval I_i at which t_i ends corresponds to the interval at which t_{i+1} starts. Therefore, $e_i = (q_i, q_{i+1})$ is an edge in G for all $i \in \{0, \ldots, k-1\}$. Interval I_i corresponds to this edge and a maximal subcurve C of P with a Fréchet distance of exactly ε. Hence, a Fréchet matching must exist between the edge and C. Because t_i comes before t_{i+1} on I_i, we know that p_i comes before p_{i+1} along C. Because $\|p_i - q_i\| \leqslant \varepsilon$ holds for all i, we know that we can use this matching to define a Fréchet matching between e_i and the subcurve from p_i to p_{i+1} (see Figure 6.18(a)). Each edge e_i therefore admits a Fréchet matching that starts where e_{i-1} ends. Thus, all these Fréchet matchings can be concatenated into one Fréchet matching that proves $d_{\mathrm{F}}(P, Q) \leqslant \varepsilon$.

What remains is to prove the converse. Suppose we have a cycle Q in G that has a Fréchet distance of at most ε to P. We must show that there is a cycle of admissible transitions that corresponds to Q. Recall that we split the parameter space of P into sector points $\mathcal{S}$. Let $\mathcal{P} = \{P(s) \mid s \in \mathcal{S}\}$ denote the corresponding points of P. We show that Q admits a Fréchet matching that matches each vertex q_i of Q to some point $p_i \in \mathcal{P}$. This implies that an admissible transition exists from an interval of edge (q_{i-1}, q_i) to (q_i, q_{i+1}) at this sector point; this in turn proves that there is a cycle of transitions that corresponds to Q. Consider some Fréchet matching μ and let k denote the number of vertices q_i—matched to p_i according to μ—such that $p_i \notin \mathcal{P}$. If $k = 0$, then we are done. For $k > 0$, we show that k can be reduced by at least one. Let q_i be a vertex of Q that is matched to p_i according to μ and $p_i \notin \mathcal{P}$. Let p'_i be the first point in $\mathcal{P}$ after p_i. In the free-space diagram, we observe that the entire region above q_i, left of p'_i and below (and to the right

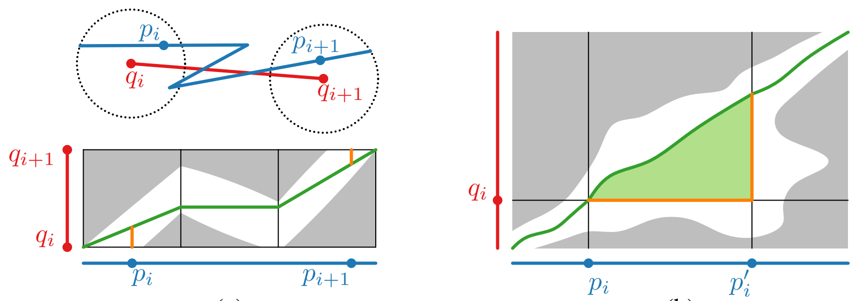


Figure 6.18 (a) We may use a matching for an interval to find a matching between edge (q_i, q_{i+1}) and the subcurve between p_i and p_{i+1}. (b) Sketch of a free-space diagram with matching μ in dark green. By construction, the light green region must be part of the free space. The change from μ to μ', indicated in orange, ensures that q_i is matched to a point $p'_i \in \mathcal{P}$.

of) μ must be part of the free space (see Figure 6.18(b)): because μ is part of the free space, any point—corresponding to a vertex of Q—that is not part of the free space in this region implies that p'_i is not the first sector point after p_i. Convexity of the cells [10] implies that all other points are also part of the free space. Hence, we may change μ into a new Fréchet matching μ' that matches q_i to p'_i without changing the matching after p'_i or before p_i. Note that any other vertices crossed in between p_i and p'_i are now also matched to p'_i. We conclude that k is reduced by at least one.

6.3.2 Brute-force optimization

A naive brute-force algorithm can be obtained by simply trying all simple cycles in G and computing whether the Fréchet distance to P does not exceed ε. A downside of this approach is that the Fréchet distance needs to be computed repeatedly, for an exponential number of cycles. Using the interval graph, it is straightforward to write a brute-force algorithm that avoids this repeated computation of the Fréchet distance.

First, we pick the sector (s_i, s_{i+1}) that is spanned by a minimum number of intervals. For each interval I, we run a recursive brute-force algorithm to find a cycle of transitions—corresponding to a simple cycle in G—from the transition point of I corresponding to sector point s_{i+1} to the transition point corresponding to s_i. This cycle uses only admissible transitions: upon finding a cycle, we already know that the Fréchet distance is at most ε. During the recursion we maintain the number of bends in the current path as well as the minimum number of bends in any cycle that was found. When the number of bends in the current path is greater than or equal to the minimum number of bends, the recursion is stopped: completing the path to a cycle cannot result in less bends.

There are—roughly speaking—two ways to find a cycle: we either traverse the transition points or the intervals. Traversing the transition points is conceptually easier. However, often there are multiple transition points that can be used to go from one interval to the next on a cycle. Many identical cycles are explored in such a situation. Therefore, it is better to work on an interval-basis. We maintain a list of intervals that have been used and mark the corresponding edges and vertices in the graph accordingly. Moreover, we maintain the earliest transition point t that can be used to reach the last interval in the current list (constrained by using that particular sequence of intervals). Let I denote the last interval and assume its (directed) edge is (u, v). For each interval I' that is part of an edge (v, w), we find the first transition point t'—in or after the sector point of t—that admits a transition from I to I'. We recurse on each of these intervals using t' as the new earliest transition point. We process these intervals in reverse order of their endpoint (i.e., the interval that can represent the largest part of curve P after the current transition point t). When the very first interval is reached (in sector point s_i or an earlier transition point), we have computed a new cycle with a minimal number of bends.

6.3.3 ILP formulation

Rather than using an explicit brute-force solution, an integer linear program (ILP) can be used to model NP-hard optimization problems. Due to their wide-spread applicability for optimization problems, ILPs have been studied intensively and a number of implementations exist that perform well in practice. We present an ILP formulation to solve the simple map-matching problem, using the interval graph described in Section 6.3.1. This may allow us to leverage optimized ILP implementations. We describe the ILP formulation by introducing variables, an optimization criterion and linear constraints.

Variables. All variables in the ILP formulation are constrained to be either 0 or 1; the ILP is a 0-1 linear program. We introduce the following variables.

- A variable t for each transition point, to describe a cycle of admissible transitions in the interval graph.
- An indicator variable I for each interval, indicating whether one or more of its transition points are used.
- An indicator variable B_v for each vertex v, indicating whether the described cycle causes a bend at v.
- An additional indicator variable U_I for each interval I in a single given sector.

The last variable is required to "bootstrap" the ILP: most constraints work based on the assumption that something is "used". Hence, if nothing is used, all constraints are satisfied. The last variable is used to alleviate this problem.

Optimization criterion. We wish to minimize the number of bends in the schematization. This is easily formalized using the variables B_v as follows.

$$\text{Minimize} \sum_{v \in V} B_v$$

Constraints. We first introduce constraints to express that the transition points describe a cycle. Let $\mathcal{A}_{\text{out}}(t)$ denote the admissible transition endpoints that are reachable from transition point t with one transition: $\mathcal{A}_{\text{out}}(t) = \{t' \mid (t, t') \text{ is an admissible transition}\}$. We use $\mathcal{A}_{\text{in}}(t)$ to denote its inverse, $\mathcal{A}_{\text{in}}(t) = \{t' \mid t \in \mathcal{A}_{\text{out}}(t')\}$. For convenience, we assume that the previous and next transition point on the same interval (if they exist) are also in $\mathcal{A}_{\text{in}}(t)$ and $\mathcal{A}_{\text{out}}(t)$ respectively. To ensure that the transition points form a cycle, we add two types of constraints. The first expresses that, if a transition point is used, at least one of its admissible transitions is also used (that is, the endpoint of this transition is set to 1). The second expresses that a transition point has at most one used incoming transition, regardless of whether this transition point is used.

$$t - \sum_{t' \in \mathcal{A}_{\text{out}}(t)} t' \leqslant 0 \text{ , for all transition points } t \tag{6.1}$$

$$\sum_{t' \in \mathcal{A}_{\text{in}}(t)} t' \leqslant 1 \text{ , for all transition points } t \tag{6.2}$$

We now introduce the constraints for I, the indicator variables of intervals. An interval is “used” in the cycle if at least one of its transition points is used. We denote the transition points of interval I by $\mathcal{T}(I)$; let k denote the number of transition points. Constraint 6.3, given below, then captures the constraint that the use of a transition point implies the use of an interval. Note that this formulation allows an interval to be considered “used” even if no transition points are used. However, the simplicity constraints and minimization criteria ensure that this does not occur in a minimal solution.

$$\sum_{t \in \mathcal{T}(I)} t - k \cdot I \leqslant 0 \text{ , for all intervals } I \tag{6.3}$$

Let us now turn to enforcing a simple result. To this end, we require that every edge uses at most one interval and every vertex uses at most one outgoing edge. We denote the intervals of an edge (u, v) by $\mathcal{I}(u, v)$. An edge is used if one of its intervals is used; thus the sum over all variables can be used as an indicator for the use of an edge. Therefore, the following constraint expresses that every vertex uses at most one outgoing edge. Note that it directly implies that at most one interval per edge is used.

$$\sum_{(u,v) \in E} \sum_{I \in \mathcal{I}(u,v)} I \leqslant 1 \text{ , for all } u \in V \tag{6.4}$$

The constraints above are all implications or cannot force the use of a transition point. Hence, they can be satisfied by setting all variables to 0, resulting in an invalid solution. So far, the formulation lacks constraints that cause at least something to be used. We observe that in any sector, exactly one of the intervals must use both transition points that define the sector. We simply need something to be used to make the ILP work. We model this as follows. Let (s_i, s_{i+1}) denote the sector that contains a minimal number of intervals; let $\mathcal{I}(s_i, s_{i+1})$ denote these intervals. For each interval $I \in \mathcal{I}(s_i, s_{i+1})$, we introduce a variable U_I. We use this variable to indicate that this interval must use both transition points. We denote the variables of these transition points by $t_{I,i}$ and $t_{I,i+1}$. Exactly one of the U_I variables is set to 1 to force the use of these transition points. This leads us to the following two constraints.

$$\sum_{I \in \mathcal{I}(s_i, s_{i+1})} U_I = 1 \text{ , for one sector } (s_i, s_{i+1}) \tag{6.5}$$

$$2U_I - t_{I,i} - t_{I,i+1} \leqslant 0 \text{ , for all intervals } I \in \mathcal{I}(s_i, s_{i+1}) \tag{6.6}$$

Constraints 6.1 to 6.6 ensure that the variables describe a simple cycle. However, we measure the number of bends using the variables B_v. We must ensure that this variable is 1 only if the cycle indeed causes a bend at the vertex. This is the case if any incoming edge (u, v) is used (i.e., any of its intervals is used), unless the next edge on the cycle is the straight continuation of (u, v). We denote this special edge by $N(u, v)$. Again, we use $\mathcal{I}(u, v)$ to indicate the intervals of an edge; we can express the use of an edge as the sum over its interval variables. This leads to Constraint 6.7 as given below. Note that the minimization criterion ensures that B_v is set to 1 only if it is strictly necessary.

$$\sum_{I \in \mathcal{I}(u,v)} I - \sum_{I \in \mathcal{I}(N(u,v))} I - B_v \leqslant 0 \text{ , for all edges } (u, v) \in E \tag{6.7}$$

Complexity analysis. Let us now analyze the maximal number of variables and constraints of the ILP. To this end, we use n to indicate the number of vertices (and edges) of the closed polygonal curve P. The number of vertices of G is denoted by m; since G is a plane graph, its number of edges is $O(m)$. An important factor in the complexity is the number of resulting sectors. Though the number of sectors is at most $2mn$, this occurs only in very degenerate cases. Therefore, we indicate the number of sectors by k to explicitly model this dependency. In addition, we assume that the maximal degree of any vertex is bounded by a low constant.

After constructing the interval graph, it is straightforward to generate the ILP in time proportional to the number of vertices and constraints. Hence, we restrict our analysis to the number of variables and constraints. For each of the variable types, we give an upper bound on the number of variables we need.

- Number of transition point variables. As the number of sectors is bounded by k, we know that the number of transition points is bounded by k times the number of intervals. Hence, there are $O(kmn)$ transition points. This is a rather coarse upper bound: the number of sectors that an interval spans is expected to be is much lower than k.
- Number of interval variables I. Each interval of edge (u, v) must start a unique sector point of $\mathcal{S}_u$. As there are $O(m)$ edges and $|\mathcal{S}_u| \leqslant 2n$, we know that there are at most $O(mn)$ intervals. This assumes that all edges of a polygon lie within distance ε of all vertices; this is rather degenerate for reasonable values of ε. Hence, we may expect the number of intervals to be much smaller in practice.

- Number of bend variables B_v. A bend variable corresponds exactly to one vertex, thus there are exactly m bend variables.
- Number of indicator variables U_I. One variable is introduced for each interval that is contained in the sector containing the minimum number of intervals. A trivial upper bound is the number of intervals, $O(mn)$. Again, we would expect the actual minimum to be much smaller—possibly constant—in practice.

Similarly, we give the number of constraints and terms per constraint type.

(6.1) There are at most $O(kmn)$ transition points, thus the number of constraints of this type is bounded by the same number. The number of terms in one constraint is bounded by $O(n)$, assuming a low constant degree of a vertex.
(6.2) Similar to the previous constraint, there are at most $O(kmn)$ constraints of this type; the number of terms in a single constraint is at most $O(n)$.
(6.3) With one constraint per interval, there are $O(mn)$ constraints. The number of terms is bounded by k as each term corresponds to a transition point of one interval.
(6.4) There are m constraints of this type: one for each vertex of G. Each term corresponds to an interval of an outgoing edge. As the number of intervals of one edge is at most $O(n)$, there are $O(n)$ terms per constraint assuming a constant degree.
(6.5) There is exactly 1 constraint of this type: only for the sector with minimal number of intervals. The terms are exactly the variables U_I. The number of terms is therefore coarsely upper bounded by $O(mn)$.
(6.6) There is one constraint for each variable U_I, each having only 3 terms. The number of constraints is thus $O(mn)$.
(6.7) There is one constraint for each edge; there are $O(m)$ edges. The number of terms is bounded by $O(n)$ as both (u, v) and $N(u, v)$ may have up to $2n$ intervals.

We conclude that the following (coarse) upper bounds hold: the number of variables is $O(kmn)$; the number of constraints is $O(kmn)$; the number of terms is $O(kmn^2)$. An ILP can be represented using a matrix where each column corresponds to a variable and each row to a constraint. An entry is the linear factor in the corresponding variable-constraint pair (0 if the constraint does not depend on the variable). From the bounds above, we conclude that the matrix is not necessarily sparse, though not all $O((kmn)^2)$ entries can have nonzero values.

Constraint 6.2 and Constraint 6.4 enforce a simple cycle in the interval graph that corresponds to a simple cycle in the given graph G. Even without these constraints, the result may be a simple cycle in G; we may expect that many of these constraints are not necessary to obtain a simple cycle. Therefore, we choose to mark these as *lazy constraints*. These constraints are added to ILP only if a potential solution is found that violates the constraint. In the next section, we briefly discuss some experimental results based on this ILP formulation.

6.3.4 Experimental results

In this section we discuss some experimental results. The used territorial outlines, graphs, ε-values and corresponding schematic outlines are illustrated in Figures 6.19 through 6.23. We first discuss the resulting schematic outlines; afterwards, we consider the performance of the ILP formulation and the brute-force method.

Schematization. The only given input is the territorial outline. To compute a schematization via map matching, we need to specify a plane graph that adheres to a desired orientation set $\mathcal{C}$. We generate these graphs based on equally spaced lines for each orientation. Moreover, we also need to specify a value for ε. The value of ε and the spacing between the lines were configured manually. The illustrated results were obtained using the brute-force method of Section 6.3.2. In each figure, we visually represent the constructed graph and the specified value for ε. The number of bends in the resulting schematization are indicated in the corresponding captions.

In Figure 6.19, we illustrate octilinear results for Australia. In (b–c), we show the used graph and value of ε, resulting in a schematization with 9 bends. To obtain a more complex schematization, we need to decrease the value of ε. However, if this value is decreased too much, the map-matching problem does not admit a solution. Therefore, we also refine the graph for (d–e). Ideally, we would specify a very dense graph and change only the value of ε to control complexity of the resulting schematization. However, due to the computational complexity, it is infeasible to combine a complex graph with a high value of ε. Therefore, we must often alter the graph in addition to the value of ε.

Another octilinear schematization is given in Figure 6.20: it shows a schematization of Languedoc-Roussillon (a department of France). Here, we see two possibilities to improve the schematization. First, on the eastern side, we could shift one of the bends to more accurately resemble the territorial outline. However, this does not improve the number of bends. Hence, for our optimization methods, these are considered "equal". The suggested improvement is "locally correct": it is the best local choice. Can we incorporate locally correct Fréchet matchings (see Chapter 5) in our computation? However, this likely means slowing down a possibly lengthy computation. A second improvement is suggested on the eastern border: again, the territorial outline could be represented more accurately without increasing the number of bends. However, this solution cannot be found by our map-matching approach: the indicated change does not lie within the given graph. This indicates a drawback of formulating it via map matching: we need to specify a suitable graph. On the other hand, these improvements are mostly visible when comparing the result directly with the original outline. The general structures seem to be maintained rather accurately. An important question is whether such "local fine-tuning" of the schematization has a great impact on the recognizability of the map.

Our map-matching approach is not limited to octilinear results; we can use different orientation sets to generate a graph. Figure 6.21 illustrates hexilinear results and Figure 6.23 illustrates rectilinear, hexilinear and octilinear results. Irregular orientation sets can also be used to generate a graph, as illustrated in Figure 6.22. In fact, any plane graph can be used in our methods: the graph need not be $\mathcal{C}$-oriented and may even model smooth curves using approximating polygonal curves (see Chapter 9).

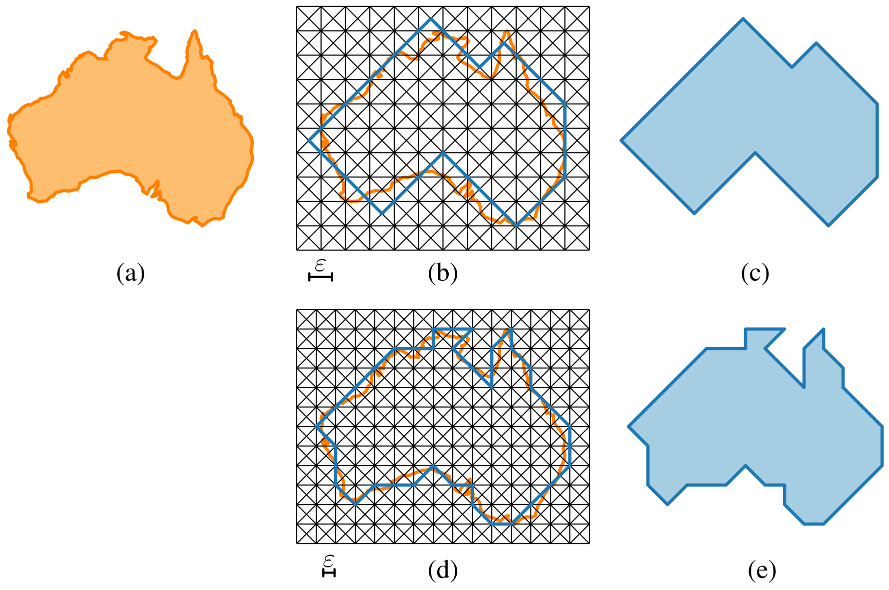


Figure 6.19 Results for schematizing Australia using C_4. (a) Territorial outline. (b–c) Optimal result with 9 bends. (d–e) Optimal result with 25 bends. Compared to (b–c), ε is reduced and the graph is refined.

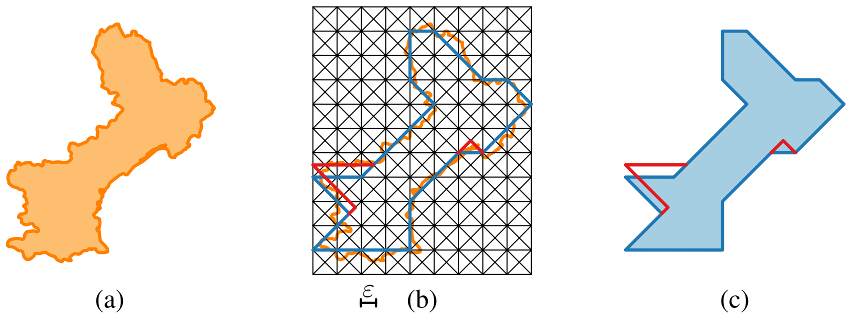


Figure 6.20 Result for schematizing Languedoc-Roussillon using $\mathcal{C}_4$. (a) Territorial outline. (b–c) Optimal result with 10 bends. Possible improvements are indicated in red.

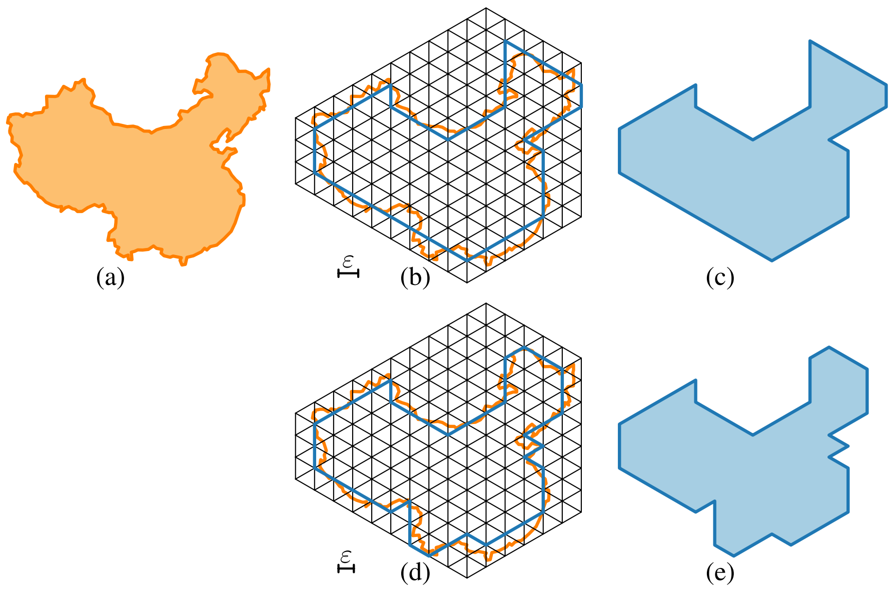


Figure 6.21 Results for schematizing China using $C_3(\frac{\pi}{2})$, using one graph but different values of ε. (a) Territorial outline. (b–c) Optimal result with 13 bends. (d–e) Optimal result with 21 bends.

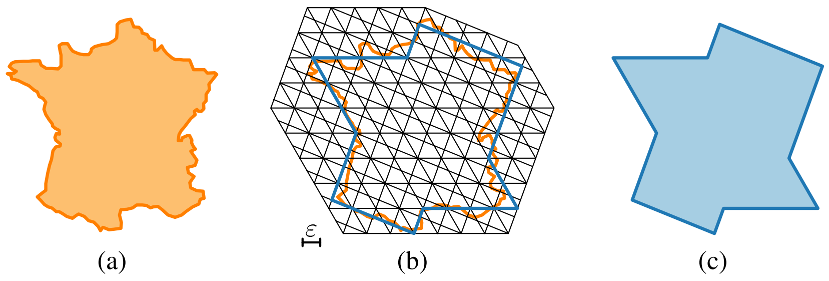


Figure 6.22 Result for schematizing France using an irregular set $\mathcal{C} = \{0, \frac{7\pi}{18}, \frac{12\pi}{18}, \frac{16\pi}{18}\}$. (a) Territorial outline. (b–c) Optimal result with 10 bends.

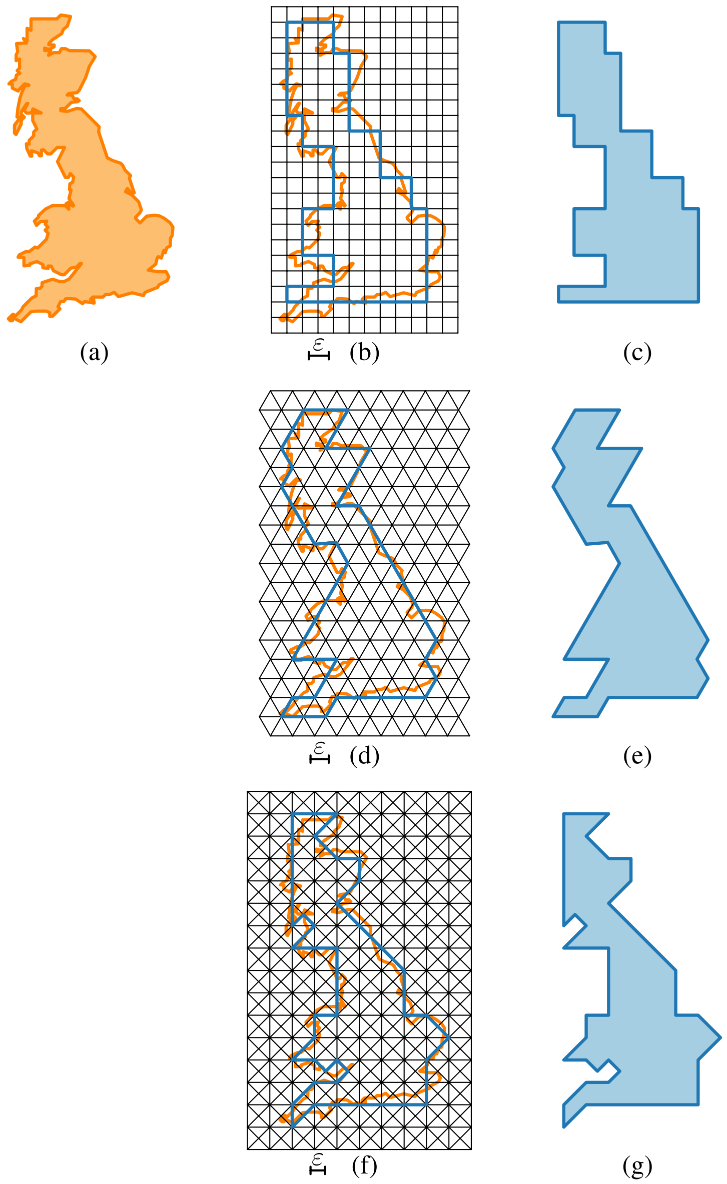


Figure 6.23 Results for schematizing Great Britain using orientation sets $\mathcal{C}_2$, $\mathcal{C}_3$ and $\mathcal{C}_4$. (a) Territorial outline. (b–c) Optimal rectilinear result with 22 bends. (d–e) Optimal hexilinear result with 23 bends. (f–g) Optimal octilinear result with 31 bends.

Performance. We now discuss the performance of the ILP formulation in comparison to the brute-force method. We implemented both approaches in Java and used *IBM ILOG CPLEX Optimization Studio 12.6* (CPLEX for short) to solve the integer linear programs, using the built-in support for lazy constraints. These experiments were run on a DELL Precision M4500 Laptop with 4GB RAM and an Intel Core i5 M560 processor with two cores of 2.67GHz each (two threads per core). CPLEX uses up to four threads. For our brute-force implementation, we also allowed the use of up to four threads. A "unit of work" for a thread is one interval that spans the sector with a minimal number of intervals: if only few intervals span this sector, then this also bounds the number of effective threads.

Using CPLEX in its default configuration caused very high memory usage. Therefore, we allowed CPLEX to write compressed files to disk to use more storage. Furthermore, we set the branching strategy to "strong branching": this increases computation time per branch node but reduces the number of nodes visited. We tried three different node-selection strategies: "best bound", "best estimate" and "depth-first search".

Table 6.1 lists statistics for the various instances (Figure 6.19 through Figure 6.23). This includes the number of vertices in the given closed curve P and in the constructed graph G. For the latter, we count only vertices that lie within ε distance of P: all other vertices are removed from the graph during the transformation. The table also provides some complexity measures for the interval graph (sectors, intervals, transition points) and the resulting ILP (variables, constraints, lazy constraints, terms). Finally, it also lists the optimal solution. Table 6.2 lists the results obtained via CPLEX and via the brute-force algorithm in terms of computation time and the best solution found. For these experiments, we used a 6-hour time limit for computation; aborted computations are indicated with an asterisk. For the brute-force algorithm, we list values obtained through a single-thread implementation and through a parallel implementation with up to 4 threads.

The brute-force method is significantly faster than the ILP formulation; the figures in this section illustrate the results obtained using this method. Comparing the computation times between the single-thread and multithreaded implementation, we see that the use of multiple threads helps speed up the computation. However, the speed up is far from a factor 4. This is likely due to the additional memory required and the need for some synchronization between the threads. However, for the instances in Figure 6.23, the computation times are rather high. Only for Figure 6.23(f–g), the 6-hour time limit was not enough to verify optimality of the found solution. We removed the time bound to verify that 31 is indeed the optimal solution; this took roughly 24 hours.

Unfortunately, CPLEX did not find any solution for a number of instances. For some instances, it did find the optimal solution, although it lacked time to verify its optimality (regardless of the node-selection strategy). The depth-first-search strategy was most effective in finding a (suboptimal) solution, though it sometimes deviates more from the optimal solution than the other strategies. Possibly, the problem resides in the map-matching problem itself: we know that no efficient approximation algorithms exist (Corollary 6.2). A common strategy applied in solving ILPs is to relax the integrality constraints and round the resulting solution. A linear program can be solved optimally in polynomial time. Hence, the result of this strategy cannot yield a good approximate solution to the problem unless P=NP. This may partially explain the performance of the ILP method.

Table 6.1 Complexities for our test instances, referred to by their figure number. The columns $|P|$ and $|G|$ list the number of vertices in curve P and graph G respectively. ILP complexity is measured after reduction steps performed by CPLEX.

Instance			**Interval graph**			**Integer Linear Program**				**Optimum**
Figure	$\lvert P\rvert$	$\lvert G\rvert$	*Sectors*	*Intervals*	*Transition pts*	*Variables*	*Constraints*	*Lazy*	*Terms*	*Bends*
6.19(b–c)	223	127	314	580	9 148	9 786	10 222	7 248	52 863	9
6.19(d–e)	223	94	214	300	1 994	2 307	2 489	1 367	9 052	25
6.20(b–c)	847	91	234	325	2 750	3 086	3 305	1 881	13 277	15
6.21(b–c)	229	87	212	338	2 975	3 397	3 649	2 188	15 083	13
6.21(d–e)	229	69	162	223	1 363	1 618	1 761	896	6 186	21
6.22(b–c)	170	119	264	434	4 685	5 132	5 440	3 488	23 508	10
6.23(b–c)	250	178	528	625	10 397	10 947	11 328	8 017	48 144	22
6.23(d–e)	250	87	276	359	2 939	3 369	3 620	1 997	14 024	23
6.23(f–g)	250	126	364	497	4 367	4 846	5 179	3 039	20 841	31

Table 6.2 Statistics of our experiments. CPLEX strategies are "best bound" (BB), "best estimate" (BE) and "depth-first search" (DFS). For the brute-force method, the number of threads is indicated. Computations indicated with a * were aborted at the time limit. Optimal solutions are indicated in bold.

Instance	**CPLEX**						**Brute-force**			
Figure	*Time (BB)*	*Bends*	*Time (BE)*	*Bends*	*Time (DFS)*	*Bends*	*Time (1)*	*Bends*	*Time (4)*	*Bends*
6.19(b–c)	6h*	∞	6h*	∞	6h*	∞	9s	**9**	4s	**9**
6.19(d–e)	6h*	**25**	6h*	**25**	6h*	**25**	4m13s	**25**	3m59s	**25**
6.20(b–c)	6h*	**15**	6h*	∞	6h*	**15**	2s	**15**	1s	**15**
6.21(b–c)	6h*	14	6h*	14	6h*	34	6s	**13**	5s	**13**
6.21(d–e)	6h*	**21**	6h*	**21**	6h*	**21**	4s	**21**	3s	**21**
6.22(b–c)	6h*	∞	6h*	∞	6h*	11	0s	**10**	0s	**10**
6.23(b–c)	6h*	∞	6h*	∞	6h*	∞	1h26m	**22**	49m27s	**22**
6.23(d–e)	6h*	∞	6h*	∞	6h*	**23**	1m26s	**23**	30s	**23**
6.23(f–g)	6h*	∞	6h*	∞	6h*	∞	6h*	**31**	6h*	**31**

Table 6.3 Lower bound, upper bound and the integrality gap for the three used CPLEX strategies.

Instance	**Integrality gap**								
Figure	*Best bound*			*Best estimate*			*Depth-first search*		
6.19(b–c)	3.00	∞		3.00	∞		0.50	∞	
6.19(d–e)	9.00	25	64.00%	7.00	25	72.00%	2.51	25	89.96%
6.20(b–c)	7.67	15	48.89%	6.00	∞		2.01	15	86.62%
6.21(b–c)	4.99	14	64.36%	3.99	14	71.48%	2.00	34	94.12%
6.21(d–e)	11.01	21	47.59%	5.44	21	74.07%	2.00	21	90.47%
6.22(b–c)	5.00	∞		4.99	∞		1.02	11	90.70%
6.23(b–c)	4.00	∞		4.00	∞		1.00	∞	
6.23(d–e)	7.00	∞		7.08	∞		2.09	23	90.91%
6.23(f–g)	9.00	∞		8.85	∞		3.00	∞	

In finding a solution, CPLEX maintains a lower bound on the number of bends in any solution and an upper bound (the number of bends in the best solution so far). The difference between this lower and upper bound is referred to as the integrality gap. Table 6.3 lists these bounds and the integrality gap. As we may see in this table, the best-bound and best-estimate strategies perform much better in terms of the lower bound. If they found a solution, the integrality gap is much smaller. This indicates that CPLEX is closer to finding an optimal solution with these strategies compared to depth-first search.

The progression of the intermediate solution (in number of bends) over time is illustrated in Figure 6.24. This figure shows that the brute-force method often finds a (suboptimal) solution within 1 second. For most instances, an optimal solution is found within 5 minutes. In contrast, it typically takes much longer for the ILP to find a first solution, if it finds any at all. Of the three CPLEX strategies, the depth-first-search strategy is most efficient in finding a first solution. However, this strategy performs worse in long-term computations as was concluded above.

Figure 6.25 shows the “approximation ratio” of intermediate solutions in comparison to the optimal solution for the brute-force method. From this, we learn that the first solutions are typically a 2.5-approximation or better. After only 2 seconds, all test instances achieved an approximation ratio below 1.25. The question is how suitable these approximate solutions are as schematization. Figure 6.26(a–i) shows all solutions that were found for the instance of Figure 6.23(d–e). It seems that these intermediate solutions are reasonable, though it may have unwarranted bends at comparatively arbitrary positions. We think that this reasonable quality is due to our interval processing order: we always first try the interval that can reach the furthest sector point. This choice corresponds to the edge that represents the local geometry rather well. To support this hypothesis, Figure 6.26(j) shows a result with 31 bends using a random processing order: it is significantly worse than the result in Figure 6.26(a). We also observe that some suboptimal patterns repeat itself in the computation. To improve computation time, it is desirable to find a method that avoids recomputing these local patterns.

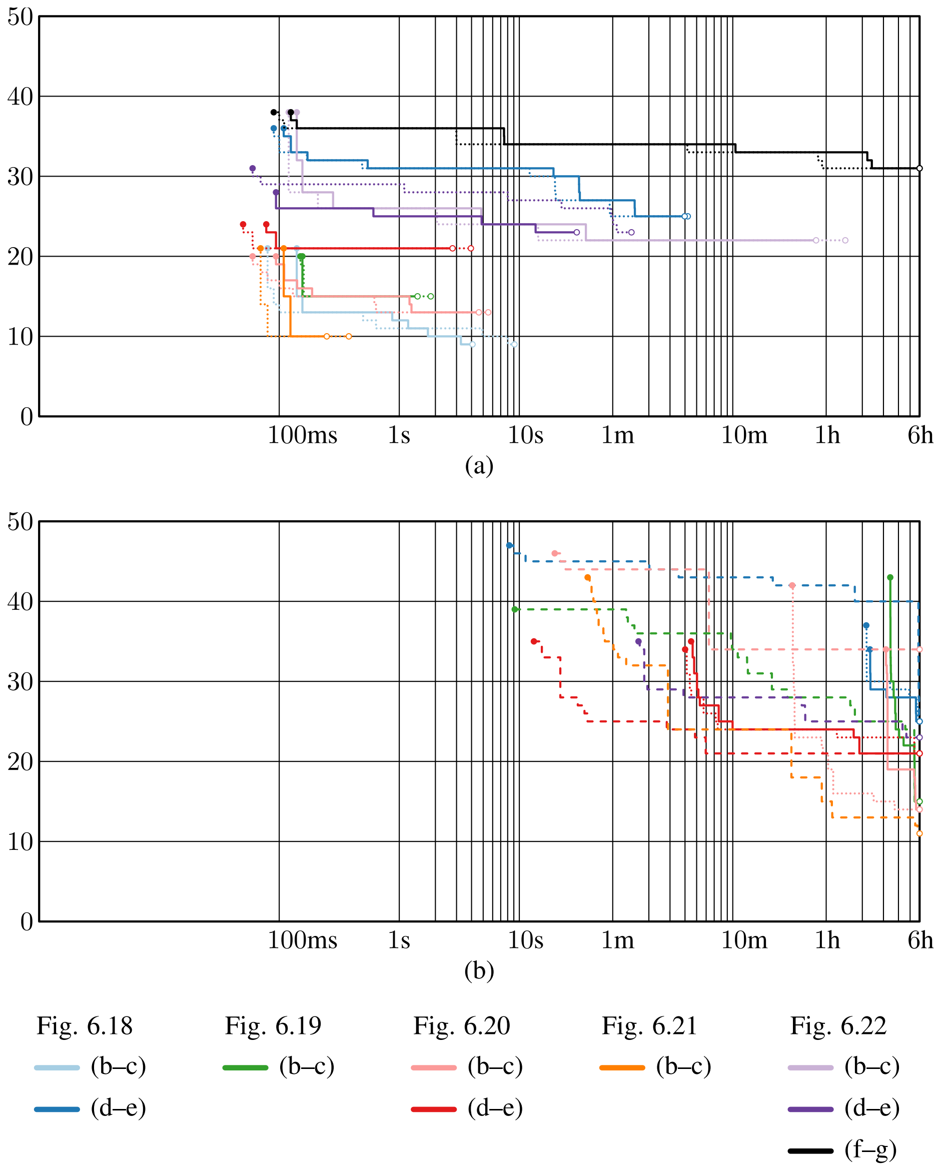


Figure 6.24 Progression of number of bends over time during computation. Time is given on a logarithmic scale (in milliseconds). (a) Brute-force results. Solid lines and dotted lines indicate the multithreaded and single-thread variant respectively. (b) CPLEX results. Solid lines, dashed lines, and dotted lines indicate “best bound”, “best estimate” and “depth-first search” strategies respectively.

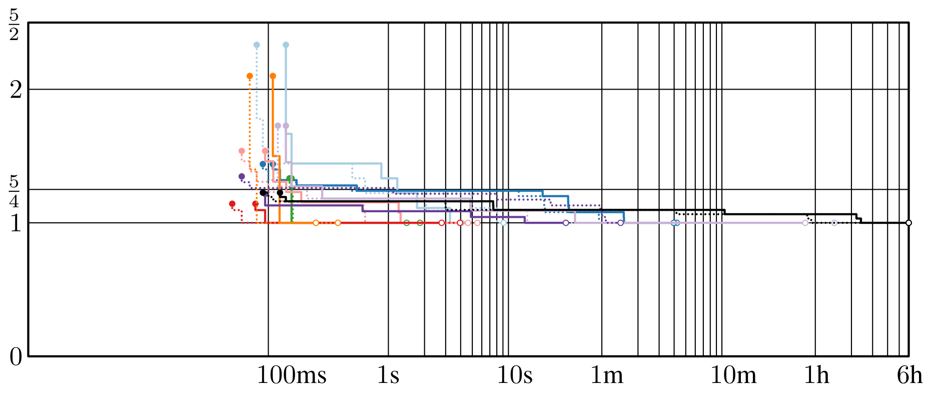


Figure 6.25 Progression of number of bends over time relative to optimal solution for the brute-force method. Time is given on a logarithmic scale (in milliseconds). The representation and colors correspond to those of Figure 6.24(a).

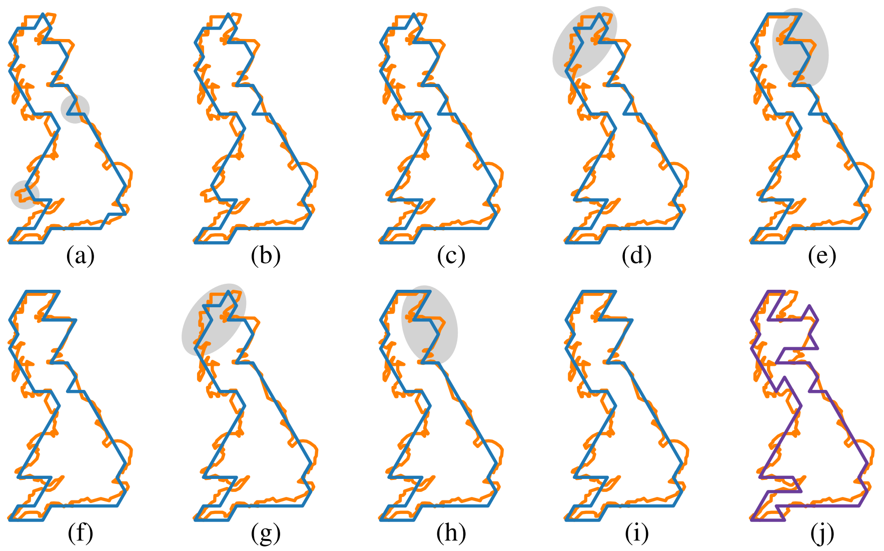


Figure 6.26 Intermediate results (a–h) and optimal result (i) for Figure 6.23(d–e), starting at 31 and ending at 23 bends. Some unnecessary bends (a) and repeated patterns (d,g; e,h) are highlighted. (j) Random interval selection leads to poor approximate solutions.

6.4 Conclusions

We have shown that it is NP-complete to determine the existence of a simple cycle in a plane graph such that its Fréchet distance to a given simple closed polygonal curve is at most 1. In addition, we considered the implications of our proof. Most importantly, it shows that no polynomial-time algorithm exists with a polynomially bounded approximation factor, unless P=NP. It also implies that no FPT algorithm can be obtained by characterizing polygons via their turn profile and trying all possible profiles. Finally, the proof readily extends to a number of variants, including the use of the weak or discrete Fréchet distance.

We introduced an interval graph that allows us to formulate the simple map-matching problem as an ILP. The interval graph also leads to a simple brute-force algorithm that avoids repeated computation of the Fréchet distance in the exponential-time search. Based on these two methods, we discussed some experimental results on small instances. These results indicate that solving the given ILP remains infeasible, though other ILP formulations may exist that are more efficient. The brute-force algorithm performs better and can often quickly find an optimal or close-to-optimal result. The obtained results show that our method maintains the most salient structures of the given territorial outlines. This encourages further research for the simple map-matching problem.

Open questions. An interesting question is to see what further restrictions we can pose on the input. For example, the constructed graph is relatively sparse and constructed specifically for the proof. With schematization in mind, we define our own graph; typically, this graph is constructed from a line arrangement using only a few line orientations. Is the problem NP-complete if we consider, for example, a square grid or equilateral triangle grid? The same construction does not seem to work directly. Though variable and propagation gadgets can be defined, clause gadgets pose a challenge.

Another open question is whether the (general) simple map-matching problem admits solutions via other ways of dealing with NP-hard problems. For example, does it admit a moderately exponential-time algorithm, that is, an algorithm in which the exponential dependency has a low base number?

In our experimental results, we observed that minimizing only the number of bends is not always sufficient to get the best visual result. In some cases, the resulting schematic outline locally deviates further than necessary from the geographic outline. This is caused by a value for ε that is too large for the local situation. However, a large value for ε may be required to allow for a simple cycle elsewhere along the outline. This raises the question whether we can extend local correctness, as defined in Chapter 5, to map matching. A foremost question is how local correctness interacts with simplicity: a local improvement to the schematization may be hindered by the simplicity constraint.

Chapter 7

Schematization via Iterative Replacement

In the previous chapter we studied an optimization problem for schematization by modeling it as a map-matching problem. Unfortunately, this leads to an NP-hard problem, even in an approximation setting. In this chapter we investigate a heuristic *iterative algorithm* that is fast and works well in practice. This algorithm repeatedly applies operations that cause local changes to a subdivision. The local changes are chosen such that they have the lowest impact on the shape: in other words, it aims to maintain similarity in a greedy way. The operations are performed until the desired complexity is reached. We describe an *edge-move* operation that, when applied in pairs, results in a simplification algorithm (described in Section 7.2) with the following properties:

- the result has the same topology as the input;
- the area of each face is preserved;
- every orientation in the result occurs in the original input;
- every nonconvex simple polygon admits a pair of edge-moves.

The operation is illustrated in Figure 7.1. This is a simplification algorithm and as such it does not directly result in $\mathcal{C}$-oriented schematization. However, as edge-moves do not introduce new orientations, the result is guaranteed to be $\mathcal{C}$-oriented if the input is $\mathcal{C}$-oriented. In Section 7.3 we describe an algorithm to perform *angular restriction*, that is, to convert any subdivision into an area-equivalent $\mathcal{C}$-oriented one (for both regular and irregular sets $\mathcal{C}$). Combining the two algorithms yields a $\mathcal{C}$-oriented schematization algorithm that preserves both area and topology. In Section 7.4 we discuss results obtained by applying these algorithms to various territorial outlines; a preview of these results is given in Figure 7.2. We conclude that, with user-defined orientation sets and complexities, the resulting schematic outlines maintain the high-level structures of the geographic outlines. Note that our algorithms are also suitable to simplify other shapes, such as buildings (see Chapter 8).

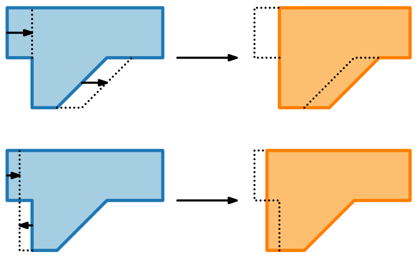


Figure 7.1 A combination of two edge-moves preserves the area and reduces the complexity; no new orientations are introduced. Multiple options may be possible.

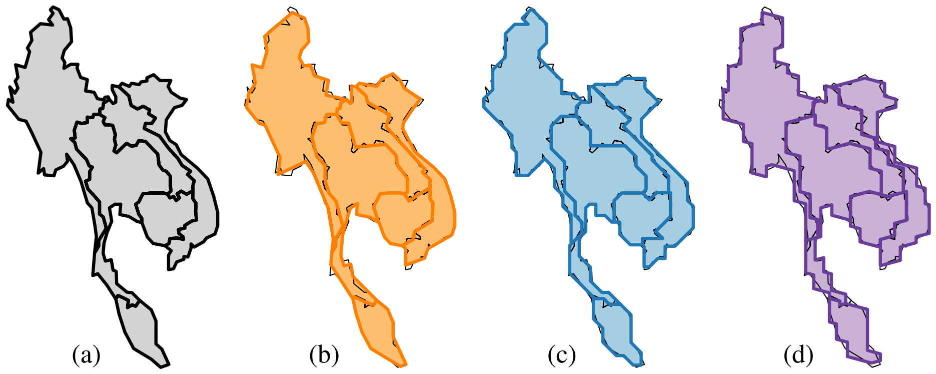


Figure 7.2 Results of our algorithm for a group of countries in southeast Asia, each with 130 edges. The area of each country is preserved. (a) Input subdivision. (b) Simplification. (c) Octilinear schematization. (d) Schematization with irregular orientations.

Related work—simplification. If we formulate topologically safe simplification as an optimization problem (e.g. minimize the complexity of the result), many variants are NP-hard [95] and thus are unlikely to admit an efficient algorithm. Nonetheless, numerous methods exist for the simplification of lines, polygons and subdivisions. These either simplify heuristically, disregard topology, or guarantee correct topology by imposing additional constraints that are stricter than necessary. One of the most popular methods in automated cartography is the Douglas-Peucker method [69] (note that it has also been independently developed by Ramer [143]). This vertex-restricted algorithm does not guarantee a minimal complexity and discards topology—though extensions have been made to address this issue [156]. Van de Kraats *et al.* [114] discuss the case where the input subdivision represents a printed circuit board. Estkowski and Mitchell [77] describe a heuristic method for simplifying parallel lines, such as elevation contours. Mustafa *et al.* [130] obtain topologically safe simplification by restricting results to Voronoi cells. Van der

Poorten and Jones [139] use the constrained Delaunay triangulation to find and simplify features without changing the input topology. Raposo [144] uses a hexagonal tiling to compute a nonvertex-restricted simplification; he describes a method of "untwisting" the result to remove any intersections.

Imai and Iri [109] transformed vertex-restricted simplification into a shortest-path problem on a suitably defined graph; this approach can be used for both minimizing the complexity and for minimizing some error bound. Chan and Chin [52] proved that this graph can be constructed in $O(n^2)$ time, thus yielding a quadratic-time algorithm for the simplification problem. A similar approach has been followed by a number of methods, e.g. [1, 25, 27]. De Berg *et al.* [25] introduce special input points—"landmarks"—to guarantee a topologically safe result. Bose *et al.* [27] study simplification with an areal displacement criterion for monotone curves. They show that it is NP-hard to decide whether a vertex-restricted area-equivalent solution exists and give an approximation algorithm. These measures are also studied for general curves by Daneshpajouh *et al.* [63].

More akin to the simplification method that is described in this chapter, there are also a number of iterative approaches. A well-known example is the algorithm by Visvalingam-Whyatt [172]. This method repeatedly eliminates the point for which the triangle formed by its incident edges has the lowest area. In its original formulation, topological constraints were not considered; the method can thus be implemented in $O(n \log n)$ time using a priority queue. Tutić and Lapaine [166] give an iterative nonvertex-restricted simplification algorithm that produces area-equivalent results. However, their algorithm is not topologically safe and no prioritization is used. The latter is likely problematic when a very low complexity is desired.

The algorithms mentioned above, however, are suitable only for simplification; they are not designed to obtain $\mathcal{C}$-oriented results.

Related work—schematization. Work on $\mathcal{C}$-oriented schematization with constraints on similarity or recognizability is sparse. For schematization of a single line, Neyer [132] gives an algorithm to compute a $\mathcal{C}$-oriented schematization constrained by a maximal Fréchet distance ε. This method requires that the distance between two consecutive vertices is at least 2ε, reducing its effectiveness to schematize detailed geographic territorial outlines. Merrick and Gudmundsson [126] describe an algorithm to generate $\mathcal{C}$-oriented paths that visits the points of the original curve "in order" and apply this technique to compute metro map layouts. Delling *et al.* [64] describe a method to generate $\mathcal{C}$-oriented route sketches with orthogonal-order constraints for monotone paths. Gemsa *et al.* [88] show that such route sketching problems are NP-hard in general and provide a mixed-integer linear program. However, it is generally not advisable to simplify or schematize each border of a subdivision separately, especially when aiming for a result with low complexity. In a noncartographic setting, Cicerone and Cermignani [59] describe a heuristic method for rectilinear and octilinear schematization based on discretizing space near each vertex. However, this method does not guarantee area-equivalent results nor does it necessarily preserve the correct topology.

However, there is an ample body of work on the $\mathcal{C}$-oriented schematization of networks (e.g. transit maps). Avelar [17] discusses design aspects of schematic networks and presents an iterative algorithm to generate schematic road networks. Cabello *et al.* [47] give an algorithm to produce schematic transit maps that use two or three links per path, if that is possible. Stott *et al.* [163] apply multicriteria-optimization techniques to compute octilinear transit maps. Note that octlinearity is one of their optimization criteria; thus the resulting maps may deviate from a strictly octilinear design. Nöllenburg and Wolff [133] specify seven design rules for octilinear metro maps and present a mixed-integer programming approach to generate metro maps using one edge per path. Wolff [177] provides a survey on metro map construction. Another extensive overview of algorithms for network schematization is given by Swan *et al.* [164]; they study their applicability to automated transit-map construction for web services. Algorithms for network schematization can be used to schematize subdivisions. However, they usually do not take criteria such as shape and size preservation into account. As such, these algorithms are unsuitable to schematize territorial outlines.

Unrestricted schematization largely overlaps with simplification, though some methods explicitly target the creation of schematic maps [20, 56, 78]. A simulated-annealing approach to optimize for parallelism was presented by Reimer and Meulemans [150]. Recently, curved schematization has gained increasing attention, a geometric style that uses smooth curves rather than line segments for representing shape. Manually drawn examples of such curved schematizations can be found, for example, in chorematic diagrams [149] and transit maps [151]. Van Goethem *et al.* [91] describe a topologically safe framework to generate curved schematizations and illustrate this framework with Bézier curves and circular arcs. Circular arcs are chosen such that the result is area-equivalent to the input. However, the framework admits only vertex-restricted methods. Van Dijk *et al.* [66] use this method to create circular-arc schematizations of "focus maps" in which a certain area of interest is enlarged. Using an iterative method, Van Goethem *et al.* [92] describe a nonvertex-restricted algorithm for computing area-equivalent circular-arc schematizations. Van Dijk *et al.* [67, 68] also present an iterative algorithm, focusing on nonvertex-restricted and smooth junctions. For transit maps, Fink *et al.* [79] present a force-directed method to compute a schematized network with Bézier curves. An ILP for concentric circular-arc transit maps was formulated by Fink *et al.* [80].

A number of methods exist that compute a simplification with G^1-continuous circular arcs, often referred to as biarcs [72, 105, 106, 122, 123]. These methods do not explicitly target schematization, but their use could be considered for schematization purposes. However, it is not always desirable to obtain a smooth schematic shape: the use of sharp bends allows for more expressive power. Drysdale *et al.* [72] also describe a method that uses regular (not G^1-continuous) circular arcs, though this method imposes restrictions on "gates" which hinder a high complexity reduction.

We restrict the angles in a subdivision beforehand to obtain $\mathcal{C}$-oriented results. This problem is often encountered in computer graphics: a subdivision in $\mathbb{R}^2$ has to be converted to a similar subdivision in "pixel space", $\mathbb{Z}^2$. This rasterization problem is studied for example by Christ *et al.* [57]. However, such methods require that vertices have integer coordinates and consider only rectilinear orientations. Moreover, these methods do not guarantee that the area of the result is exactly equal to the area of the input.

7.1 Preliminaries

In this chapter we design algorithms to simplify or schematize subdivisions that represent territorial outlines.

Vertex degrees. For simplicity, we assume that all vertices have a degree of at least two and at most four. Since we design our algorithms for territorial outlines, these assumptions are valid on most inputs. If desired, vertices of other degree can be supported as well. Degree-0 vertices could be used as "landmarks" [25]. Degree-1 vertices require an appropriate definition of edge-moves near such vertices. Vertices of degree higher than four pose a problem only for the angular-restriction algorithm presented in Section 7.3. This algorithm may still be applied if a sufficient number of orientations is provided and the incident edges of these junctions are spread reasonably well around the vertex.

Area preservation. The algorithms in this chapter ensure that the faces of the resulting subdivision have the same area as the corresponding faces in the detailed input subdivision. Hence, we refer to our algorithms as *area-preserving* methods. The result is called *area equivalent* to the input. Large (relative) size distortions interfere with an observer's mental map, making a schematization harder to recognize. By ensuring that faces strictly maintain their given size, we avoid this interference. Thus, this property helps to make regions easier to recognize in a set of territorial outlines.

7.2 Simplification

In this section we describe an area-preserving simplification algorithm based on an operation called *edge-move*. We first introduce this operation. Then we describe a heuristic algorithm for iteratively selecting edge-moves in subdivisions. Finally, we show that any nonconvex simple polygon admits edge-moves that preserve topology.

7.2.1 Edge-moves

Below, we give a precise definition of the edge-move operation. We first introduce it for a simple polygon and afterwards extend it to subdivisions.

Definitions and notation. We are given a simple polygon $P = \langle p_0, \ldots, p_{n-1} \rangle$ with n vertices. We denote the edges of ∂P by $e_0, \ldots, e_{n-1}$. The directed edge e_i starts at vertex p_i and ends at vertex p_{i+1}. As with the vertices, we treat the edges modulo n: in other words, e_i is used as a shorthand for $e_{i \bmod n}$. We call an edge *convex* or *reflex* if both its vertices are convex or reflex respectively.

Configurations and edge-moves. Three consecutive edges $\langle e_{i-1}, e_i, e_{i+1} \rangle$ are called a *configuration*. An edge-move always operates on a configuration. We refer to its central edge e_i as its *inner edge*, the other edges are its *outer edges*.

Let $X = \langle e_{i-1}, e_i, e_{i+1} \rangle$ be a configuration of polygon P. The outer edges, e_{i-1} and e_{i+1}, define two *tracks*, infinite lines through the edges. An *edge-move* on X moves e_i

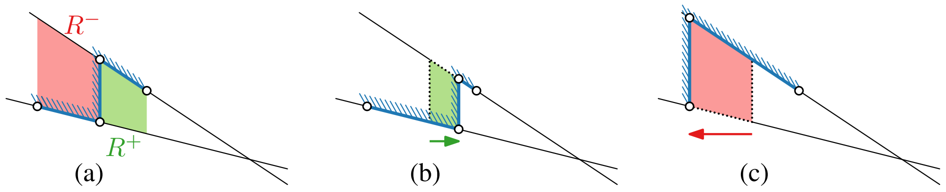


Figure 7.3 (a) A configuration with its positive and negative contraction region. (b) A positive edge-move. (c) A negative contraction.

such that its orientation is preserved and its vertices are on the tracks, making the outer edges longer or shorter. An edge-move is *valid* if at least one vertex of e_i remains on its original outer edge and e_i remains on the same side of or on the intersection point of the tracks (if it exists). A *contraction* is an edge-move that causes one of the edges of X to reach length zero. Contractions are "extremal edge-moves" and reduce the complexity of P. An edge-move is *positive* if it adds area to P and *negative* if it removes area. Figure 7.3 shows some examples.

A configuration supports edge-moves, either positive, negative, or both. A *positive configuration* supports positive edge-moves; a *negative configuration* supports negative edge-moves. The *positive contraction region* $R^+(X)$ of a positive configuration $X = \langle e_{i-1}, e_i, e_{i+1} \rangle$ is the region enclosed by e_i, the tracks, and the position of e_i after a positive contraction. A *feasible positive configuration* is a configuration for which $R^+(X)$ does not contain any points on ∂P, except for those that belong to X. Similarly, we define the *negative contraction region* $R^-(X)$ and a *feasible negative configuration*. If a positive or negative configuration is feasible, then any valid positive or negative edge-move respectively is feasible. If a positive configuration is infeasible, then there is some point on $\partial P \backslash X$ in $R^+(X)$. A point in $\partial P \backslash X \cap R^+(X)$ that is closest to the line through e_i is called a *positive blocking point*. Analogously, X can have a *negative blocking point*. Examples of blocking points are given in Figure 7.4.

Complementary moves. Since we desire an approach that preserves the area of the polygon, we combine two *complementary* feasible configurations. Such a complementary pair consists of one positive and one negative configuration and we execute an edge-move on both simultaneously. The one with the smaller contraction region is contracted, while

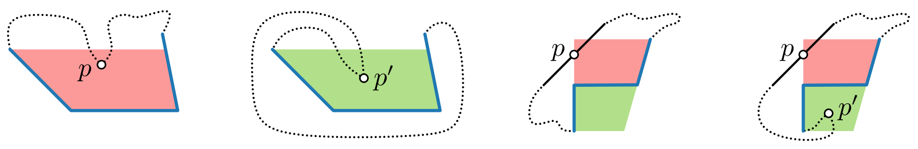


Figure 7.4 Configurations (blue line) with negative and positive blocking points, labeled p and p' respectively. Negative and positive contraction regions are indicated in red and green respectively.

the other is moved just far enough to compensate for the area change. Two configurations *conflict* when they share an edge, unless they share only outer edges and one of these has a convex and a reflex vertex. In this special case the two edge-moves both either shorten or lengthen the shared edge. We call two nonconflicting complementary feasible configurations a *proper configuration pair*.

Subdivisions. We now extend the definition of edge-moves to subdivisions. To preserve the area of each face, it is important to combine only edge-moves of which the inner edges are incident to the same faces. In addition, we must define edge-moves in the presence of junctions (vertices of degree 3 or higher). A configuration with inner vertices of degree 2 supports edge-moves as defined for polygons. However, if an inner vertex of a configuration is a junction of degree 4 or higher, then performing edge-moves on this configuration could alter the topology or result in an area change. Hence, we do not allow moving such junctions; as a consequence any incident edge cannot be moved. For junctions of degree 3, there is some flexibility. The following three cases can occur for an edge-move with inner edge e incident to a vertex v of degree 3 (refer to Figure 7.5).

(a) The other edges of v have the same orientation. These edges must lie on different sides of the line through e. When moving e in either direction, vertex v can slide along the other two edges.

(b) The other edges of v do not have the same orientation, but lie on different sides of the line through e. When moving e in either direction, vertex v can slide along the edge on that side, but a copy of v must be introduced on its original position to preserve area and maintain correct topology.

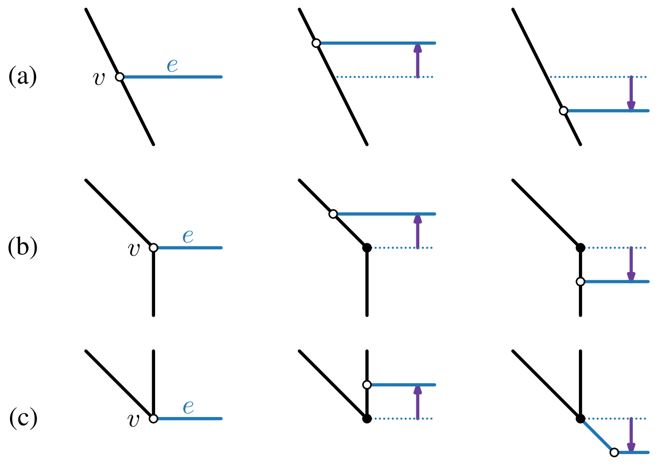


Figure 7.5 Three cases for degree-3 vertex v (white) when moving edge e. Copied vertices are indicated in black. From left to right: initial situation; moving e upward; moving e downward.

(c) The other edges of v do not have the same orientation, but lie on the same side of the line through e. When moving e in the direction of the other edges (upward in Figure 7.5(c)), vertex v can slide along the edge that makes the smallest angle to e, but a copy of v must be introduced on its original position to preserve area and maintain correct topology. When moving e in the other direction (downward in Figure 7.5(c)), vertex v can slide along the extension of either edge. We use the edge that has the largest angle to e, unless this edge has the same orientation as e (an angle of π). A copy of v must be introduced on its original position and this vertex has degree 3.

When combining edge-moves that introduce a new vertex, it is important to ensure that the complexity of the subdivision is still reduced. Combinations of edge-moves that do not decrease the complexity are not permitted. Computing the total complexity reduction of a single contraction or edge-move is straightforward. Finally, to ensure correct topology, it is important that a junction is never removed or moved on top of another junction.

7.2.2 Simplification algorithm

In the previous section we introduced edge-moves for polygons and subdivisions. By performing these operations in proper configuration pairs, the complexity of the subdivision is reduced while preserving area and topology. Here we describe how to use these edge-moves to build an efficient simplification algorithm.

Algorithm 7.1 provides an overview of our simplification algorithm. It iteratively finds two complementary feasible configurations that incur minimal change. This minimal configuration pair is executed to reduce the complexity of the subdivision. This process is repeated until either the desired output complexity is reached or no minimal configuration pair exists. It makes use of "blocking numbers" to keep track of the feasibility of configurations. Below, we provide the details for the two crucial steps: how to compute a minimal configuration pair (Line 3) and the steps involved in performing the contraction (Line 7). The latter must ensure that the blocking numbers remain correct.

Choosing edge-moves. We need to choose a proper configuration pair to perform a contraction and an edge-move simultaneously. We search for a pair that incurs the smallest

Algorithm 7.1 SIMPLIFY(S, k)

Require: a subdivision S and integer k; every face-face boundary in S is known.
Ensure: S has no feasible complementary pair or S has at most k edges.

```
1: Initialize blocking numbers for each edge
2: while $|S| > k$ do
3:     Determine minimal configuration pair $(X_1, X_2)$ over all boundaries in $S$
4:     if no minimal configuration pair exists then
5:         return $S$
6:     else
7:         Contract $(X_1, X_2)$ to reduce $|S|$
8: return $S$
```

visual change; we refer to this pair as the *minimal configuration pair*. This minimality can be formalized in various ways. For our algorithm, we use the area of the contraction region of the smallest of the two involved configurations. This corresponds to half the symmetric difference of a face before and after the operation is performed. We found that minimizing the symmetric difference does not always yield satisfying results (see Section 3.2). However, the problem of severely changing the shape does not occur here, since edge-moves make only local changes.

There may be more than a single border between two adjacent faces; edge-moves in one may be combined with edge-moves in the other. To avoid the need to explicitly search for other borders, we assume that the subdivision is partitioned into *face-face boundaries*. That is, for every pair of adjacent faces, all borders that separate them are stored together. In a preprocessing step, these boundaries can be computed in $O(n^2)$ time and $O(n)$ space, where n is the complexity of S. Note that there are at most a linear number of such boundaries since S is a subdivision (a plane graph). The sum of the number of edges over all boundaries is n.

To find the minimal configuration pair, we proceed as follows for each face-face boundary. First, we find the six smallest positive and the six smallest negative feasible configurations on this boundary in $O(b)$ time, where b is the number of edges of the boundary. Here, “smallest” refers to the minimality measure: the area of the contraction region. Since an edge-move can conflict with at most five other edge-moves, at least one of these must be part of a proper configuration pair if the boundary admits any feasible pair. As we need to find only the minimal pair, we compute which of these twelve candidate configurations admits the smallest contraction that is part of a proper configuration pair. For each, we can simply traverse the border in $O(b)$ time to search for a nonconflicting complementary feasible configuration. If there are multiple complementary moves, we use the one that is nearest, measured in the number of edges along the boundary. There are only $O(1)$ candidates for one boundary. Hence, computing the minimal pair of a single boundary takes $O(b)$ time. The minimal configuration pair is then simply the minimal pair over all boundaries; it is computed in $O(n)$ time.

If none of the boundaries has a proper configuration pair, no minimal configuration pair exists. In this case, the algorithm terminates before reaching the desired complexity k. By Theorem 7.1 (proven in Section 7.2.3), we know that this does not occur for simple polygons (unless k is less than twice the number of orientations in the input).

In the analysis above, we assumed that we can determine in constant time whether a configuration is feasible. To this end, each edge e stores two *blocking numbers*. The positive blocking number indicates the number of edges that (partially) reside in the positive contraction region of configuration X with inner edge e; the edges of X are excluded in this number. Analogously, the negative blocking number indicates the number of edges that overlap the negative contraction region. In the remainder, we refer to the negative and positive blocking numbers collectively as blocking numbers. To initialize these blocking numbers (Line 1), we can simply iterate over all pairs of edges and increment the values accordingly. Hence, this takes $O(n^2)$ time in total. However, we need to update the blocking numbers when contracting the minimal configuration pair. This is discussed in the upcoming paragraph.

Performing a contraction. Once the algorithm has computed a minimal configuration pair (X_1, X_2), it performs the contraction to reduce the complexity of S. A contraction is performed in a sequence of steps to ensure that the blocking numbers have the correct values afterwards.

In the first step, we discard the contribution that the edges of X_1 and X_2 made to the blocking numbers. To this end, we iterate over all edges in S. For each edge, we decrement the positive (or negative) blocking number by one for each edge of X_1 or X_2 that overlaps the positive (or negative) contraction region. Since the number of edges of X_1 and X_2 is 6, we can perform this entire step in $O(n)$ time.

In the second step, we perform the edge-moves that constitute the contraction of (X_1, X_2). The subdivision S is updated accordingly.

In the third step, we add the contribution to the blocking numbers for the edges that changed during the contraction (i.e., the remaining edges of X_1 and X_2). We repeat the same linear-time process as before, but now increment the numbers instead.

As a result of the contraction, the contraction region changes for a number of configurations. This implies that the blocking numbers may be incorrect for the inner edges of those configurations. Hence, in the fourth and last step, we ensure that these blocking numbers are correct as well. The contraction region of a configuration changes only if one or more of its edges were part of X_1 or X_2. As we do not allow edge-moves for junctions of degree 4 or higher, only a small constant number of configurations are affected. For each of the affected edges, we simply reinitialize the blocking number by iterating over all other edges of S. This takes $O(n)$ time per edge and thus also $O(n)$ time in total.

Analysis. To initialize the algorithm, we compute the boundaries of the subdivision and set the blocking numbers for the edges. Both steps take $O(n^2)$ time. As described above, each simplification step can be performed in $O(n)$ time using these numbers. Each contraction reduces the complexity of $|S|$ by at least one: at most $O(n - k) = O(n)$ steps are performed. Thus, Algorithm 7.1 computes a simplification in $O(n^2)$ time.

Increasing flexibility. It is relatively straightforward to modify the algorithm such that infeasible configurations can be used to compensate for area change. That is, we may allow edge-moves on configurations that have a blocking point p, if we ensure that this edge-move does not move the inner edge to or beyond p. We need to quickly decide whether a configuration can compensate for a change in area. To this end, the algorithm should not maintain the blocking number but the actual blocking point of a configuration. We can then compute in $O(1)$ time the maximal area change that the configuration can compensate for. A simple implementation leads to a cubic-time algorithm. However, for territorial outlines, we expect edge-moves to be obstructed by relatively few edges. Hence, the actual execution time is expected to not differ much from the original algorithm. This modification is especially relevant for subdivisions as it gives more flexibility. We apply this modified algorithm to obtain the results presented in Section 7.4.

Alternative choices. We use the area of the contraction region to define the minimal configuration pair. As an alternative, we may use the Fréchet distance between the configuration before and after the edge-move. However, some operations may cause a relatively large Fréchet distance but only a small change in area. The visual change seems to be more strongly correlated to the area than the distance. This is similar to the use of area (rather than distance) to decide on simplification steps in the Visvalingam-Whyatt algorithm [172].

To compute the minimal configuration pair, we select the nearest complementary edge-move, measured in the number of edges along the boundary. As an alternative, we could opt to use one of the smallest complementary moves which have already been found: as there are six, at least one of these must not conflict. Though it does not improve the asymptotic running time, it eliminates the need for extra traversals. However, this change reduces the "locality" of the minimal configuration pair. To perform edge-moves that make small local changes, we opted to use the nearest complementary move instead.

We do not allow edge-moves that move junctions of degree 4 or higher as these either alter topology or area. By combining more than two edge-moves, we can move area along a "cycle" of faces, allowing edge-moves involving these junctions. This does incur a higher computational cost. However, we consider such an extension to be unnecessary: in our experiments, the desired complexity could be reached without this addition.

7.2.3 Existence of edge-moves for polygons

The algorithm presented in the previous section needs to find a proper configuration pair, i.e., a nonconflicting pair of complementary edge-moves. One of these edge-moves needs to be a contraction to reduce the complexity and ensure progress. In a subdivision, such a pair may not exist and the desired complexity of the output may be unattainable. In this section we prove that this does not occur for a simple polygon (i.e., a polygon that has no self-intersections), unless it is convex. This is formulated in the theorem below.

Theorem 7.1. *Every nonconvex simple polygon has a nonconflicting pair of complementary edge-moves, one of which is a contraction.*

Definitions and notation. In order to prove the theorem, we first introduce some additional terminology and notation. We define a *chain* S as a sequence of at least three consecutive edges of simple polygon P. Its edges are denoted by $\langle s_1, \ldots, s_m \rangle$ where m is the number of edges in S. Its vertices are denoted by $u_0, \ldots, u_m$ and edge s_i is directed from u_{i-1} to u_i. The edges s_1 and s_m are the *outer edges* of S; the other edges are its *inner edges*. Likewise, u_0 and u_m are outer vertices and the other vertices are inner vertices. By $\alpha(S)$, we denote the sum of the exterior angles of the inner vertices of S.

A *lid* is an open line segment between a point on s_1 (strictly before u_1) and a point on s_m (strictly after u_{m-1}) and is fully contained in the interior of P. If S has any lid, it is a *closable chain*. If the open line segment (u_0, u_m) is a lid, S is a *proper chain*. For a closable chain S and a lid l, we denote by $R_l(S)$ the region enclosed by S and l. For a proper chain, $R(S)$ denotes this region using the lid (u_0, u_m) implicitly. Due to the lid, we know that for every closable chain, any point on ∂P that is inside $R_l(S)$ must be part of S. Some examples are given in Figure 7.6. A configuration is a (not necessarily closable) chain of length three.

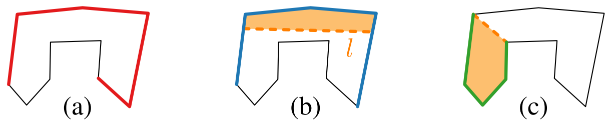


Figure 7.6 (a) Unclosable chain. (b) Closable chain with a lid l and $R_l(S)$ given in orange. (c) Proper chain with $R(S)$ indicated in orange.

Proving the theorem. We are now ready to give the proof of Theorem 7.1. We do this via a sequence of six lemmas. First, we prove two lemmas about closable chains.

Lemma 7.2. *For any closable chain S, $\alpha(S) > 0$ holds.*

Proof. Since S is a closable chain, it must have some lid l. Let u and v denote the endpoints of l. Endpoints u and v and the inner vertices of S define a simple polygon P' that corresponds to $R_l(S)$. We know that the sum of the exterior angles for any simple polygon is exactly 2π. Hence, we find that $\alpha(P') = 2\pi = \alpha(S)+\alpha(u)+\alpha(v)$. Moreover, we know that the exterior angle of any vertex is strictly less than π. Hence, we may conclude that $\alpha(S) = 2\pi - \alpha(u) - \alpha(v) > 0$. □

Lemma 7.3. *Let S be a closable chain without a convex inner edge such that the first inner vertex is reflex. Then S contains a configuration X such that the first inner vertex is reflex and $\alpha(X) > 0$.*

Proof. Since S is closable, we know that $\alpha(S) > 0$ (Lemma 7.2). By assumption we know that S contains no two consecutive convex inner vertices and that its first inner vertex is reflex. Hence, we may characterize the inner vertices of S by k sequences of one or more reflex vertices followed by a single convex vertex. Let ς_i denote the i^{th} sequence. Slightly abusing notation, let $\alpha(\varsigma_i)$ denote the sum of exterior angles of *all* the vertices of sequence ς_i. Now we know that $\sum_{i=1}^{k} \alpha(\varsigma_i) = \alpha(S) > 0$. In particular, this implies that $\alpha(\varsigma_i) > 0$ holds for at least one sequence. Since reflex vertices have a negative exterior angle, this implies that the exterior angle of the single convex vertex is bigger than the reflex vertex preceding it. Hence, at the end of a sequence ς_i with $\alpha(\varsigma_i) > 0$, we find configuration X such that the first inner vertex is reflex and $\alpha(X) > 0$. □

We now prove the existence of feasible negative configurations in chains. We provide two lemmas assuming different properties of the chain (Lemma 7.4 and Lemma 7.5).

Lemma 7.4. *Let $S = \langle s_1, \ldots, s_m \rangle$ be a closable chain without a convex inner edge such that the first inner vertex is reflex. Then S has a feasible negative configuration X such that the first inner vertex is reflex, $\alpha(X) > 0$ and $R^-(X) \subseteq R_l(S)$ for any lid l of S.*

Proof. We prove this lemma by induction. For the base case, assume that $m = 3$. Configuration $X' = \langle s_1, s_2, s_3 \rangle$ has a first inner vertex that is reflex. Any lid l of S must be on one side of the line through s_1, whereas this track enforces a triangular contraction region on the other side. Hence, $R^-(X') \subseteq R_l(S)$ holds and X' is a feasible negative configuration. Figure 7.7 shows closable and, for comparison, unclosable configurations.

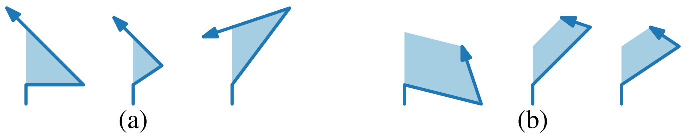


Figure 7.7 Configurations with $\alpha > 0$; the first inner vertex is reflex. (a) Closable chains. (b) Unclosable chains.

For the inductive step, we use as induction hypothesis that this lemma holds for any closable chain with less than m edges. Let $X' = \langle s_{i-1}, s_i, s_{i+1} \rangle$ be the first configuration such that the first inner vertex is reflex and $\alpha(X') > 0$. This implies that the second inner vertex is convex. From Lemma 7.3, we conclude that S must contain such a configuration. If X' is feasible, we are done. If X' is not feasible, let p denote the corresponding blocking point. We prove that p is part of S and comes after s_{i+1}.

Refer to Figure 7.8(a–b). Chain S is closable and hence has some lid l that does not intersect any part of S (dotted in the figure). Suppose l intersects $R^-(X')$, the contraction region of X'. Since the contraction region is bounded by s_i and s_{i+1} of S on two sides, we know that the lid must intersect the track of s_{i-1} in some point q. Note that this is actually possible, as illustrated in Figure 7.8(a). However, this implies that some other part of S also intersects this track and it does so before q (indicated in dark red in the figure). We prove this by contradiction. Assume that there is no intersection between S and the track of s_{i-1} that lies between s_{i-1} and q. This implies that the endpoint of l on s_1, the chain $\langle s_2, \ldots, s_{i-2} \rangle$ and q define a simple polygon (indicated in orange in Figure 7.8(b)). Since the sum of exterior angles of a simple polygon is 2π and u and q have an exterior angle strictly smaller than π, the sum of exterior angles in the inner vertices $\langle s_1, \ldots, s_{i-1} \rangle$ is positive. However, this contradicts that X' is the first configuration such that the first inner vertex is reflex and $\alpha(X') > 0$. Therefore, there must be an intersection of S that

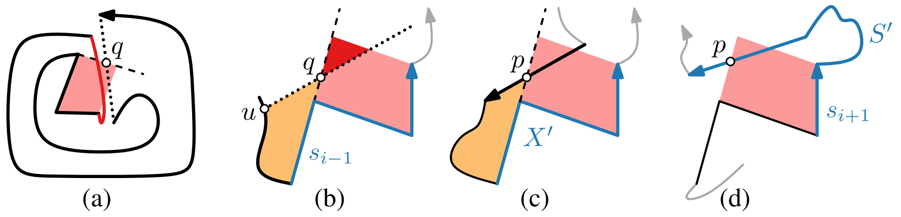


Figure 7.8 Negative configuration X' with $R^-(X')$ given in light red. Lid l is dotted; the track of s_{i-1} is dashed. (a) Intersection between l and $R^-(X')$ is possible. (b) If the orange polygon is simple, we derive a contradiction on the definition of X'. If q exists, the dark red part of $R^-(X')$ cannot contain the blocking point. (c) Blocking point p must come after X'. Otherwise, the orange polygon contradicts the definition of X'. (d) Chain S' defined by p.

is closer to the inner edge of X' than intersection q. In particular, this also means that anything outside of $R_l(S)$ that lies in $R^-(X')$ cannot be the blocking point; this area is indicated in dark red in Figure 7.8(b).

Now that we have concluded that blocking point p of X' is part of S, we must argue that it comes after s_{i+1}. Refer to Figure 7.8(c). If blocking point p occurs along an edge prior to X', then we may again construct a simple polygon using p and the edges after p up to s_{i-1} (indicated in orange in the figure). Since p and the first vertex along S after p have an exterior angle smaller than π, we again conclude that the sum of exterior angles leading up to s_{i-1} is positive, contradicting the definition of X'. We conclude that blocking point p must come after s_{i+1} on S.

Since p is a point on S after s_{i+1}, we consider the closable chain $S' = \langle s_{i+1}, \ldots, p \rangle$, that is, until the edge containing p or incident to p if p is a vertex). Chain S' is illustrated in Figure 7.8(d). Since S does not contain convex edges, we know that S' does not have convex edges and that its first inner vertex is reflex. Since S' also has less edges than S, we know from the induction hypothesis that S' has a feasible negative configuration. □

Lemma 7.5. *Let $S = \langle s_1, \ldots s_m \rangle$ denote a proper chain with a convex inner edge. Then S has a feasible negative configuration $X = \langle s_{i-1}, s_i, s_{i+1} \rangle$ with $R^-(X) \subseteq R(S)$ and $\alpha(X) > 0$. Also, s_i is convex or starts at a reflex vertex or $s_i \neq s_{m-1}$.*

Proof. We prove this lemma by induction on m, the number of edges in S. For the base case, assume $m = 3$. Since S is a proper chain and s_2 is a convex edge, $\langle s_1, s_2, s_3 \rangle$ is a configuration satisfying the conditions for X.

For the inductive step, we use as induction hypothesis that this lemma holds for any suitable chain with less than m edges. Let s_j be a convex inner edge. Hence, $X' = \langle s_{j-1}, s_j, s_{j+1} \rangle$ is a negative configuration. If X' is feasible, we are done since a convex edge implies $\alpha(X') > 0$. If X' is not feasible, then there is a blocking point that is in fact a vertex of S. This follows from the fact that the inner edge of X' is convex: this implies $R^-(X') \subseteq R(S)$ and that $R^-(X')$ is bounded on three sides by the edges of X'. Let u_k denote the blocking vertex and assume that $k > j + 1$. Consider the proper chain $S' = \langle s_j, \ldots, s_{k-1} \rangle$ (see Figure 7.9(a)). If S' has a convex inner edge, it must have a feasible negative configuration by induction. However, if it does not, $S'' = \langle s_{j+1}, \ldots, s_{k-1} \rangle$ is a closable chain such that its first inner vertex is reflex and it has no convex inner edge. Note that the direction of S'' should be reversed when $k < j - 1$. By Lemma 7.4, S'' has

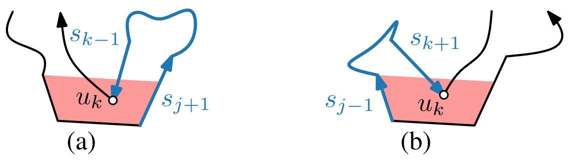


Figure 7.9 (a) Chain S'' defined by blocking vertex u_k and the last outer edge of X'. (b) Reversed chain S'' defined by blocking vertex u_k and the first outer edge of X'.

a feasible negative configuration X''. Moreover, the first inner vertex of X'' is reflex in the direction of S''. If $k < j - 1$, we use a reversed chain $S'' = \langle s_{j-1}, \ldots, s_{k+1} \rangle$ (see Figure 7.9(b)). We may follow the same line of argument. The first inner vertex *in the direction of* S'' of the found configuration may be reflex. However, we know that s_m is not part of S'' and thus s_{m-1} cannot be the inner edge of X''. Hence, the third alternative condition for the desired configuration holds. □

Now that we know that feasible negative configurations exist, we need a feasible positive configuration to make a proper configuration pair. The existence of a feasible positive configuration is proven in Lemma 7.6. However, not any positive configuration suffices: it must not conflict with the negative one. We prove in Lemma 7.7 that such a proper configuration pair indeed exists.

Lemma 7.6. *Every simple nonconvex polygon P has a feasible positive configuration X with $\alpha(X) < 0$ or all positive configurations are feasible.*

Proof. If P has a reflex edge, then let X be a configuration with a reflex inner edge. If X is feasible, we are done as $\alpha(X) < 0$. If it is not, we can define a chain $\overline{S}$ that is inverted: the interior of $\overline{S}$ is in fact the exterior of the polygon. This is done as follows.

We cut polygon P into two chains S_1 and S_2 using the blocking point and one inner vertex of X. For both, the line segment l between the blocking point and the inner vertex can be considered a proper lid as it does not cross any part P. Hence, the enclosed regions $R(S_1)$ and $R(S_2)$ are well defined. One of these enclosed regions is fully outside polygon P (i.e., their interiors are disjoint); the other encloses both P and the region enclosed by the other chain. The former is the inverted chain that we use. Figure 7.10(a–b) indicates the inverted chain for both inner vertices in blue; the other chain is indicated in black.

We use both inner vertices of X to obtain two inverted chains. Both of these are in fact proper chains and one is a single edge longer than the other. Let $\overline{S_a}$ denote the longer and $\overline{S_b}$ the shorter of the chains. If $\overline{S_a}$ has a convex inner edge (i.e., a reflex edge in the polygon), then Lemma 7.5 states that a feasible negative configuration X^- with $\alpha(X^-) > 0$ exists in $\overline{S_a}$. Since $\overline{S_a}$ is inverted, this corresponds to a feasible positive configuration X^+ in P with $\alpha(X^+) < 0$. If $\overline{S_a}$ does not have a convex inner edge, then $\overline{S_b}$ must have a first inner vertex that is reflex (with respect to $R(\overline{S_b})$). Hence, we can

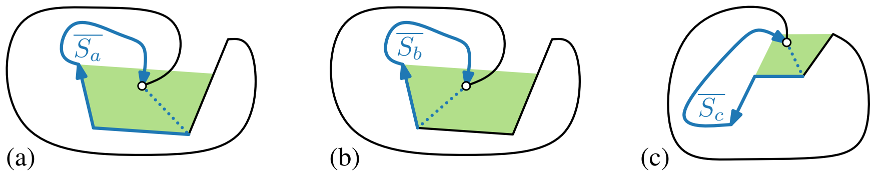


Figure 7.10 Feasible positive configurations are found using inverted chains. The dotted line is the lid used to define an inverted proper chain. (a–b) The inverted chains $\overline{S_a}$ and $\overline{S_b}$ when P has a reflex edge. (c) The inverted chain $\overline{S_c}$ when P has no reflex edge.

now apply Lemma 7.4 to $\overline{S_b}$ in a similar way to conclude that there is a feasible positive configuration X^+ in P with $\alpha(X^+) < 0$.

If P has no reflex edge, then let X be an infeasible positive configuration. If none exists, all positive configurations are feasible. Otherwise we can define an inverted chain using X. Let $\overline{S_c}$ be the inverted proper chain obtained by using the lid defined by the blocking point of X and its reflex vertex (see Figure 7.10(c)). Note that, $\overline{S_c}$ must start with a reflex vertex (a convex vertex in P), otherwise, P would have a reflex edge. Thus, by Lemma 7.4, P has a feasible positive configuration X^+ with $\alpha(X^+) < 0$. □

Lemma 7.7. *Every simple nonconvex polygon P has a proper configuration pair.*

Proof. From Lemma 7.6, we conclude that polygon P has a feasible positive configuration $X^+ = \langle e_{i-1}, e_i, e_{i+1} \rangle$. Assume without loss of generality that the second inner vertex of X^+, p_{i+1}, is reflex. Let p_j denote the first convex vertex after p_{i+1}. Configuration $X^- = \langle e_{j-1}, e_j, e_{j+1} \rangle$, of which p_j is the first inner vertex, is negative. We distinguish two cases.

Assume that X^- is feasible. If no edge is shared or if edge $e_{i+1} = e_{j-1}$ is shared (having a convex and a reflex vertex), we have a proper configuration pair as illustrated in Figure 7.11(a–b). If $e_{i-1} = e_{j+1}$ is shared but the other outer edge is not, then p_j, $p_{j+1} = p_{i-1}$, and p_i are the only convex vertices in P and there is at least one edge in between e_i and e_{j-1}. This edge is the inner edge of a feasible positive configuration, one that does not conflict with X^-; this is illustrated in Figure 7.11(c).

Now assume that X^- is not feasible. By construction, the blocking point cannot be in between v_i and v_{j+1}. If X^- is blocked by a vertex v_h, then, depending on the convexity of v_{j+1} and v_{j+2}, either Lemma 7.4 or Lemma 7.5 shows that there is a (nonconflicting) feasible negative configuration. If X^- is blocked by an edge e_h, p_{j+1} must be reflex. We distinguish two cases on the closable chain $S = \langle e_j, \ldots, e_h \rangle$.

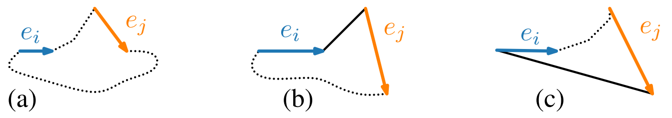


Figure 7.11 Three cases if X^- is feasible. (a) No edge is shared. (b) Edge $e_{i+1} = e_{j-1}$ is shared. (c) Edge $e_{i-1} = e_{j+1}$ is shared.

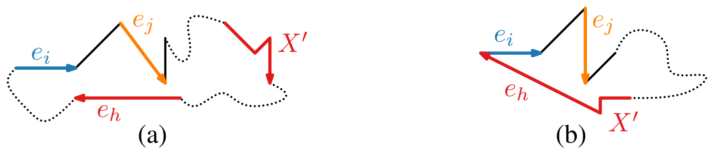


Figure 7.12 Two cases if X^- is not feasible and chain $S = \langle e_j, \ldots, e_h \rangle$ has no convex inner edge. (a) X' does not conflict with X^+. (b) X' conflicts with X^+.

If S does not have a convex inner edge, then we refer to Lemma 7.4 to find a feasible negative configuration X'. If X' does not conflict with X^+, we are done (Figure 7.12(a)). If X' conflicts with X^+, we know that $e_h = e_{i-1}$ holds and that e_{h-1} is the inner edge of X' (Figure 7.12(b)). Moreover, it holds that $\alpha(X^+) > 0$ and thus, by Lemma 7.6, we need to consider this case only when all positive configurations are feasible. Hence, the positive configuration $\langle e_i, e_{i+1}, e_{i+2} \rangle$ is feasible and it does not conflict with X'.

If S has a convex inner edge e_t, then let $X' = \langle e_{t-1}, e_t, e_{t+1} \rangle$ denote the corresponding negative configuration. If X' is feasible and not conflicting with X^+, we are done. If X' is feasible but conflicting with X^+, we can argue as above: $\langle e_i, e_{i+1}, e_{i+2} \rangle$ is a feasible positive configuration and it does not conflict with X'. If X' is not feasible, it must be blocked by some vertex p_b. Assume that the proper chain $S' = \langle e_t, e_{t+1}, \ldots, e_{b-1} \rangle$ does not contain edge e_i. If it does, we may argue analogously for the reversed chain $\langle e_b, \ldots, e_{t-1}, e_t \rangle$. Depending on the convexity of vertex p_{t+2}, Lemma 7.4 or Lemma 7.5 shows that there is a feasible negative configuration X'' in S'. Either the first inner vertex of X'' along S' is reflex, the inner edge of X'' is convex or the inner edge of X'' is not the before-last edge of S'. To derive a contradiction, we assume that X'' conflicts with X^+. This is possible only if $p_b = p_i$ (or $p_b = p_{i+1}$ in the analogous case), otherwise S' and X^+ have no edges in common. Since p_b is the blocking point of X', it is a reflex vertex. As X'' conflicts with X^+, it is the last configuration in S', i.e., $X'' = \langle e_{i-3}, e_{i-2}, e_{i-1} \rangle$. Therefore, p_{i-1} must be reflex: otherwise it would not be a conflict as the shared edge then has a reflex and a convex vertex. Since X'' is a negative configuration, p_{i-2} is therefore convex. However, now we know that the first vertex of X'' along S' is convex, the inner edge of X'' is not convex, and the inner edge of X'' is the before-last edge of S'. This contradicts the properties for X'' that we obtained from applying Lemma 7.4 or Lemma 7.5 on S'. Hence, our assumption is invalid and X'' cannot conflict with X^+. $\square$

A proper configuration pair is a nonconflicting complementary pair of configurations that admit a contraction. Thus, Theorem 7.1 follows directly from Lemma 7.7.

7.3 Schematization

In the previous section we described a topologically safe and area-preserving simplification algorithm. The edge-moves used by this algorithm do not introduce new orientations either. Hence, the orientation set of the result is equal to the orientation set of the input (or a subset thereof). Due to these properties, we obtain an area-preserving $\mathcal{C}$-oriented schematization algorithm by applying an *angular-restriction algorithm* beforehand. This algorithm turns a subdivision into an area-equivalent $\mathcal{C}$-oriented subdivision for any orientation set $\mathcal{C}$. The quality of the result improves if the input subdivision does not contain "long" edges. We use a small constant fraction $\lambda = 0.05$ of the diameter of the input as an upper bound for the edge length and split all edges which are longer.

An overview of the algorithm is given in Algorithm 7.2. It consists of three high-level steps, each of which we describe in detail in this section. First, the algorithm assigns directions and, based on this assignment, classifies the vertices and edges (Line 1–2, see

Algorithm 7.2 ANGULARRESTRICTION($S, \mathcal{C}$)

Require: S is a subdivision; $\mathcal{C}$ is an orientation set.
Ensure: A $\mathcal{C}$-oriented subdivision R that is area-equivalent and topologically equivalent to subdivision S.

```
1:  Assign directions to each incident edge for each vertex in $S$
2:  Classify vertices and edges in $S$
3:  Determine staircase region for each edge in $S$
4:  for all edges $e$ in $S$ do
5:      Determine possible interfering edges using staircase regions
6:      Compute required number of steps for $e$
7:  Initialize empty subdivision $R$
8:  for all edges $e$ in $S$ do
9:      Construct staircase for $e$ with the computed number of steps
10:     Add the staircase to $R$
11: return $R$
```

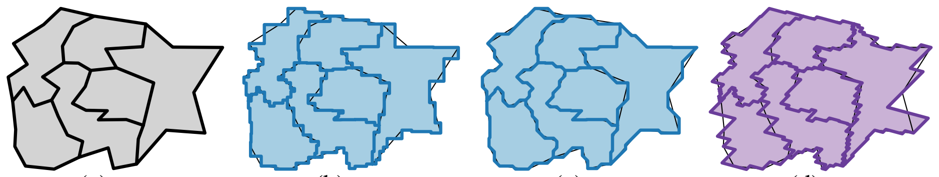


Figure 7.13 Subdivision (a) and area-equivalent angular restrictions with $\lambda = 1$: (b) Rectilinear, $\mathcal{C}_2$; (c) Octilinear, $\mathcal{C}_4$; (d) Irregular, $\{\frac{\pi}{4}, \frac{11\pi}{12}\}$.

Section 7.3.1). Using this classification, it eventually constructs a $\mathcal{C}$-oriented curve, a "staircase", for each edge (Line 7–10, see Section 7.3.2). To this end, the required number of "steps" in the staircase is computed such that the result is crossing-free (Line 3–6, see Section 7.3.3). Though the construction of staircases occurs afterwards, this is described before the details of computing the number of steps: this depends heavily on the geometry of the various staircases. Possible results of this algorithm are illustrated in Figure 7.13.

7.3.1 Classification of vertices and edges

The first step in our algorithm is to assign directions and classify vertices and edges in the subdivision (Algorithm 7.2, Line 1–2). We call an edge *aligned* if it already adheres to an orientation in $\mathcal{C}$, and *unaligned* otherwise. The orientations of $\mathcal{C}$ partition the space around each vertex into $|\mathcal{C}|$ *sectors*. Each sector has two *associated directions* (see Figure 7.14): these are the directions along the orientations that bound the sector. At both of its endpoints, an edge has one or two associated directions. If the edge is aligned, it has one associated direction, being the direction of the edge. If it is unaligned, the two

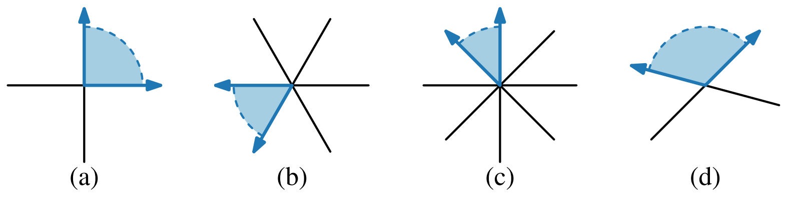


Figure 7.14 Associated directions of sectors. (a) Rectilinear, $\mathcal{C}_2$. (b) Hexilinear, $\mathcal{C}_3$. (b) Octilinear, $\mathcal{C}_4$. (d) Irregular, $\{\frac{\pi}{4}, \frac{11\pi}{12}\}$.

associated directions correspond to the associated directions of the sector in which it lies. We call a vertex *insignificant*, if the associated directions of its incident edges are disjoint. This means that we can freely choose an associated direction for each incident edge without limiting the choice for other edges. A vertex is called *significant* otherwise, indicating that we are not free to choose directions. We assume that every edge has at most one significant vertex. This property is ensured by splitting an edge with two significant vertices, as the new vertex is insignificant.

For each edge, we now assign a $\mathcal{C}$-oriented direction at its significant vertex. If an edge has two insignificant vertices, we randomly pick one to be treated as the significant vertex. Since we assume that each vertex has degree at most 4 (see Section 7.1), we can always find an assignment that ensures the following three properties:

- no two edges are assigned the same direction at a common vertex;
- the cyclic order of edges is preserved;
- the total angular deviation is minimized.

The angular deviation of an edge at its significant vertex is always smaller than π. Based on this assignment, we classify the edges as follows (refer to Figure 7.15 for examples).

- *Aligned basic edge*: the edge is aligned and its assigned direction is its (only) associated direction. See Figure 7.15(b–c,i–j,n).
- *Unaligned basic edge*: the edge is unaligned, its assigned direction is one of its associated directions, and its significant vertex does not have another incident unaligned edge in the same sector that is assigned one of its associated directions. See Figure 7.15(a–c,g–n).
- *Evading edge*: the edge is unaligned, its assigned direction is one of its associated directions, and its significant vertex has another incident unaligned edge in the same sector that is assigned one of its associated directions. Evading edges always occur in pairs and aligned evading edges cannot occur. See Figure 7.15(d–g,l–n).
- *Aligned deviating edge*: the edge is aligned and its assigned direction is different from its associated direction. Note that its significant vertex must be a junction. See Figure 7.15(f,m).
- *Unaligned deviating edge*: the edge is unaligned and its assigned direction is not one of its associated directions. Note that its significant vertex must be a junction. See Figure 7.15(e–g,m).

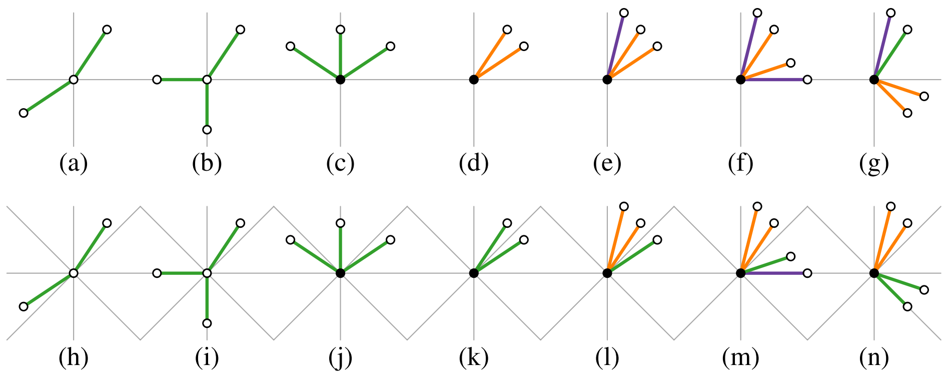


Figure 7.15 Classification of edges: basic edges (green); evading edges (orange); deviating edges (purple). Significant vertices are marked with black dots. (a–g) Rectilinear, C_2. (h–n) Octilinear, C_4.

7.3.2 Converting edges to staircases

To construct a $\mathcal{C}$-oriented subdivision, we create a *staircase* for each edge (Algorithm 7.2, Line 7–10). A staircase is a sequence of $\mathcal{C}$-oriented edges that starts and ends at the vertices of the edge. A *step* in a staircase is a combination of two $\mathcal{C}$-oriented edges such that the step starts and ends on the edge that the staircase must represent. By increasing the number of steps in the staircase, intersections can be avoided. We describe in Section 7.3.3 how to obtain the correct number of steps. Once we know the appropriate number of steps, the edge is converted in isolation. Let $e = (v, w)$ denote the edge we wish to convert and s_e the number of steps it must use. Without loss of generality, we assume that v is the significant vertex. The construction of a staircase depends on the classification of edge e.

If e is an aligned basic edge, we do not change it as it is already $\mathcal{C}$-oriented. If e is an unaligned basic edge, we treat it as follows. Let d_1 denote its assigned direction and d_2 its other associated direction. Each step uses one edge parallel to d_1, called the *assigned edge* of the step, and one parallel to d_2, called the *associated edge*. A step can start either with the former (an *assigned* step) or with the latter (an *associated* step). Moreover, every step should span exactly a fraction of $1/s_e$ of the length of e. The length of the assigned and associated edge is the same in all steps. Assuming that d_1 and d_2 are normalized vectors, these can be found by solving $(w - v)/s_e = l_1 \cdot d_1 + l_2 \cdot d_2$ for l_1 and l_2. Every step adds area to one incident face of e and removes it from the other. Since an assigned and an associated step counterbalance the area change, we combine $s_e/2$ assigned steps with $s_e/2$ associated steps.[1] By choosing these steps, the procedure guarantees an area-equivalent result. Starting at v, we alternate the steps starting with an assigned step to adhere to the assigned direction. The number of edges used in the staircase is $s_e + 1$. Figure 7.16(a–b) shows examples for the staircase of a basic edge.

[1]We ensure that s_e is even for all edges in Section 7.3.3.

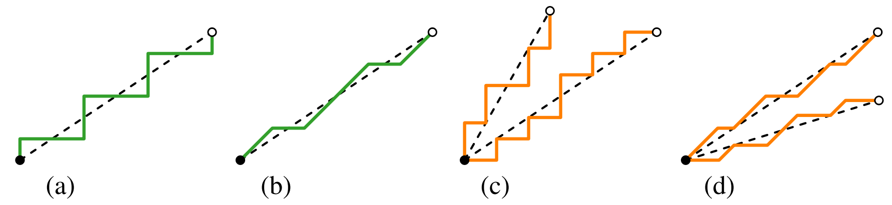


Figure 7.16 Staircase examples for basic and evading edges, indicated in green and orange respectively. Black vertices are significant. (a) Rectilinear basic edge. (b) Octilinear basic edge. (c) Two rectilinear evading edges. (d) Two octilinear evading edges.

For an evading edge e, we know that there is another evading edge in the same sector of its shared significant vertex. To avoid intersections, we apply so-called *evasive behavior*. We build a staircase similar to the staircase for an unaligned basic edge. However, instead of alternating the steps, we now first place all assigned steps (starting at v) followed by all associated steps. This results in a staircase of which the first half lies completely on the far side of the evading edge with respect to the other evading edge. This guarantees that there are no intersections near the significant vertex. The number of edges in the staircase is $2s_e - 1$. Figure 7.16(c–d) shows examples of this evasive behavior.

An aligned deviating edge is not converted with steps and uses a fixed number of edges. Instead of a number of steps s_e, we derive a value δ_e for such an edge; in Section 7.3.3 we show how to obtain a sufficiently small value to prevent intersections from occurring. In addition, it uses a small constant ε to ensure that the edge $e = (u, v)$ adheres to its direction at the insignificant vertex w. We use $\varepsilon = 0.1$. Let d_1 denote the assigned direction and d_2 the direction of the edge (i.e., $d_2 = w - v$). We start at vertex v. The first edge is directed along d_1 and has length δ_e. The second edge has length $\|e\|(1-\varepsilon)/2$ and is directed along d_2. The third edge has length $2\delta_e$ and is directed in the opposite direction of the first edge. The fourth edge is analogous to the second. The fifth edge is analogous to the first. The sixth edge is directed along d_2 and has length $\varepsilon\|e\|$. The number of edges used in the staircase is six, independent of δ_e. Figure 7.17(a–b) shows examples.

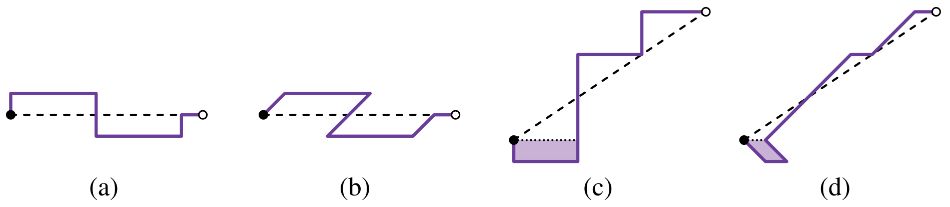


Figure 7.17 Examples of rectilinear and octilinear staircases. Black vertices are significant. (a–b) For an aligned deviating edge. (c–d) For an unaligned deviating edge. Appended region is shaded in purple.

For an unaligned deviating edge, we make a staircase that resembles that of an evading edge, but adapt it to use the correct assigned direction. We first create the staircase as if the edge was an evading edge. For the purpose of the evasive behavior, we use as an "assigned direction" the associated direction closest to its (actual) assigned direction. However, instead of using s_e steps, we use only $s_e - 1$ steps: $(s_e/2) - 1$ assigned steps followed by $s_e/2$ associated steps. This violates both the area-preservation constraint as well as the assigned direction. To correct for this, we append a region—the area of a single step—to the first edge by "dragging" the first edge of the intermediate staircase in the assigned direction. Note that s_e must be at least four. The number of edges used in the staircase is $2s_e - 1$. Figure 7.17(c–d) shows examples.

7.3.3 Computing the number of steps

Here we describe how to compute the number of steps, s_e, for an edge e such that the resulting angular restriction is free of intersections (Algorithm 7.2, Line 3–6). We ensure that the staircase of an edge is less than a distance $d_e/2$ away from the edge itself, where d_e is the minimal distance from e to another point in subdivision S. However, we need not and should not take all other edges into account. Neighboring edges of e have points that are arbitrarily close to e. This would cause an infinite number of steps. Moreover, we may exclude edges that cannot cause intersections regardless of the number of steps.

Interference. We first determine which edges may interfere with edge $e = (v, w)$, that is, which edges may have staircases that could intersect the staircase of e. Let e' denote some other edge of S. If e and e' do not share a vertex, we first make a rough estimate of whether the staircases could intersect. To this end, we define the *staircase region* of an edge. The staircase region is a bounding region of an edge that contains the staircase regardless of the number of steps.

For an aligned basic edge, the staircase region is simply the edge itself. For unaligned basic and evading edges, the staircase region is the area bounded by lines oriented according to the associated directions (both at v and w); this is illustrated in Figure 7.18(a).

For unaligned deviating edges, we use a similar staircase region as an evading edge, but it must accommodate for the appended area that is used to make the staircase adhere to the assigned direction (see Figure 7.18(b–c)). The appended area is largest for a minimal

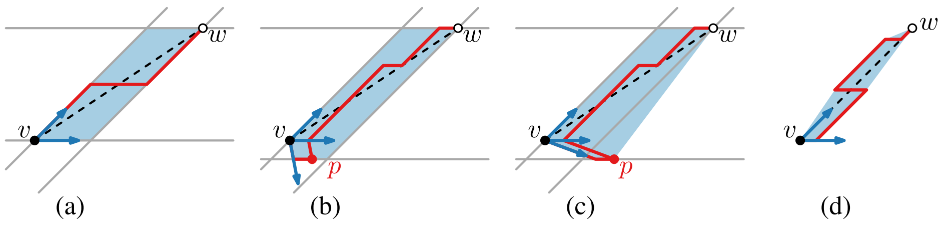


Figure 7.18 Examples of staircase regions. (a) Unaligned basic and evading edges. (b–c) Unaligned deviating edge. (d) Aligned deviating edge.

number of steps (4 steps); let point p denote the vertex of this maximal appended area that is furthest from v. The staircase region is the region containing e that is enclosed by the following lines: lines through w adhering to the associated directions of e; line through v adhering to associated direction with largest angle to assigned direction; line through p adhering to associated direction with smallest angle to assigned direction. In the above, we assumed that p lies in the defined region (see Figure 7.18(b)). However, if this is not the case, we can extend the staircase region by including the vertices of the appended region; this is illustrated in Figure 7.18(c).

Aligned deviating edges do not use steps, but a value δ_e instead. We use a value Δ_e as an upper bound for δ_e. To define this value, we use a constant fraction of the edge length: $\Delta_e = 0.1\|e\|$. We compute the staircase using $\delta_e = \Delta_e$ and use its convex hull as staircase region (see Figure 7.18(d)).

Any staircase with more steps (or $\delta_e < \Delta_e$) is contained in the staircase region. Hence, there is interference only if the staircase regions of e and e' intersect, assuming that these edges do not share a vertex. If e and e' share a vertex v, they interfere only if the edges reside in the same sector with respect to v. To this end, aligned deviating edges are considered to reside in both sectors. For these pairs, we determine interference purely based on the classification. Aligned basic edges cannot cause interference. Unaligned basic edges interfere with unaligned and aligned deviating edges: by definition an unaligned basic edge cannot be in the same sector as an evading edge. Evading edges interfere with other evading edges, unaligned and aligned deviating edges. Deviating edges, both aligned and unaligned, cannot interfere with one another: the assigned direction is not one of the associated directions and hence another edge must lie in between.

Edge distance. We define the distance d_e for an edge e as the minimum over all distances $d_{e,e'}$ for all edges e' that interfere with e. Distance $d_{e,e'}$ is computed as follows. If e and e' do not share a vertex, then $d_{e,e'}$ is simply the minimal distance between the edges. However, if e and e' do share a vertex, then we must again look at the classification. Depending on this classification, we ignore parts of the edges (measured from the shared vertex) in the distance computation. In these cases, the staircase construction guarantees that no intersections are introduced.

If e is an unaligned basic edge, then e' is either an aligned or unaligned deviating edge. If e' is aligned, then we ignore a fraction of $(1-\varepsilon)/2$ of e'. This fraction corresponds to the part of the staircase of e' that resides on the "far side" of e'. If e' is unaligned, then we ignore a fraction of e' equal to the length of the first step. That is, we ignore a fraction of $1/(s_{e'}-1)$. This is illustrated in Figure 7.19(a).

If e is an evading edge, then e' is either an evading or deviating edge. If e' is an evading edge, we ignore the first half of e (but not of e'). Due to the evasive behavior in the construction, we know that there are no intersections here. This is illustrated in Figure 7.19(b). If e' is a deviating edge, we treat it as if e was an unaligned basic edge.

If e is an aligned deviating edge, then e' is either an unaligned basic edge or an evading edge. Regardless of e', we ignore a fraction of $(1-\varepsilon)/2$ of e (see Figure 7.19(c)).

If e is an unaligned deviating edge, then e' is either an unaligned basic edge or an evading edge. A fraction of $(s_e/2-1)/(s_e-1)$ evades the other edge, where s is the

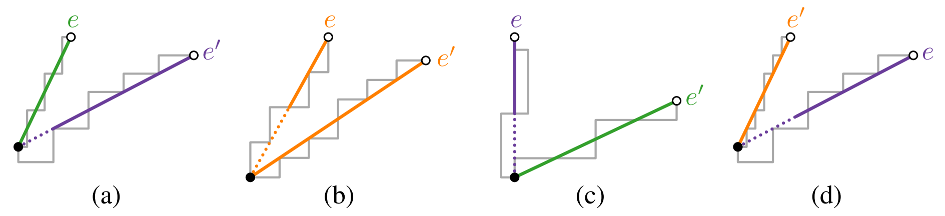


Figure 7.19 Examples of parts ignored in the computation of $d_{e,e'}$, as indicated by dotted lines. (a) Basic edge e and unaligned deviating edge e'. The staircase of e' is already known. (b) Two evading edges. (c) Aligned deviating edge e. (d) Unaligned deviating edge e.

computed number of steps for e. Since this fraction is increasing with s and there is a minimum of 4 steps, the minimal fraction is $1/3$. Regardless of the class of e', we ignore a fraction of $1/3$ of e. This is illustrated in Figure 7.19(d).

The edge distance depends on $s_{e'}$ if e is an unaligned basic or evading edge and e' is an unaligned deviating edge. Since $s_{e'}$ can be computed for e' without dependencies, this poses no problem. We first compute the distances and step numbers for all deviating edges, followed by the computation for the remaining edges.

Step number. The number of steps s_e for an edge e depends on its classification. Let v and w denote the significant and insignificant vertex of e respectively.

For basic and evading edges, the maximal distance is attained in the apex of a step. Let α_1 denote the absolute angle between vector $w - v$ and the assigned direction of e. Similarly, α_2 denotes the absolute angle between vector $w - v$ and the other associated direction of e. Basic computations show that a step must span less than $l_{\text{max}} = \big((\tan \alpha_1)^{-1} + (\tan \alpha_2)^{-1}\big) d_e/2$ to avoid deviating more than a distance $d_e/2$ from e. Hence, the number of steps is computed as $\lceil \|e\|/l_{\text{max}} \rceil_2$, where $\lceil x \rceil_2$ denotes the smallest even integer that is strictly greater than x. Note that the staircase may not be at exactly distance $d_e/2$ from e; hence, if x is an even integer, $\lceil x \rceil_2$ equals $x + 2$.

An aligned deviating edge does not use the step number. Instead it requires a distance δ_e. This distance is an upper bound on the maximal distance between the staircase of e and e itself. Hence, we use $\delta_e = \min\{d_e/2, \Delta_e\}$, where $\Delta_e = 0.1\|e\|$ as defined for the staircase regions.

For an unaligned deviating edge, the maximal distance to e is attained in one of the corners of the appended region. The easiest way to compute s_e is by first computing the maximal distance to e when using a step of unit length and the corresponding appended region. Let d_1 denote this maximal distance for e. Scaling to a different step length scales the entire step and appended region uniformly. Hence, we find that $d_f = f \cdot d_1$ when scaling by a factor f. Since f corresponds directly to the step length, we find that the maximum step length is $d_e/(2d_1)$. From this, we can derive that $s_e = \max\{4, \lceil 2d_1\|e\|/d_e + 1 \rceil_2\}$.

7.3.4 Analysis

For the analysis of Algorithm 7.2, let n denote the complexity of the input subdivision S and m the complexity of the resulting $\mathcal{C}$-oriented subdivision R. Since every edge of the input is converted into a staircase, m is at least as big as n. Often, consecutive edges may have oppositely assigned directions at their shared vertex. In such cases, the shared vertex becomes superfluous in the resulting subdivision. Removing these vertices allows m to become smaller than n, though this is unlikely to occur in practice.

Depending on the vertex-edge distances and the angles between edges with a shared vertex, the staircases use less or more steps. The smaller the vertex-edge distances or angles become, the more steps a staircase needs such that no intersections are introduced. Most importantly, the number of steps for a single edge does not depend on n. Hence, m is worst-case linear in n. A theoretic upper bound can be computed based on minimal angles, lengths, and distances. However, the increase predicted by such an upper bound is far greater than the increase observed in practice. Deviating edges especially may cause a large increase (locally) in the number of edges in worst-case conditions. These are caused by vertices of degree 3 or 4 such that all edges lie in the same sector or same adjacent sectors. This situation rarely occurs for territorial outlines.

Table 7.1 lists the increase in complexity caused by Algorithm 7.2 for the territorial outlines that are used in Section 7.4. To measure the increase, the table contains both the

Table 7.1 The increase in complexity caused by Algorithm 7.2 for the various inputs used in Section 7.4. The increase is given as a percentage of the output complexity, m, with respect to the input complexity, n.

Figure	**Input**	n	$\mathcal{C}$	m	**Increase**
Fig. 7.20	Languedoc-Roussillon	847	$\mathcal{C}_4$	2 046	242%
Fig. 7.21	Antartica	225	$\mathcal{C}_3$	686	305%
			$\mathcal{C}_3(\frac{\pi}{2})$	735	327%
			$\mathcal{C}_6$	618	275%
Fig. 7.22	Australia	223	$\mathcal{C}_2$	682	306%
			$\mathcal{C}_3$	580	260%
			$\mathcal{C}_4$	597	268%
			$\mathcal{C}_5$	616	276%
Fig. 7.23	Great Britain	549	$\mathcal{C}_3$	1 808	329%
			$\mathcal{C}_4$	1 369	249%
			$\{\frac{\pi}{12}, \frac{5\pi}{12}, \frac{7\pi}{12}\}$	2 428	442%
Fig. 7.24	Italy	1 297	$\mathcal{C}_4$	3 020	233%
Fig. 7.25	Japan	2 000	$\mathcal{C}_4$	5 286	264%
			$\mathcal{C}_6$	5 494	275%
Fig. 7.26	Southeast Asia	250	$\mathcal{C}_4$	658	263%
			$\{\frac{\pi}{12}, \frac{\pi}{2}, \frac{11\pi}{12}\}$	719	288%
Fig. 7.27	The Netherlands	1 506	$\mathcal{C}_4$	3 804	253%
Fig. 7.28	Dutch provinces	2 010	$\mathcal{C}_4$	5 086	253%

complexity of the input subdivision as well as the complexity of the $\mathcal{C}$-oriented subdivision as computed by the angular-restriction algorithm. The latter indicates the complexity, after removing the beforementioned superfluous vertices. In these experiments, the increase was on average 284% and never exceeded 450%.

The angular-restriction algorithm described in this section executes in $O(n^2 + m)$ time. This assumes a simple implementation in which the interfering edges are found by iterating over the other edges (Algorithm 7.2, Line 5). This may possibly be improved upon by using a more intelligent search. However, note that an improvement here does not yield an asymptotic improvement, when combining Algorithm 7.2 with the simplification algorithm which runs in $O(m^2)$ time.

7.4 Experimental results

We have implemented the algorithms described in this chapter and we used them to generate results for various territorial outlines. Here we present and briefly discuss results for single outlines (polygons) and multiple outlines (subdivisions). There are three different types of results, indicated in the figures with different colors. Simplification results are given in orange and have been computed using only Algorithm 7.1. Schematization results are obtained by applying the simplification algorithm to the result of Algorithm 7.2. Results that use a regular orientation set are indicated in blue; results that use an irregular set are indicated in purple.

Polygons. Figure 7.20 shows a sequence of octilinear schematizations for Languedoc-Roussillon (a department of France), starting from its angular restriction to the final convex polygon. The first results (a–c) are barely distinguishable from the input. The results with medium complexity (d–f) are clearly recognizable as schematizations and have a shape that corresponds well to the original geometry. Even at low complexity (g), the result is very reasonable, given the constraints. The final result is a convex polygon with five edges (h). On this extremely low complexity, the result deteriorates; a better schematization is illustrated. Many shapes suffer problems at extremely low complexities; the resulting convex polygon is often not desirable. Hence, for the other examples, we shall showcase only results with a few more edges.

Figure 7.21 shows results for the continent of Antarctica. Two schematizations use hexilinear orientation sets with different initial angles (c–d). In general, the horizontal variant (c) better represents the horizontal parts (that is, the upper and lower boundary); the vertical variant (d) better represents the vertical parts (that is, the left and right boundary). However, combining the two sets into a dodecilinear orientation set yields a result (e) that is worse than either one. Hence, we conclude that simply providing more orientations does not necessarily imply a better schematization.

Figure 7.22 shows results for Australia using a variety of regular orientation sets. The rectilinear result is likely hard to recognize separately, but we consider it a good schematization for the given constraints. It is likely to combine well with, say, a rectilinear network. The other schematizations are closer to the original shape. Especially the hex-

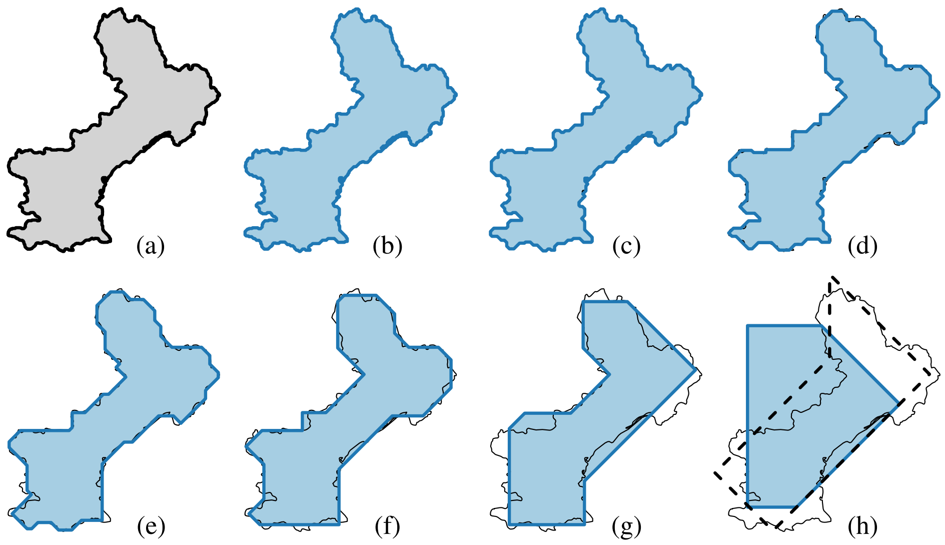


Figure 7.20 Octilinear schematization results with decreasing complexity for Languedoc-Roussillon. (a) Angular restriction, 2 046 edges (up from 847). (b) 499 edges. (c) 250 edges. (d) 100 edges. (e) 50 edges. (f) 24 edges. (g) 11 edges. (h) 5 edges. An alternative, superior solution is sketched by the dashed polygon.

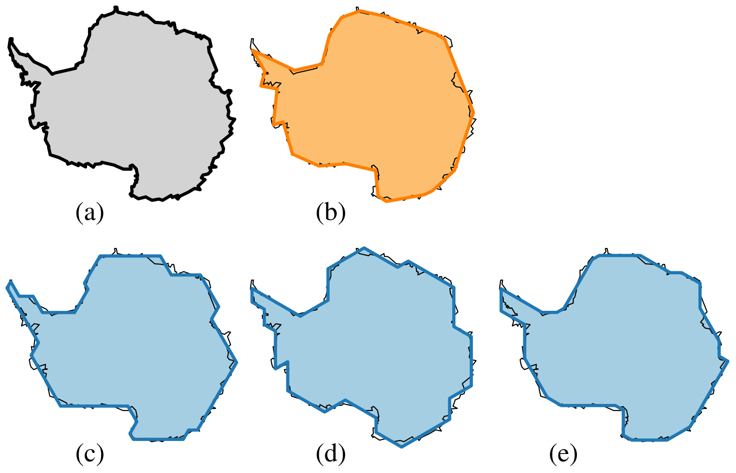


Figure 7.21 Results for Antarctica with 25 edges. (a) Input. (b) Simplification. (c) Hexilinear, $\mathcal{C}_3$. (d) Hexilinear, $\mathcal{C}_3(\frac{\pi}{2})$. (e) Dodecilinear, $\mathcal{C}_6$.

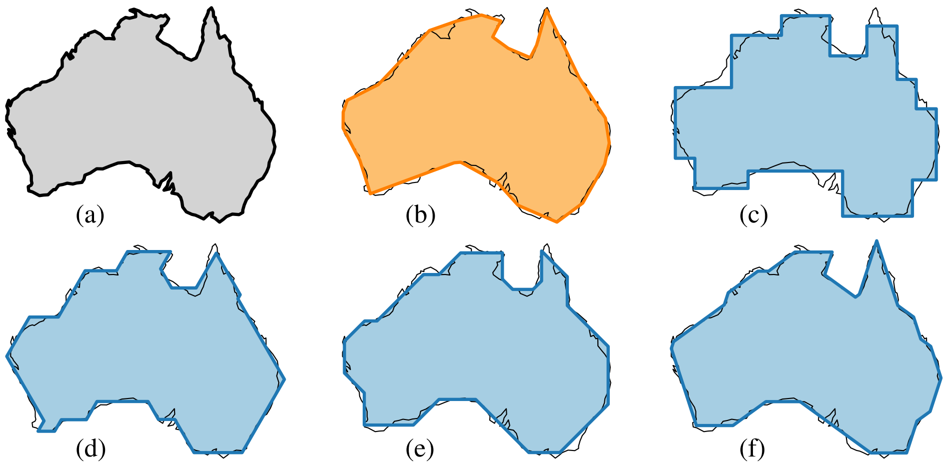


Figure 7.22 Results for Australia with (at most) 25 edges. (a) Input. (b) Simplification. (c) Rectilinear, $\mathcal{C}_2$. (d) Hexilinear, $\mathcal{C}_3$. (e) Octilinear, $\mathcal{C}_4$. (f) Decilinear, $\mathcal{C}_5$.

ilinear and decilinear schematizations work well. However, the ocilinear variant seems slightly worse. In particular, the Gulf of Carpentaria in the north is represented worse.

Figure 7.23 shows various results for Great Britain. We consider the hexilinear schematization to be the best result. With octilinear orientations, the schematization requires multiple edges to represent the rather straight eastern coastline. The result with the irregular set works quite well considering its restrictions. However, a small edge remains on the southern coast. These edges could probably have been put to better use by giving some more detail to Wales instead. Also, the Thames estuary is represented only slightly.

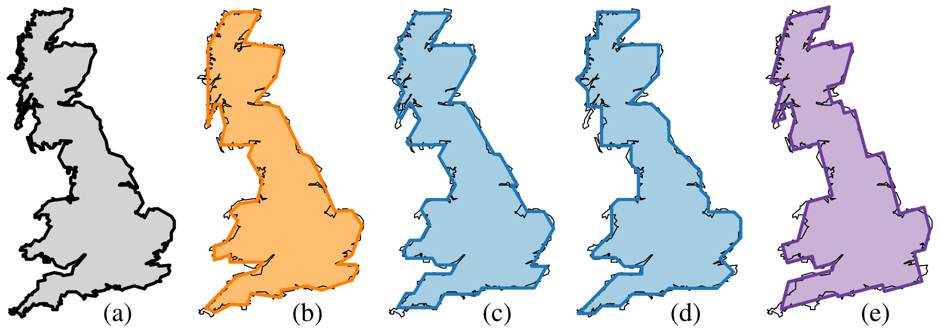


Figure 7.23 Results for Great Britain with 50 edges. (a) Input. (b) Simplification. (c) Hexilinear, $\mathcal{C}_3$. (d) Octilinear, $\mathcal{C}_4$. (e) Irregular, $\{\frac{\pi}{12}, \frac{5\pi}{12}, \frac{7\pi}{12}\}$.

Subdivisions. Figure 7.24 shows the simplification and octilinear schematization of the regions of Italy. For schematization purposes, it uses quite a high number of edges. However, this seems unavoidable in order to maintain some geographic shape, since there are many borders to be represented. Our algorithm is able to further reduce the complexity, but recognizability suffers greatly. In particular, the southern regions of Puglia and Calabria deteriorate and make Italy lose its characteristic "boot shape".

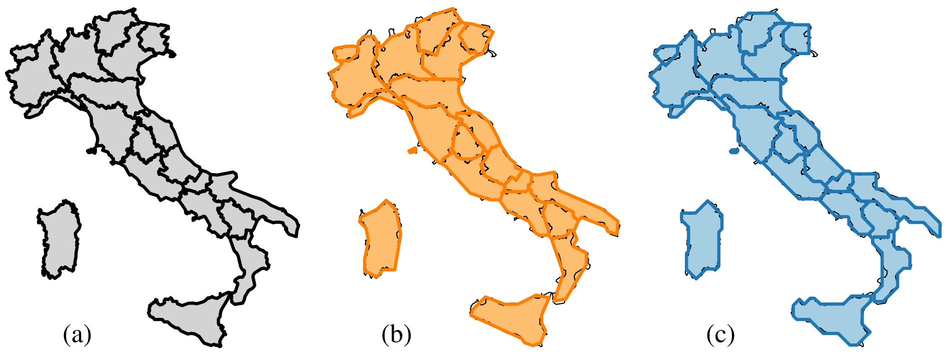


Figure 7.24 Results for the provinces of Italy with 244 edges. (a) Input. (b) Simplification. (c) Octilinear, $\mathcal{C}_4$.

Figure 7.25 shows a simplification and schematizations of the four major islands of Japan. The dodecilinear result (d) is considered $\mathcal{C}$-oriented, but it has relatively many orientations and edges. This causes the result to look more like an unrestricted schematic outline instead. Despite its angular restrictions—or perhaps because of it—the dodecilinear result approximates some parts better than the simplification. For example, the southernmost island (Kyushu) retains more of its shape and the southern shore of the largest island (Honshu) has a better representation.

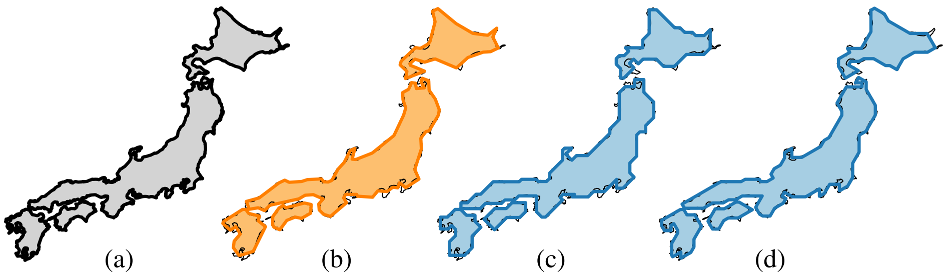


Figure 7.25 Results for four islands of Japan with 120 edges. (a) Input. (b) Simplification. (c) Octilinear, $\mathcal{C}_4$. (d) Dodecilinear, $\mathcal{C}_6$.

Figure 7.26 (also shown in the introduction of this chapter) shows results for a group of countries in southeast Asia (Cambodia, Laos, part of Malaysia, Myanmar, Thailand, and Vietnam). Their shapes have been preserved quite well in all three results. However, both schematizations (and to a lesser extend the simplification) have made the narrow part of Thailand even more narrow, even so far as to suggest that these are actually two separate regions, rather than one. Though it is geometrically quite accurate, it would probably be better for legibility to give this strip some slight exaggeration. Moreover, the local topology changed for some of the degree-3 vertices in the schematization. For example, the southernmost vertex of the Myanmar-Thailand border in the octilinear schematization changed from having the shoreline continue to having the border continue southwards. Similarly, the easternmost vertex of the Malaysia-Thailand border changed. We do not consider this to be very problematic in these cases. However, if so desired, it can likely be counteracted by restricting edge-moves near degree-3 vertices.

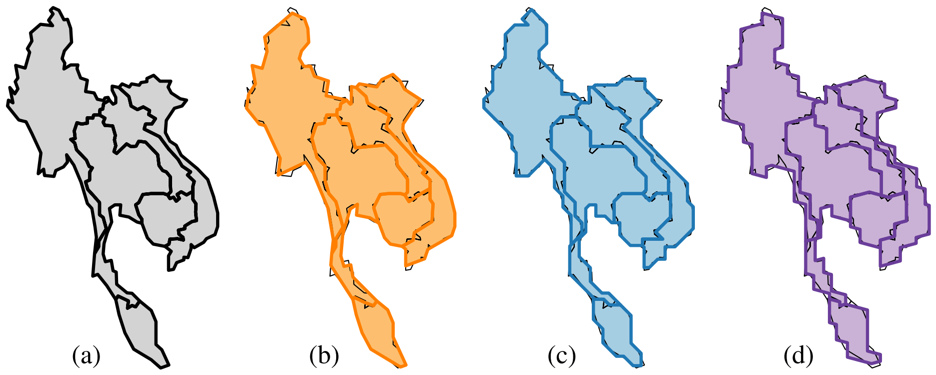


Figure 7.26 Results for a group of countries in southeast Asia with 130 edges. (a) Input. (b) Simplification. (c) Octilinear, $\mathcal{C}_4$. (d) Irregular set, $\{\frac{\pi}{12}, \frac{\pi}{2}, \frac{11\pi}{12}\}$.

In Figure 7.27 we show our schematization result for the Netherlands, at equal complexity as the manual schematization given in Figure 1.1(a). For comparison, we also show only the outline of this manual schematization. Note that, for example, the exaggeration of southern Limburg is not possible with our algorithm. Figure 7.28 illustrates two results for schematizing the provinces of the Netherlands. Using the same number of edges as for Figure 1.1(b) results in a coarser schematization. However, the input has approximately 33% more edges; the second result compensates for this by allowing 320 edges in the result. The resulting schematization has a reduction in detail comparable to the result without the province borders.

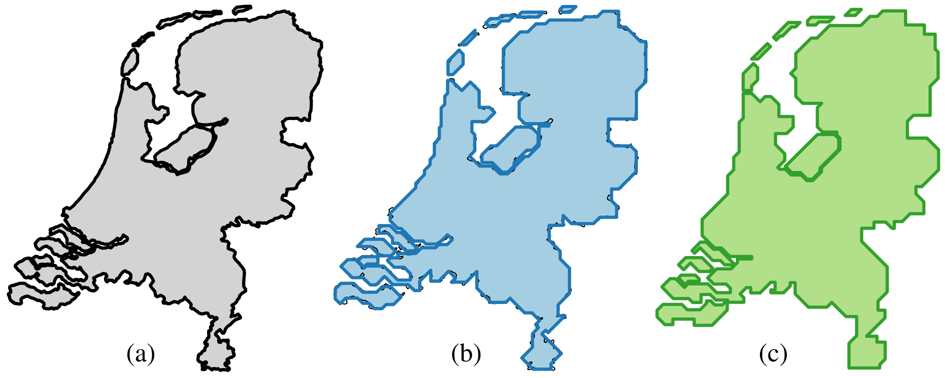


Figure 7.27 Result for the Netherlands. (a) Input. (b) Octilinear, $\mathcal{C}_4$, with 240 edges. (c) Outline extracted from Figure 1.1(a); this outline has 240 edges.

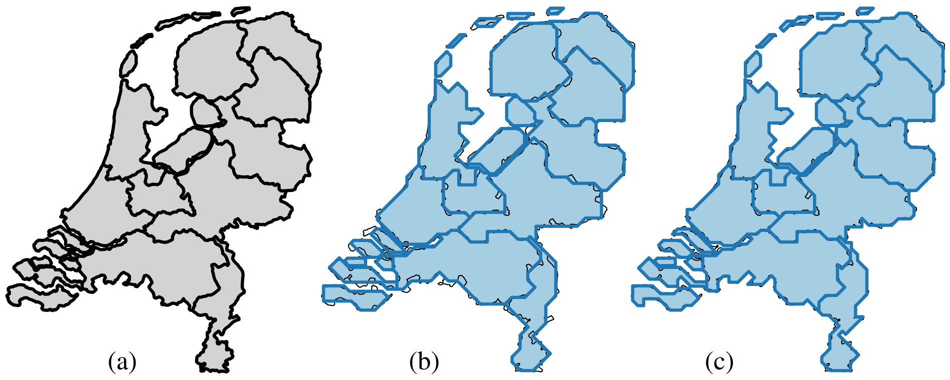


Figure 7.28 Results for the provinces of the Netherlands. (a) Input. (b) Octilinear, $\mathcal{C}_4$, with 240 edges. (c) Octilinear, $\mathcal{C}_4$, with 320 edges.

7.5 Conclusions

In this chapter we studied the problem of simplifying and schematizing territorial outlines in combination with an area-preservation constraint. To this end, we introduced an operation called an edge-move. In our algorithm, we perform these edge-moves in pairs such that the complexity of the outline is reduced in an area-preserving and topologically safe way. We proved that any nonconvex polygon admits such a pair of edge-moves. The algorithm is a nonvertex-restricted simplification algorithm.

Since edge-moves do not change orientations, we know that the output of this simplification algorithm is $\mathcal{C}$-oriented if the input is $\mathcal{C}$-oriented. Hence, we introduced an area-preserving algorithm to convert any subdivision into a topologically correct, $\mathcal{C}$-oriented

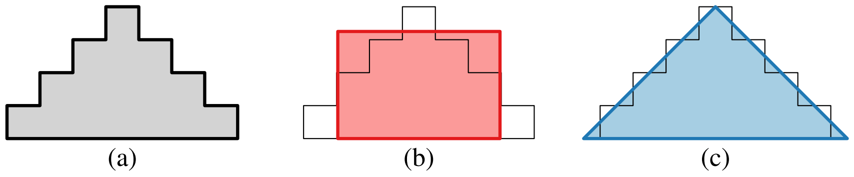


Figure 7.29 (a) A rectilinear "triangle". (b) An octilinear result of our algorithm uses only rectilinear orientations. (c) A proper octilinear schematization, unattainable by our current algorithm.

subdivision. Combining these two algorithms, we obtain an algorithm to compute a $\mathcal{C}$-oriented schematization of any simple polygon or subdivision. We observe that in some cases it is actually desirable to introduce an orientation that is not present in the input, as is illustrated in Figure 7.29.

Our experiments show that these algorithms preserve the important structures of the input. However, we assumed that the desired orientations are given. The quality of schematization depends quite strongly on the chosen orientation set $\mathcal{C}$. Hence, it is desirable to develop algorithms to decide what set $\mathcal{C}$ is suitable for a given outline.

Our algorithm chooses operations that change the area as little as possible. That is, we greedily choose the contraction that causes the least symmetric difference with respect to the current result. While this strategy allows for an efficient algorithm, it may not always be the "best" choice for the complete simplification. That is, choices that are "worse" according to the symmetric difference may lead to a better solution in the end. Moreover, it might lead to an asymmetric result for a symmetric input. Hence, other strategies or other criteria might be more appropriate. For example, Haunert [100] presents an algorithm for detecting symmetries in buildings and building groups. Is such a method effective also for territorial outlines? If we can augment territorial outlines with symmetry information, how can we incorporate this into our schematization algorithm?

To improve upon the greedy nature of the iterative algorithm, one may consider a strategy to optimize a common distance measure. However, as argued in Chapter 3, such optimization may yield unsatisfactory results as well. Nonetheless, we observe that not all counterexamples are reachable by edge-moves, while the illustrated better result is reachable. For example, this is the case for the symmetric difference. The question rises whether it is desirable to find a result that minimizes the symmetric difference over all results that can be obtained using valid edge-moves. If so, is there an efficient algorithm to compute it? Our greedy approach does not minimize the symmetric difference. Moreover, an optimal result for a given complexity k does not necessarily lead to an optimal result for lower complexity.

Part III

Widening the scope

Chapter 8

Building Generalization

Generalization is the process of deriving a scale-appropriate map from detailed geographic data. It may refer to the manual process as performed by cartographers or to the automated process; in the context of this thesis, however, we shall use it to refer to the automated process. Generalization is typically used to produce topographic maps, ensuring maximum detail in a legible map. Generalization involves a variety of different *generalization operators*. Such an operator describes a spatial transformation on the geometry representing various elements such as roads, rivers and buildings. We briefly discuss some of the main operators. We refer to the work of Regnauld and McMaster [147] for an extensive discussion on generalization operators.

Simplification is one of the basic operators for generalization. It is the task of reducing the number of points that represent a shape. In the model of Regnauld and McMaster, simplification refers to vertex-restricted variants. Vertex displacement is treated as a separate operator: *smoothing*. Nonvertex-restricted simplification combines these two operators.

As the scale of a map decreases, small features become hard to read. A number of operators exist to alleviate such issues. Generally speaking, there are three options: features are enlarged, aggregated with other features, or removed completely. Feature enlargement includes the operators *enhancement* and *exaggeration*. Aggregation includes a variety of operators, depending on the type of features. Two examples are *aggregation* (of points) and *amalgamation* (of areas). Removal of features is accomplished by the operator *elimination* (or selection). *Typification* is an operator that replaces elements by a "typical shape" that does not necessarily accurately reflect the actual geometry. For example, a set of buildings can be replaced by a number of equally sized squares.

Buildings are man-made objects; therefore, walls typically make 90-degree angles. Due to possible measurement imprecision in the data, the building outline may deviate from this. It is often desirable to restore these right angles in the building outline. This building-specific operator is referred to as *building wall squaring*.

When a number of features are spatially close, the reduction of the map scale may cause these features to coalesce into what appears as a single feature. Often, this is undesirable and the operator *displacement* is used to avoid this coalescence.

In this chapter we apply the algorithms of Chapter 7 in the context of building generalization. The purpose of this is to show the versatility of this approach: it can be used effectively for problems other than schematization of territorial outlines. In Section 8.1 we consider building wall squaring. We describe a few preprocessing steps to make our algorithm suitable for this task. In Section 8.2 we consider the generalization of urban areas, by extending our methods with aggregation and elimination operators. Both sections include some experimental results to demonstrate the effectiveness of our methods.

Related work. A central technique in generalization is referred to as (multi-)*agents* [21, 116, 155, 161]. It is a framework for multicriteria optimization: each agent is in charge of generalizing certain aspects and regions of the map. The agents apply generalization operators to obtain their generalization goals and constraints. However, these may conflict; the solution is optimized to balance the degree to which each agent meets its goals and constraints. This framework requires efficient and effective implementations for the various generalization operators.

Implementations of the various generalization operators have received significant attention, such as typification [46, 142, 146] and displacement [18, 119, 158]. Here we focus on the operators for building generalization that are considered in this chapter: wall squaring, simplification and aggregation. A large number of simplification methods exist (see Chapter 7); here we focus on methods that are designed specifically for buildings.

For building wall squaring, it is essential to understand and derive what the "orientation" of a building is. This need resulted in several methods to compute the orientation [73, 142, 145, 154]. Regnauld [145] observed that there is a difference in detecting the wall orientations and the general orientation of a building. To apply our methods for building wall squaring, we are interested in the wall orientations rather than the general orientation. To this end, we adapt a method described by Duchêne *et al.* [73].

Sester [157, 158] applies least-squares adjustment for building simplification and wall squaring. Mayer [125] uses scale-spaces to shift edges of an outline to simplify buildings and includes a method for building wall squaring; he also extends these ideas to 3D buildings. Haunert and Wolff [103] apply shortcuts to formulate an ILP for the topologically safe simplification of buildings. In addition to minimizing the number of edges in the result, they add terms to the optimization criterion to factor in area change, corner angles and "edge directions" of buildings.

Aggregation conceptually consists of two high-level steps: deciding which objects to aggregate and how to aggregate them. For the first step, Steiniger and Weibel [162] distinguish five relation types: geometric, topological, semantic, statistical and structural. For example, buildings may be grouped based on the road network (topology) [26, 116, 155]. However, other relations can also be considered for buildings (e.g. [26, 182]). For the second step, the actual aggregation, a number of methods have been developed. Ware *et al.* [175] use a triangulation to aggregate buildings by "adding" triangles to connect them. Regnauld and Revell [148] describe an approach to building aggregation, involving a set of special operations, including grouping, rotation and simplification. Damen *et al.* [62] use the Minkowski sum and subtraction ("polygon offsetting") for the simplification, aggregation and elimination of buildings.

Not only buildings but also regions can be aggregated. The regions are often referred to as area partitions in this context. A number of aggregation methods have been suggested (e.g. [55, 102, 134]). Van Oosterom [134] introduced the generalized area partition (GAP) tree to represent a hierarchy of aggregation. It has been extended, for example, to include buildings [140], topological constraints [171] and higher dimensional data [135].

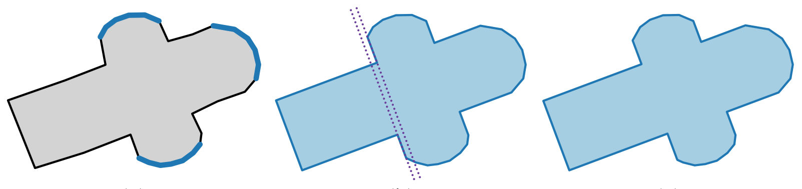
In comparison to the methods discussed above, our method is unique in preserving the exact area of buildings during simplification. Our techniques may be useful in conjunction with other techniques for building generalization. For example, we choose to integrate aggregation and elimination into our simplification algorithm of Chapter 7. This allows for a single execution of the algorithm that produces a range of generalizations. If the desired scale is known, these operators can also be performed beforehand using other known techniques. As mentioned, road networks can often be used to partition the input into small logical groups. We develop our methods without taking such auxiliary information into account. However, if such data is available, our methods can incorporate these to not only speed up the computation but also to guarantee that buildings do not cross roads.

8.1 Building wall squaring

Buildings are man-made objects and as such often have right-angle corners. However, given outlines may be inaccurate. Building wall squaring is the operator that restores these right-angle corners where appropriate. We first discuss how to detect arcs in a building outline, as deforming arcs as a result of squaring should be avoided. We perform squaring by restricting the orientations, followed by a sequence of edge-moves to return to the original complexity of the building. However, we do not wish to assume that the orientation of the building is known; hence, we also present a method to detect the set of orientations to be used for the angular restriction.

Arc detection. Some buildings have characteristic arcs, such as a church having circular chapels (see Figure 8.1). Although it may require a high number of vertices in a polygonal representation, such a feature has low *visual* complexity: the arc may be perceived as a single object. To avoid squaring and deforming such arcs, we detect these beforehand, marking the vertices as *arc vertices*. We characterize an arc as a sequence of vertices that have similar exterior angles and segment lengths. We define an *arc sequence* to be a sequence $\langle v_1, \ldots, v_n \rangle$ of vertices such that the following five conditions are met.

- $n \geqslant 4$. A sequence of three vertices has only one bend and thus we cannot discern whether it describes an arc or a bend.
- $|\alpha(v_i)| \leqslant \pi/6$ for $i \in \{2, \ldots, n-1\}$, where $\alpha(v_i)$ denotes the exterior angle at vertex v_i. Sharp bends break the perception of an arc. Therefore, we constrain the exterior angle to be at most 30 degrees.
- $|\alpha(v_i) - \alpha(v_{i+1})| \leqslant \pi/12$ for $i \in \{2, \ldots, n-2\}$. The exterior angle of consecutive vertices must be similar. In particular, they may not deviate more than 15 degrees. A gradual change in exterior angle is allowed to support arcs that change in curvature.



(a) (b) (c)

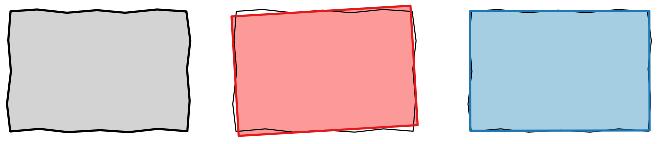
Figure 8.1 (a) Church with round walls. Detected arcs are indicated in blue. (b) Its squared outline. (c) Walls are aligned with edge-moves.

- $1/3 \leqslant \|v_i - v_{i-1}\| / \|v_{i+1} - v_i\| \leqslant 3$ for $i \in \{2, \dots, n-1\}$. If consecutive distances between vertices in the sequence differ greatly, the impression of an arc is reduced. Hence, we bound the ratio by 3.
- $\overline{v_i - v_1} \cdot \overline{v_n - v_1} \leqslant 0.99$ for some $i \in \{2, \dots, n-1\}$, where $\overline{v}$ denotes a normalized vector. A straight wall that is represented using multiple vertices should not be considered an arc. Therefore, we require that at least one vertex lies sufficiently far from the line connecting the first and last vertex in the sequence.

Any vertex that is part of an arc sequence is marked as an arc vertex. Note that the constants in the above conditions can be considered as parameters. They can be changed to classify or avoid classifying certain sequences as arcs sequences. We maintain the given constant values throughout our experiments.

Orientation detection. To detect the orientations in a building, we extend the method by Duchêne *et al.* [73]. Their method takes a set of candidate orientations. For each candidate orientation, it computes the weighted sum of edge lengths of edges contributing to that orientation. An edge contributes to a candidate orientation if its orientation deviates only by a small angle from the candidate orientation or its perpendicular. In other words, the angles are considered modulo 90 degrees. The weight of an edge depends linearly on this angle. We modify their method in two ways.

The linear contribution function ensures that a maximal value always occurs at the orientation of one of the edges. As illustrated in Figure 8.2, this is not always desirable when dealing with noisy data. Instead, we use a Gaussian function with a peak equal to the edge length and a width of 7.5 degrees (see Figure 8.3(a)). Consider an edge of



(a) (b) (c)

Figure 8.2 (a) A building with imprecise walls. (b) Result with linear contribution function. (c) Result with Gaussian contribution function.

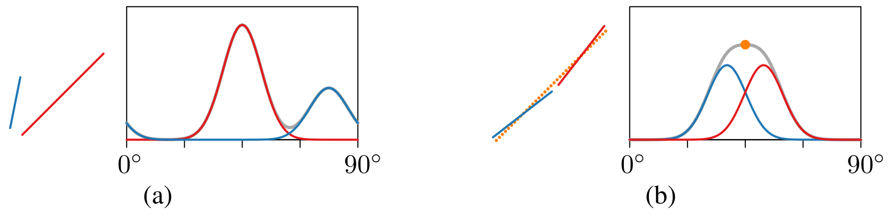


Figure 8.3 (a) Two Gaussian functions with a width of $7.5°$. (b) The maximal value (orange) does not need to correspond to an input orientation.

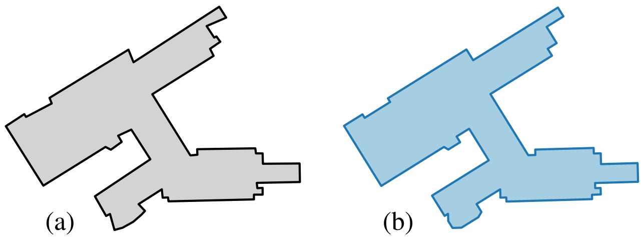


Figure 8.4 (a) Building with two distinct orientations. (b) Its squared outline.

length l with an orientation of o degrees (modulo 90). The contribution of this edge to an angle $a \in [0, 90)$ is computed as $l \cdot e^{-x^2/112.5}$ where $x = \min\{|a - o|, 90 - |a - o|\}$ is the angular deviation between o and a modulo 90 degrees. Comparing to a linear contribution function, the Gaussian function results in a higher contribution to orientations that are close to the orientation of the edge. The maximal value may now correspond to an orientation that is not present in the input (see Figure 8.3(b)). Instead of using a sampled approach to find an orientation with maximal contribution, we compute the desired orientation using numerical methods.

A building may have multiple orientations, varying between parts of the outline, as shown in Figure 8.4. Duchêne *et al.* [73] compute only the main orientation of a building (and its perpendicular), using the presence of other orientations to define a confidence level. Instead of locating only the main orientation, we discard the edges that deviate at most 15 degrees from the detected orientation (or its perpendicular). We then repeat the detection process until no edges remain.

Squaring. Now that arcs and orientations have been detected, we perform two steps to obtain the squared outline. First, we restrict the input to use the detected orientations, using the angular-restriction algorithm described in Section 7.3. However, we do not change the edges that end in two arc vertices, to avoid deforming the arcs. If the edges of a degree-2 vertex have opposing associated directions, the vertex is marked as being *superfluous*. These superfluous vertices are positions along a (nearly) straight wall and should be removed after squaring. This occurs, for example, for all vertices except for the four corner points in the building outline depicted in Figure 8.2.

After the angular restriction, we execute the simplification algorithm described in Section 7.2. Long edges in the building outline cause a staircase with long edges; to simplify these staircases, the long edges have to be moved. This typically involves a contraction area that is large in comparison to those of the finer details of the building. Therefore, we redefine the minimal contraction pair to be based on the distance that the inner edge of the configuration moves instead of the area it sweeps over. Edge-moves that involve the displacement of an arc vertex are not used. The desired complexity is set to the number of nonsuperfluous vertices of the original outline.

Simplification. It may be desirable to further simplify the building outline. Our simplification algorithm can perform more edge-moves to further reduce the complexity. However, the arcs should also be simplified if the desired complexity is sufficiently low. In this case, we unmark the arc vertices and impose the orientation restriction on the incident edges. That is, during the simplification we treat the arc vertices as regular vertices. This step should be done after squaring and before any further simplification: this avoids large deformations that may otherwise be unnecessary.

Wall alignment. Edge-moves can be further exploited to align edges (walls) that have the same orientation and are almost collinear. For two edges that are similarly directed (e.g. the building interior is above the edge for both), this can be done in an area-preserving way (see Figure 8.5(a)). However, if the walls are oppositely directed, this need not be possible or may require a large deformation. Though it would be possible to apply other edge-moves to compensate for area gain or loss, we choose not to allow this: it may cause a deformation that is worse than the misaligned walls; it risks misalignment of walls that were aligned. By relaxing the area-preservation constraint, the misaligned walls with opposite directions can move towards each other (see Figure 8.5(b)). Another example of aligning walls is given in Figure 8.1(c). For the other results, we choose not to apply this optional postprocessing step.

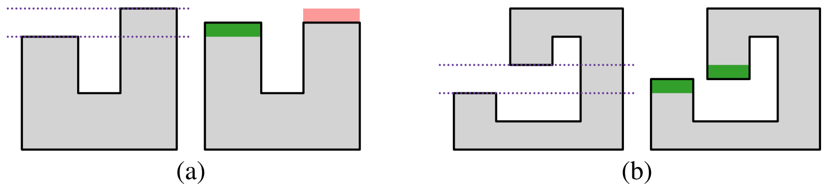


Figure 8.5 (a) If two misaligned walls are similarly directed, edge-moves can align them while preserving area. Area gain and loss are indicated in green and red respectively. (b) If the walls are oppositely directed, we may align them at the cost of changing area.

Results. The earlier results shown in Figure 8.1(b), Figure 8.2(c) and Figure 8.4(b) were obtained by applying the methods described in this section in combination with the techniques of Chapter 7. Below, we briefly discuss two additional results.

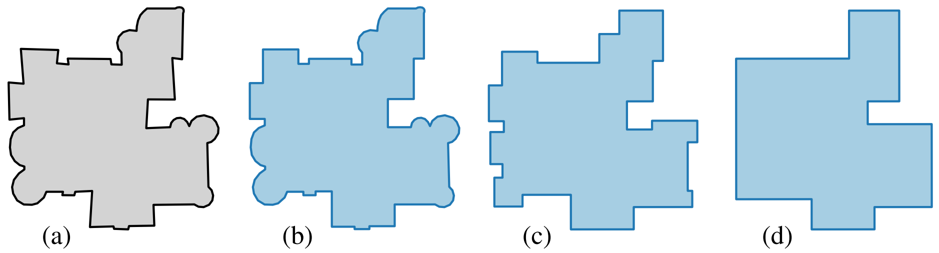

(a) (b) (c) (d)

Figure 8.6 (a) Building with 95 edges. (b) Squared outline with 95 edges with arc detection. (c) Simplified to 38 edges after squaring. (d) Simplified to 14 edges after squaring.

Figure 8.6 shows three results of a complex building obtained using the methods described in this section. The building has been neatly squared after detecting arcs (see Figure 8.6(b)). However, edges that are not part of an arc but that do end in two arc vertices are not modified, for example the vertical edge in the bottom right. The cause is that its end points are both arc vertices; hence the edge is considered to be part of an arc. Releasing the arc vertices and imposing the angular restrictions on the edges allows us to further simplify the building while maintaining the foremost features (Figure 8.6(c–d)).

Figure 8.7 illustrates the effect of arcs on the visual complexity for the main building of the airport in Boston. The result without arc detection (b) obviously obtains the lowest (measured) complexity. However, the visual complexity is not reduced in comparison to (c): what is perceived as a single arc is now crudely represented using more than one line segment. Figure 8.8 shows the result for the same building when also applying wall squaring (b) and simplifying afterwards (c), both using arc detection beforehand. The building has quite a few small arcs that appear to be a simple offset of a larger arc. Since these vertices are fixed by the arc detection, these details are not eliminated by simplification. For simplification with detected arcs, it would be desirable to move such arcs to merge with the larger one. Details of such an operation are left as future work.

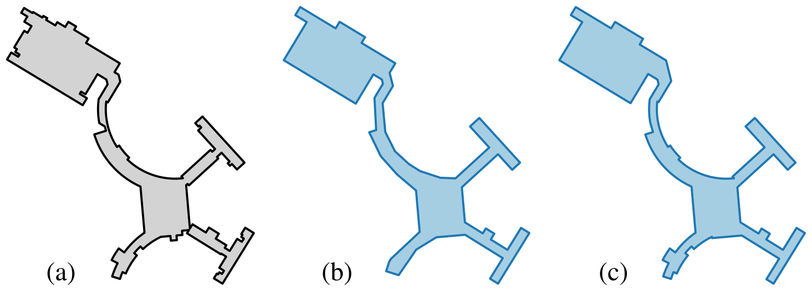

(a) (b) (c)

Figure 8.7 (a) Airport of Boston (361 edges). (b) Simplified to 54 edges. (c) Simplified to 295 edges with arc detection.

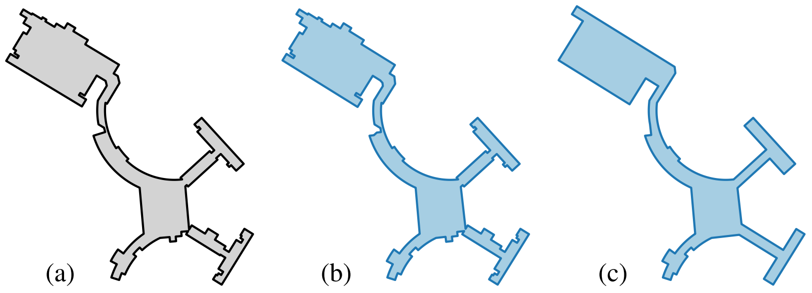


Figure 8.8 (a) Airport of Boston (361 edges). (b) Squared outline with 360 edges with arc detection. (c) Simplified to 283 edges with arc detection after squaring.

8.2 Generalizing urban areas

In this section we investigate generalization for urban areas, focusing on the buildings in such an area. Urban-area generalization can be achieved in part by simplification. A collection of building outlines can be represented as a subdivision. Therefore, the method that we described in Section 7.2 can be applied to obtain generalized versions of urban areas as well. Depending on the size of the area and the target scale, simplification alone does not suffice: additional generalization operators are required. In this section we integrate aggregation and elimination into our simplification method. We assume that we have detailed accurate building outlines available. Hence, we do not use squaring. Squaring could even cause misaligned walls for buildings on opposite sides of streets, greatly reducing the impression of a street caused by the empty region in between. This is illustrated in Figure 8.9.

To integrate elimination and aggregation into our simplification method, we describe operations that can be interleaved with the edge-moves. However, instead of reducing the complexity by moving edges, such operations aggregate two buildings or eliminate a building. For each operation, we also indicate a cost of operation; this cost is compared to the costs of edge-moves (area of contraction region) to prioritize operations.

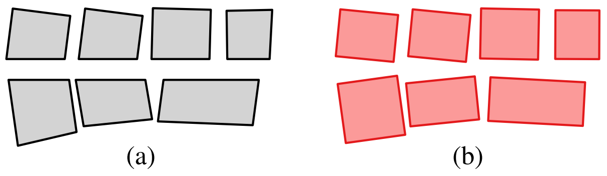


Figure 8.9 (a) Original outlines give the impression of a street. (b) Misaligned walls would cause the impression of the street to diminish.

Components. Before describing our aggregation operations, we introduce the notion of components via a dual graph (refer to Figure 8.10). Consider a subdivision S and its dual graph S^*, constructed as follows. Dual graph S^* contains a vertex f^* for every bounded face f in S. An edge between two vertices, f_1^* and f_2^*, exists if the corresponding faces, f_1 and f_2, share at least a vertex in S. We define a *component* of a subdivision S to be a maximal subgraph of which the incident bounded faces correspond to a maximal connected component in S^*. A component is enclosed by a cycle of which each edge is incident to the unbounded face of S. Note that this differs slightly from the standard definition of a (connected) component in a subdivision.

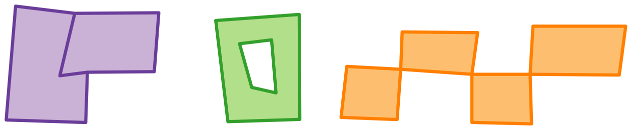


Figure 8.10 A subdivision with three components, each indicated with a unique color. The "courtyard" in the green component is not a separate entity but part of the component that encloses it.

Aggregation. The goal of our aggregation operations is to gradually turn a component into a single bounded face and a subdivision into a single component. We distinguish four operations; refer to Figure 8.11 for illustrations.

(a) An *interior merge* aggregates two adjacent faces within the same component.

(b) A *junction merge* aggregates two faces in the same component that meet only at a junction.

(c) An *edge-split* splits an edge that is incident only to the unbounded face.

(d) An *exterior merge* aggregates two separate components.

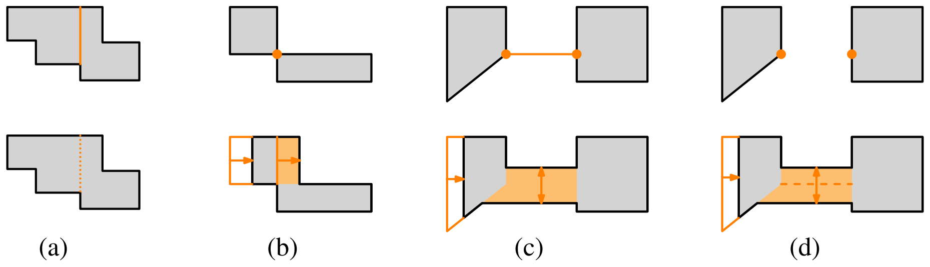


Figure 8.11 Four aggregation operations. Top and bottom rows depict situations respectively before and after performing the operation. Area deformation is exaggerated to clarify the effect. (a) An interior merge. (b) A junction merge. (c) An edge-split. (d) An exterior merge.

An interior merge aggregates two adjacent faces. That is, it requires that the faces have at least one border in common. The interior merge removes all common borders between the faces. The cost of an interior merge equals the area of the smaller of the two faces multiplied by some parameter α_{in}. Increasing or decreasing α_{in} makes the process respectively less or more likely to merge faces rather than performing edge-moves. The operation is illustrated in Figure 8.11(a).

Even if a component has no adjacent faces, it is possible that it consists of multiple faces (see Figure 8.11(b–c) for examples). There are two cases possible: a junction is alternatingly incident to a bounded face and the unbounded face; or an edge is incident only to the unbounded face. The former is resolved via a *junction merge*, the latter via an *edge-split*. Note that these operations are used only if an interior merge is not possible.

For a junction merge, we perform a positive edge-move to "cut" the junction, merging two "vertex-adjacent" bounded faces. Again, the increase in area can be compensated for by an edge-move in one of the merged faces. The cost of this operation is the area of the smallest merged face multiplied by α_{in}.

An edge-split operates on an edge that is incident only to the unbounded face. It assumes that at least one endpoint is a vertex that lies on a bounded face of the component. We duplicate the edge, creating a zero-area region in between the edge and its duplicate. At least one of these edges allows a positive edge-move. In the bounded face, we search for a negative edge-move to compensate for the area change. This effectively merges the face with the split edge, and possibly the face at the other endpoint of the edge. The number of edges in the component that are incident only to the unbounded face is reduced by one. The cost of this operation is the area of the (smallest) face multiplied by α_{in}.

An *exterior merge* aggregates two separate components. It first adds the shortest line segment between the two components. This new edge is adjacent only to the unbounded face. An edge-split is performed on this edge as part of the exterior merge. To prevent introducing a new orientation, we may create a staircase—using two distinct orientations in the components—for both new edges (see Section 7.3). The cost of an exterior merge equals the area of the smaller of the two components multiplied by some parameter α_{ex}.

Above, we introduced two parameters, α_{in} and α_{ex}, to prioritize aggregation operations and edge-moves. We require that $\alpha_{\text{in}} \leqslant \alpha_{\text{ex}}$ holds. This guarantees that when an exterior merge occurs, the smaller of the two components consists of only one face: that is, it is a simple polygon. It must then have a negative edge-move to compensate for the change in area. In our experiments, we use $\alpha_{\text{in}} = 0.2$ and $\alpha_{\text{ex}} = 0.5$.

By combining these four operations of aggregating faces and components, we obtain a method that is complete for subdivisions. This is formulated in the following theorem.

Theorem 8.1. *Let S be a simple polygonal subdivision. Unless S is a simple convex polygon, subdivision S can be generalized, while preserving area, orientations, and planarity, by one of the following operations: a complementary pair of feasible edge-moves, an interior merge, a junction merge, an edge-split or an exterior merge.*

Proof. If a component contains multiple bounded faces, then either an interior merge, junction merge or edge-split can be performed. If two bounded faces are adjacent, an interior merge is possible. Otherwise, the dual of this component (where each bounded face is connected to bounded faces with which it shares at least a vertex) must be a tree; in particular, this dual must have a leaf. The bounded face that is dual to a leaf consists of three or more vertices of which only one is a junction. This junction admits either a junction merge or an edge-split. This reduces either the number of faces in the component or the number of edges incident only to the unbounded face, whereas the other operations do not increase these values. Therefore, these operations make progress towards generalizing S if a component exists that contains multiple faces.

If no component has multiple faces, each component in S is a simple polygon. If there is only one component, then Theorem 7.1 implies that a complementary pair of feasible edge-moves exists. Otherwise, multiple components exist; the closest pair of the components admits an exterior merge. □

Implementation. By the theorem above, it is always possible to generalize a simple polygonal subdivision, while preserving the total area and orientations inherent in the input. However, we choose to not apply the additional steps that ensure the last two properties. We do not restrict orientations beforehand and there seems little value in introducing comparatively arbitrary orientations in an exterior merge. In addition, we relinquish the area-preservation constraint for the following reason. When considering a group of buildings, then the area that it occupies is greater than the area of the buildings alone. Any empty space in between the buildings can be considered to be part of such a neighborhood. Thus, enforcing area preservation may in fact cause apparent area *loss*. Hence, we do not compensate for the area increase caused by the exterior merge and even perform edge-moves on both newly introduced edges such that a maximal area is added. We do maintain the (possibly increased) area when performing the other operations. That is, the area is only increased when performing an exterior merge.

Also, it is undesirable to merge small components with a component that is far away. To this end, we use an *elimination* operation that simply removes a component from the subdivision. A component is considered an outlier when the nearest other building is further away than three times its diameter.

Discussion. We present results for a data set of building outlines of Boston[1]. This data set, with 295 781 components and 1 750 427 edges, is depicted in Figure 8.12. On the complete data set and on subsets, we ran Algorithm 7.1 without and with the extensions described in this section; we refer to the results respectively as “simplification” and “generalization”. In the following, we discuss only two subsets: the northern part of Roxbury (2 192 components; 15 192 edges) and North End (131 components; 2 035 edges). We refer to the former simply as “Roxbury”. The input and results of these subsets are given in Figure 8.13 and Figure 8.14. Some results for the complete data set are available online[2].

At a glance, the simplification and generalization of Roxbury (Figure 8.13) are quite similar; we consider their visual quality to be comparable. The advantage of the simplification is that the algorithm is easier to implement without the aggregation and elimination

[1]This data set is freely available as part of the Massachusetts Geographic Information System, MassGIS: `www.mass.gov/mgis/lidarbuildingfp2d.htm`, accessed February 2011.

[2]`www.win.tue.nl/~wmeulema/results.html#boston`

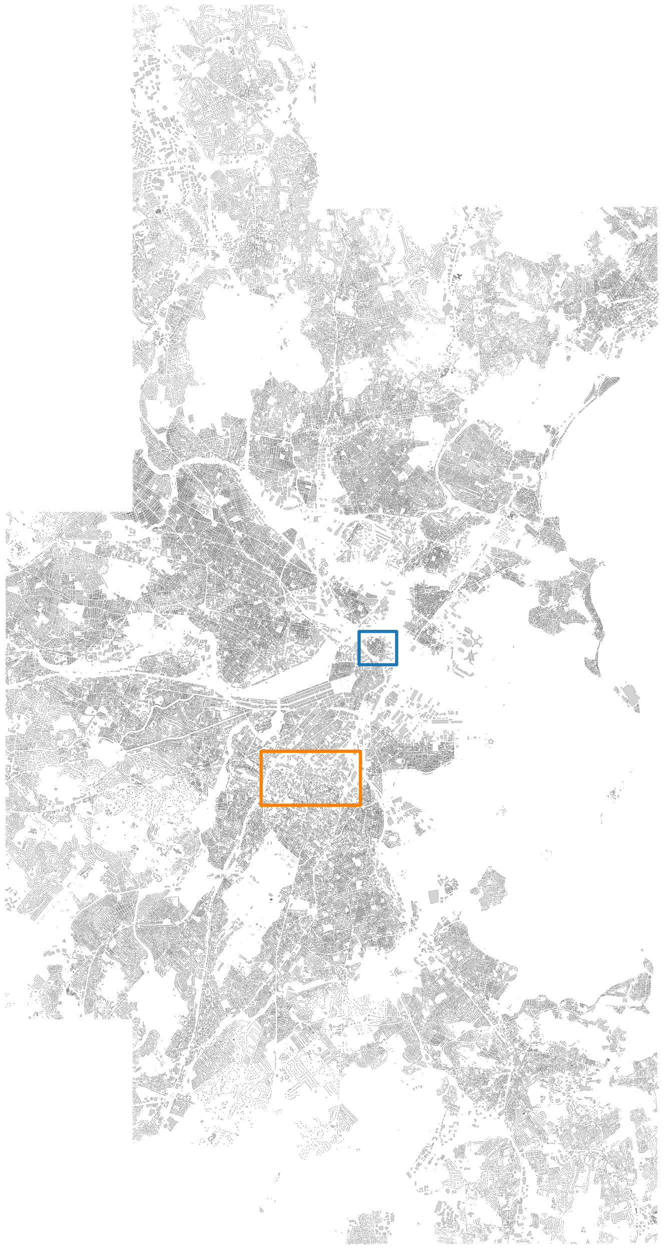
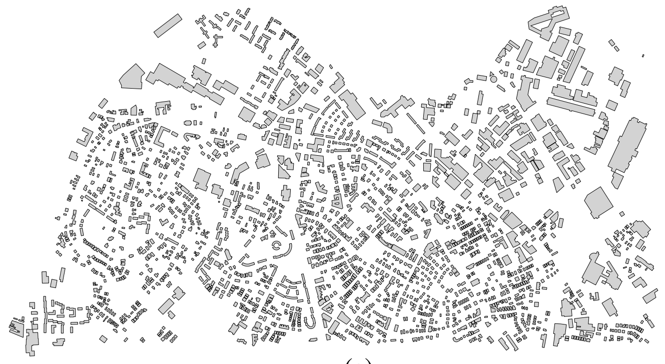
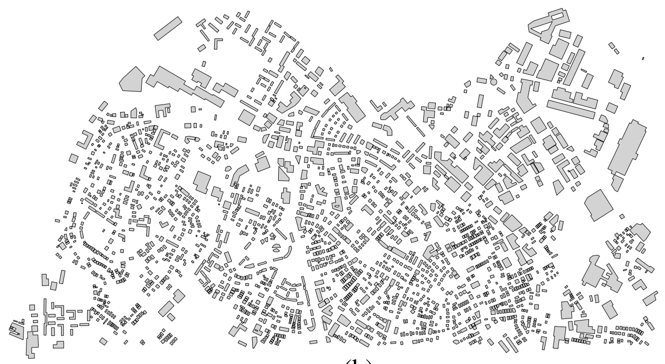


Figure 8.12 Overview of the Boston data set. North End (blue) and the northern part of Roxbury (orange) are indicated.


(a)


(b)

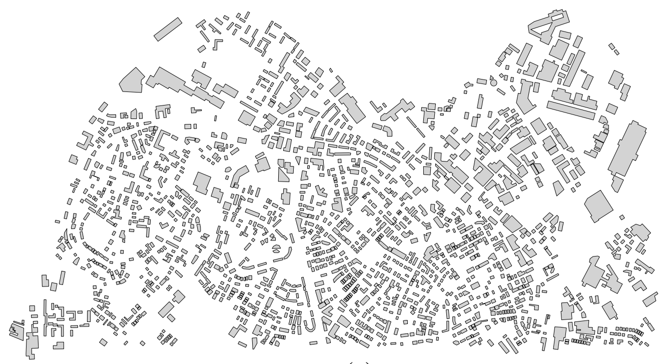

(c)

Figure 8.13 (a) Part of Roxbury, Boston (2 192 components and 15 192 edges). (b) Simplified to 9 965 edges. (c) Generalized to 1 593 components and 8 071 edges.

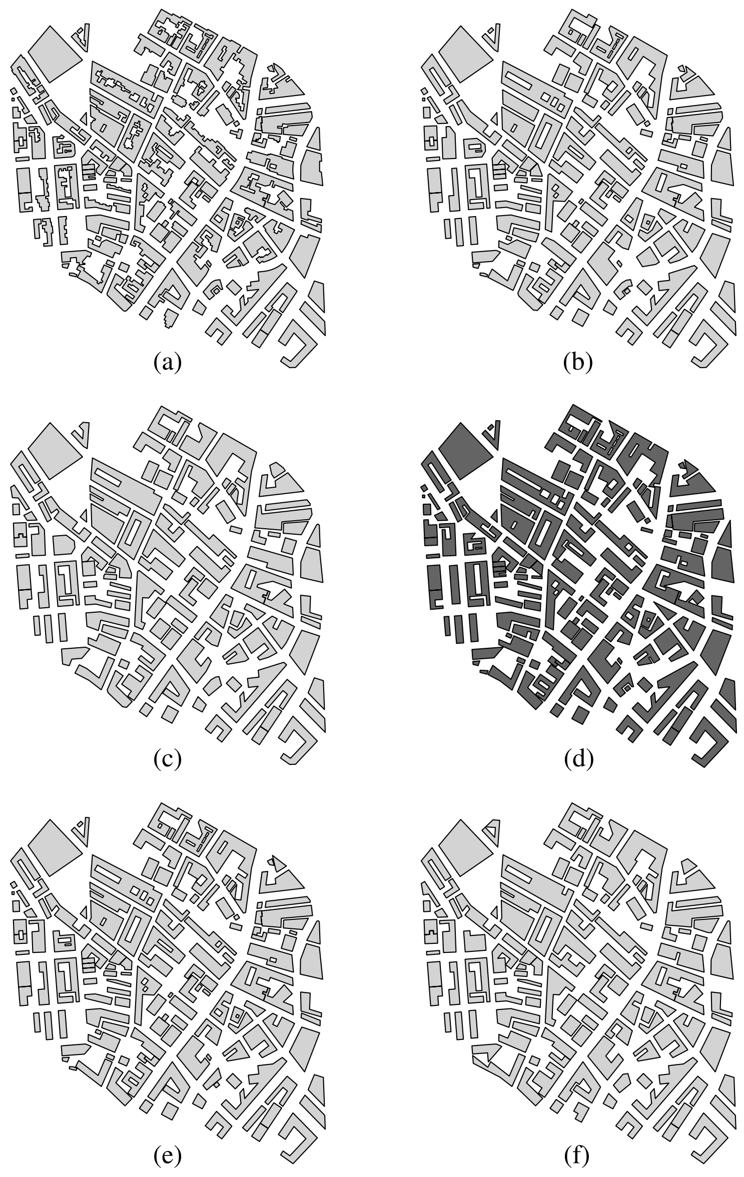


Figure 8.14 (a) North End, Boston (131 components and 2 035 edges). (b) Simplified to 939 edges. (c) Generalized to 242 components and 783 edges. (d) Result of Haunert and Wolff with 932 edges [103, Figure 10]. (e) Simplified to 939 edges allowing “invalid” edge-moves. (f) Generalized to 242 components and 783 edges allowing “invalid” edge-moves.

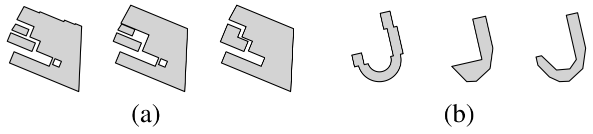


Figure 8.15 Highlights of Roxbury, each depicting input, simplification, and generalization. (a) Using aggregation, these buildings are represented with less edges without a significant loss of visual quality at small scale. (b) Aggregation elsewhere results in a higher visual quality at the same overall complexity level.

operators that were described in this section. However, the generalization uses about 20% less edges while maintaining the overall appearance. Thus, the additional effort seems worthwhile to allow for a higher reduction in complexity. When we take a closer look, we may notice some differences. Some of these differences are highlighted in Figure 8.15. Small buildings are merged when generalizing the urban area. This has little impact on the visual quality but greatly reduces the complexity. As a result, other buildings can even use some more edges: these buildings then retain more of their original shape. Also for the results of North End (Figure 8.14(b–c)), we conclude that the generalized result is at least comparable to the simplified result, while using almost 17% less edges.

The Boston data set has been used to analyze other simplification methods, such as the ILP method presented by Haunert and Wolff [103]. One of their results for North End is shown in Figure 8.14(d). Our simplification result has a similar complexity as their result. The visual quality of the results are comparable. We highlight typical differences by the example shown in Figure 8.16. Individually, the buildings shown are simplified more accurately by the method of Haunert and Wolff, though locally more edges are used. However, the region as a whole can be considered to be simplified better by our method as it preserves the impression of a street.

We observe a difference in what is considered topologically safe. The method of Haunert and Wolff [103] guarantees that there are no intersections, but does not account

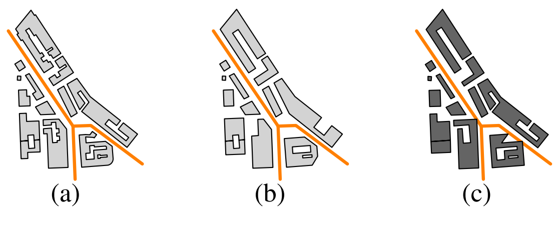


Figure 8.16 Highlight of North End. (a) Buildings give the impression of streets (orange). (b) In our result, the impression is preserved. (c) In the result of Haunert and Wolff [103], this impression is reduced. Note the apparent change in topology in the bottomleft building.

for shared walls and holes (e.g. courtyards). Hence, some apparent holes may vanish (see Figure 8.16) or adjacent buildings may disconnect. In contrast, our method works on subdivisions and—without aggregation—preserves the exact topology. This causes the complexity measures given by Haunert and Wolff [103, Table 1] to differ slightly from those we give. Their result, treated as a subdivision, has 932 edges. As our method preserves the courtyard (Figure 8.16), it is forced to locally use 7 more edges. Hence, we may consider the comparison with our results of 939 edges to be at equal complexity.

The main advantage of our methods is that they are simple, fast, and more scalable. In the worst-case scenario, the ILP formulation by Haunert and Wolff [103] has $O(n^6)$ constraints, thus producing quite large ILPs. They overcome such problems by using lazy constraints. Our methods have a worst-case complexity of $O(n^3)$ when allowing infeasible configurations to compensate for area change. Since a component is affected only by "nearby" components, the execution time in practice behaves significantly better. Table 8.1 shows the execution time on North End, Roxbury and the complete data set in comparison to the times reported by Haunert and Wolff [103]. Our methods were implemented in Java and run on a DELL Precision M4500 Laptop with 4GB RAM and an Intel Core i5 M560 processor with two cores of 2.67GHz each (two threads per core). However, our algorithm runs in only a single thread. The algorithms are approximately as fast for small data sets, but our methods are far more scalable to large data sets.

Another advantage of our simplification method (Algorithm 7.1) is that it preserves the area of each face exactly. While Haunert and Wolff use a term to (partially) optimize for the area change, no guarantees can be given that the area of a face is maintained precisely. Furthermore, our method integrates with a simple aggregation method, while this has to be done separately for the method of Haunert and Wolff.

Our method, as described, cannot eliminate short convex and reflex edges easily, as these are not allowed to move outward or inward respectively. The effect is that round corners are moved inward completely or that small cutoffs are not removed (as illustrated in Figure 8.17). This is caused by the definition of valid edge-moves: only edge-moves of which at least one of the vertices remains on the original outer edge are allowed. However, allowing "invalid" edge-moves is possible and causes no algorithmic problems. Since these edge-moves are not required for completeness, we may restrict their use without interfering with this property of the method. We allow an invalid edge-move with inner edge e only if the intersection of the tracks is in the direction of the edge-move and the

[3]Personal communication with J.-H. Haunert, May 2011.

Table 8.1 Execution times, measured until no further operations can be performed. Last column lists execution times as reported by Haunert and Wolff for their ILP method [103].

Data set	**Simplification**	**Generalization**	**ILP** [103]
North End	2.29s	2.95s	2.06s
Roxbury	3.44s	9.04s	44s[3]
Boston	2h17m44s	4h57m07s	N/A

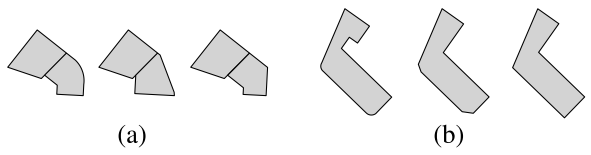


Figure 8.17 Issues that can be solved by allowing "invalid" edge-moves. Both cases depict input, regular result, and result when allowing invalid edge-moves. Examples are found in central and southern North End, respectively. (a) Rounded corners can only shrink. (b) Small cutoffs cannot be removed.

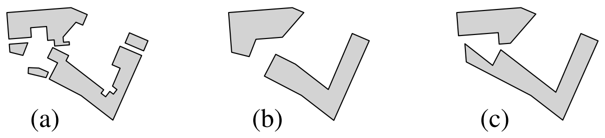


Figure 8.18 A counterintuitive change in which buildings are aggregated in North End, as a result of allowing invalid edge-moves. (a) Input. (b) Regular result. (c) Result when allowing invalid edge-moves.

orthogonal distance between this intersection and the line through e is at most the length of e. Figure 8.14(e–f) shows the result of our methods on North End when allowing these invalid edge-moves. The problems indicated have been alleviated. However, allowing these edge-moves comes at a price. The impression of the street at some locations is reduced similar to the result of Haunert and Wolff (see Figure 8.16). If a street network is available, then such additional constraints can be included in the definition of feasible edge-moves. That is, we may use the road network to block edge-moves. This can also be integrated into the ILP method of Haunert and Wolff. We may also observe a counterintuitive change in which buildings are aggregated, as highlighted in Figure 8.18. Though the narrow distance between the two larger buildings is represented more accurately, the two smaller buildings on the left both aggregate with the same larger building. This is caused by an invalid edge-move decreasing the distance between these, in combination with the aggregation being interleaved with the simplification: it can easily be avoided by performing aggregation beforehand.

An advantage of the algorithm by Haunert and Wolff is that it is parameterized by error tolerance, which is comparatively easily obtained from a target scale. Our method is parameterized by desired complexity, which does not directly correspond to a target scale. However, an advantage of our approach is that all intermediate states can be stored easily; these can be used to find a desired complexity level afterwards. This does require a method to assess which result is considered the best for a target scale. Alternatively, it is straightforward to change the algorithm such that it performs only edge-moves that stay within some error tolerance of the original shape. This requires comparison to the *original* subdivision, rather than to the current subdivision. Otherwise, steps that each incur a small error may accumulate to a large error in total. Depending on the used measure, this leads to an increased worst-case execution time. Such a parametrization to scale would likely solve some issues of components “contending” over edges as indicated in Figure 8.15(b). It also allows for more local processing; this may potentially lead to faster algorithms.

The approach of Haunert and Wolff focuses on modeling the problem with constraints and an optimization criterion. The increased computational complexity is counterbalanced by a greater control on the objective function, in weighting different criteria. The weights of the various operations in our generalization algorithm can be modified in different ways to prefer certain outcomes over others. However, this does not give as strong a control over the result as changing the objective function of the ILP.

8.3 Conclusions

In this chapter we showed that our algorithms for the simplification and schematization of territorial outlines can also be applied in a different context. In particular, we considered building wall squaring and urban-area generalization. For wall squaring, we showed how to compute a suitable set of orientations and the required output complexity for the squared outline. Moreover, we showed a simple technique to avoid arced walls from being distorted. However, when further reducing the complexity of a building, such arced walls must either be fully preserved or “released” to be simplified using straight lines. After detecting arced walls, we may consider fitting smooth curves (e.g. circular arcs or Bézier curves) to these parts of the outline. An interesting question is how such elements would integrate with the edge-moves used in simplification. Recently, iterative methods that use circular arcs have been developed [68, 92]. However, these operations are designed to obtain a fully curved representation. An interesting question is whether we can develop operations for partially curved outlines that carefully integrate with edge-moves in order to obtain a representation that uses a mix of line segments and smooth curves.

For urban-area generalization, we introduced operations to aggregate and eliminate buildings. These operations interleave with the edge-moves applied by the simplification algorithm. Though we showed that this can be done in an area-preserving and orientation-preserving manner, such properties are of less value in this context. Instead, we ensured that aggregation added area to compensate for a potentially perceived area loss. The resulting method is comparatively simple and fast, even for large data sets. However, for data sets that far exceed the main-memory limits of a device, we could consider IO-efficient algorithms. An interesting question is whether our algorithms are suitable for such IO-efficiency techniques. Arge *et al.* [13] describe a very local algorithm for IO-efficient simplification. Our prioritization of operations causes changes to happen “simultaneously” all over the map. We could consider applying operations on a per-component or per-border basis. The question is how this affects the quality of the result. Moreover, Arge *et al.* assume that each face fits in main memory. For subdivisions that represent territorial outlines, this may be a suitable assumption. However, in the context of buildings, the unbounded face is incident to a large majority of edges—essentially the entire subdivision. Therefore, this assumption does not hold.

Chapter 9

Schematization with Other Geometric Styles

In Chapter 6 and Chapter 7, we explored $\mathcal{C}$-oriented schematization and simplification. However, schematic maps come in a variety of geometric styles (see Section 1.1.1). One advantage of our map-matching approach proposed in Chapter 6 is that different geometric styles can be accommodated rather easily: it requires only a different graph. In this chapter we briefly consider three geometric styles in the context of our map-matching technique. We first consider two known styles: parallelism and curved schematization. Then we introduce a new geometric style: *isothetic schematization*. To the best of our knowledge, this style has not been considered for schematization before. For a review of related work on schematization in various geometric styles, we refer to Chapter 7.

9.1 Parallelism

The geometric style of parallelism was introduced by Reimer and Meulemans [150]. This style states that lines should be parallel when possible, but there are no prescribed orientations. Reimer and Meulemans provide a simulated-annealing approach to find a schematic outline with a high parallelism from a simplified outline. In particular, this approach also promotes the use of line segments that "share" perpendicular lines.

Our map-matching formulation can also simulate parallelism. To this end, we construct a graph with a high number of orientations. Each edge in this graph uses one of these orientations. Let k denote the number of bends in a cycle in the graph and c the number of used orientations. We find the cycle that optimizes $k + \lambda c$ for some constant $\lambda \geqslant 0$. That is, the value of λ controls how additional orientations are weighted in comparison to additional bends. If we use $\lambda \leqslant 1/n$ where n is the number of vertices in the graph, this effectively means that the number of bends is minimized. The optimal solution is the one that minimizes the number of orientations over the solutions that use the minimum number of bends. For $\lambda \geqslant n$, the exact opposite holds: the number of orientations is minimized and the number of bends is used as secondary criterion.

For our brute-force method, we can simply keep track of the number of orientations that are used. This can then be used to compute the value of $k + \lambda c$ for a path or cycle in the graph. Since this value can only increase by completing a path to a cycle, it can still be used to bound certain branches in the brute-force search. Unfortunately, due to size of the constructed graph, the computation time becomes infeasible.

9.2 Curved schematization

Recently, there has been a surge of research that considers curved schematization (e.g. [79, 91, 92]). This geometric style uses smooth curves to represent the elements of a schematic maps. To create curved schematization with our map-matching approach, we need to change only the construction of the graph and add some postprocessing. To construct the graph, we sample an arrangement of curves and use the resulting plane graph. We keep track of where the edges in the graph originated from in order to determine which combination of edges does not cause a bend at a vertex. After computing the solution in the sampled graph, we restore the curves in the cycle to obtain a curved schematization.

A simple way of constructing a graph for curved schematization is to first define a single center point. For the arrangement, we then use a number of circles centered at this point and lines going radially outward. This leads to a concentric curved schematization. This style has been used to explore transit-map design.[1] For territorial outlines, this seems to work particularly well for countries that have a roughly circular shape, or some natural central point. In Figure 9.1 we illustrate this for the continent of Antarctica. As the central point, we use a position near the south pole. Figure 9.2 illustrates a result for Russia in

[1]See `www.tubemapcentral.com/circles/circles.html` (accessed April 2014). Fink *et al.* [80] present an automated method to compute such transit maps.

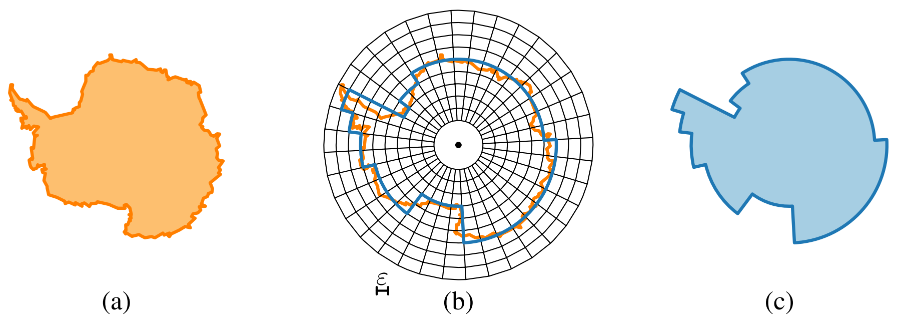


Figure 9.1 A concentric curved schematization of Antarctica. (a) Territorial outline. (b–c) Optimal result with 18 bends using a central point near the south pole.

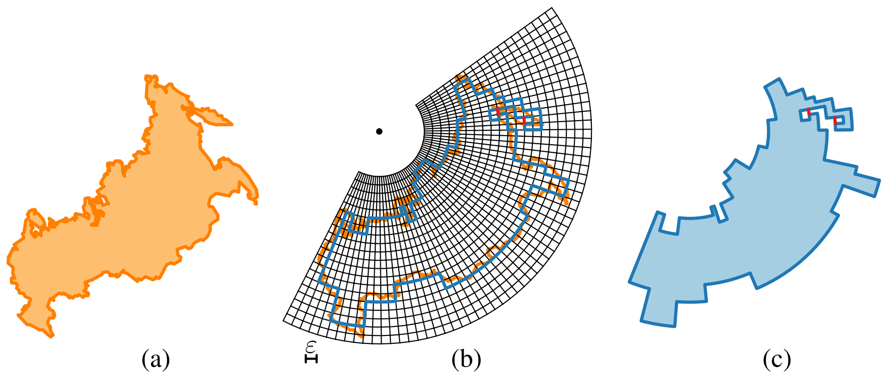


Figure 9.2 A concentric curved schematization of Russia. (a) Territorial outline. (b–c) Suboptimal result with 62 bends using a central point near the north pole. Red lines indicate some changes that could further reduce the number of bends.

orthographic projection. For this result, we used an "external" central point: the north pole. The radial lines roughly represent meridians whereas the circles correspond to the circles of latitude. In both cases, this results in interesting schematic outlines that use circular arcs. However, the density of the concentric graph increases towards the central point. In the case of Russia (Figure 9.2), this causes a large number of cycles within ε distance of the outline, greatly increasing computation time. The given result was found after roughly 2.5 hours. A better result was not found within 24 hours, though at least 58 is attainable (as indicated in red).

9.3 Isothetic schematization

An isothetic polygon uses only edges that span a line passing through one of two *focal points*. It is a generalization of a rectilinear polygon, by considering two focal points at infinity: one left of and one above the polygon. Generalizing this idea, an isothetic graph contains only edges that span a line through one of a small set of focal points. We generate such a graph by computing an arrangement of a discrete set of a lines that originate from two chosen focal points. By using an isothetic graph in our map-matching formulation, we obtain an *isothetic schematization*. Figure 9.3 shows three results that can be obtained this way. In (b–c), we use one focal point that is near the outline to obtain a strong "divergence". In (d–e), we move the focal point further away to decrease this divergence, allowing the schematization to more resemble a rectilinear schematization. An isothetic polygon has two focal points. An isothetic graph may have more focal points, as illustrated in Figure 9.3(f–g). However, similar to our observations for $\mathcal{C}$-oriented schematization in Section 7.4, using more focal points reduces the strength of the geometric style. Note that the computation for this result did not finish in the allotted time (6 hours).

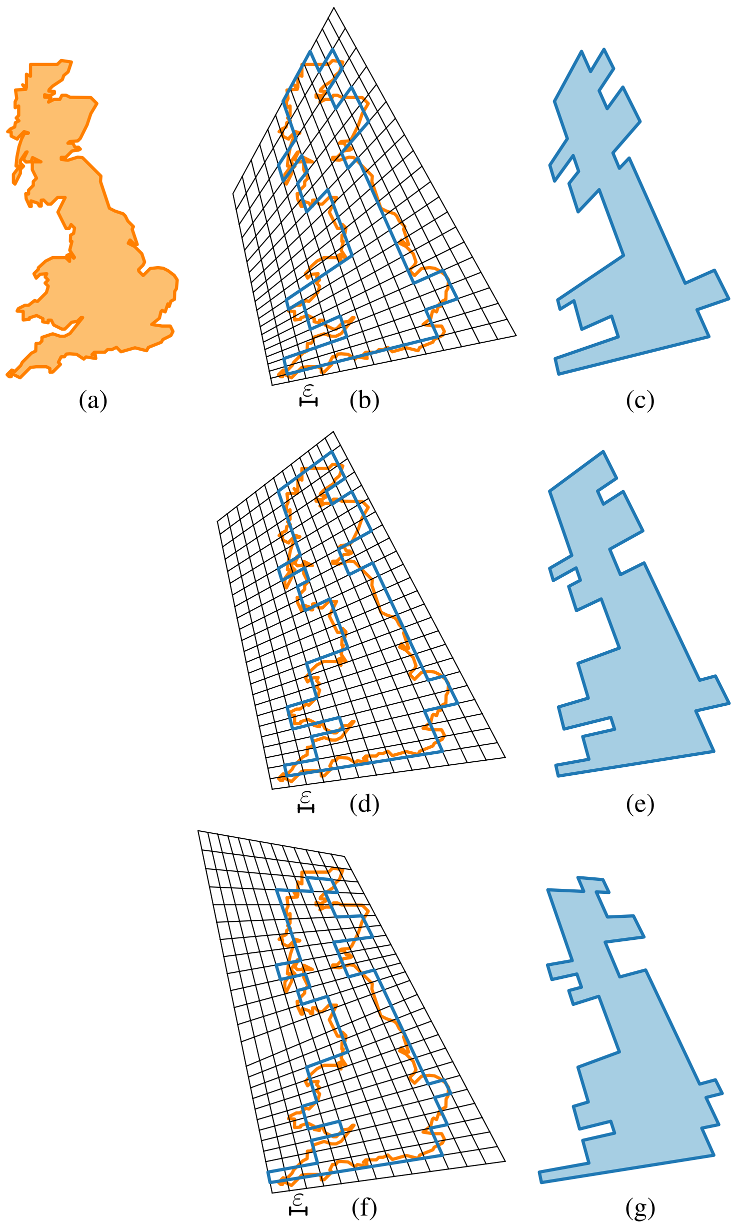


Figure 9.3 An isothetic schematization of Great Britain. (a) Territorial outline. (b–c) Optimal result with 34 bends using a nearby focal point. (d–e) Optimal result with 34 bends using different focal points. (f–g) Suboptimal result with 38 bends using three focal points.

9.4 Conclusion

In this chapter we briefly considered several different geometric styles for schematization in the context of our map-matching approach described in Chapter 6. As we have shown, it is straightforward to adapt this approach for various styles. The results look promising for both isothetic and concentric curved schematization. To the best of our knowledge, isothetic schematizations have not been investigated before. This geometric style may provide an interesting addition to the existing styles. Unfortunately, the current method is too slow to compute a schematic outline using complex graphs, limiting its use for general curved schematization and parallelism. It does indicate, however, that if more efficient algorithms are found for the simple map-matching problem, then we may use such methods not only for $\mathcal{C}$-oriented schematization but also for other geometric styles.

Chapter 10

Conclusion

In this thesis we studied the problem of automated cartographic schematization. A basic characterization of schematic maps distinguishes between different geometric styles and different geographic elements. We focused on the computation of $\mathcal{C}$-oriented schematic representations for territorial outlines such as country and province borders. In this geometric style—the general variant of the popular octilinear style—, every line segment that is used to represent the outline is oriented according to one of a given set $\mathcal{C}$. The quality of a schematic territorial outline depends on its recognizability. Recognizability is related to the geometric similarity between the geographic and schematic outline. Therefore, a key aspect for territorial outlines is the need to capture the shape of the represented regions. In other words, we need similarity measures to formalize schematization as an algorithmic problem and to assess the quality.

We studied algorithmic aspects for both similarity measures and schematization. In particular, we introduced some of the first algorithms that are designed specifically for $\mathcal{C}$-oriented schematization of territorial outlines. Moreover, we provided new algorithms to compute the Fréchet distance and showed how to improve upon the description of similarity that one may obtain from the Fréchet distance. Below, we summarize the main findings and the corresponding open problems. Afterwards, we discuss how this fits into the larger picture of cartographic schematization.

10.1 Main findings

Similarity measures. We considered a variety of similarity measures and concluded that the Fréchet distance is most suitable for schematization. For this measure, we introduced a new algorithm which constitutes the first asymptotic improvement since the results by Alt and Godau [10] in 1995. Our algorithm runs in $O(n^2\sqrt{\log n}(\log\log n)^{3/2})$ time on a pointer machine and in $O(n^2(\log\log n)^2)$ time using a word RAM model. However, our algorithms do not yet match the best known lower bound—assuming the Strong Exponential Time Hypothesis—of $\Omega(n^2)$ (up to subpolynomial factors) [30]. We prove that our algorithm leads to an algebraic decision tree of depth $O(n^{2-\alpha})$ for some $\alpha > 0$. This indicates an interesting discrepancy between these different models of computation for this problem, suggesting a possible relation with other problems that exhibit this discrepancy.

Rather than obtaining simply a number that indicates the similarity, it is often desirable to have a matching that accurately describes the similarity between curves. To this end, we introduced locally correct Fréchet matchings as a first step to assessing the quality of a Fréchet matching. We prove that a locally correct Fréchet matching exists for any two curves and show how one can be computed. This is a strictly stronger concept: any locally correct Fréchet matching is indeed a Fréchet matching. For the discrete variant, we show that a locally correct matching can be computed in roughly the same time as simply computing the discrete Fréchet distance. This local correctness property does not uniquely identify a "best" matching. Further research is required to refine the definition of a good matching. However, the best matching likely depends on the intended application.

Schematization. We formulated the schematization problem using the Fréchet distance as a map-matching problem. To this end, we studied the problem of finding a simple cycle in a plane graph that has a low Fréchet distance to a given simple polygonal curve. We prove that this problem is NP-hard, even in an approximation setting. Using brute-force techniques, we were able to demonstrate its efficacy for $\mathcal{C}$-oriented schematization. Since this formulation allows us to use any plane graph, we also considered its applicability to other geometric styles. We showed that it is also effective for schematization with concentric circular arcs. Moreover, we introduced a new geometric style: isothetic schematization. To formulate schematization as a map-matching problem, we use graphs which are automatically constructed, typically from rather simple arrangements. Therefore, an interesting question is whether the problem is indeed NP-hard for more restricted classes of graphs. In addition, another question warrants attention. By restricting the solutions to a graph, we caused a discretization of the solution space. This has a number of conceptual advantages, such as increasing the flexibility for various geometric styles of schematization. But it also helps in aligning collinear edges of the schematic polygon and may prevent a shape that has parts that are too narrow to see, by avoiding a collapse or even exaggerating the area. However, this discretization also discards many possible solutions. If we consider the continuous solution space—that is, allow *any* $\mathcal{C}$-oriented polygon as a solution—, is the problem still NP-hard?

We presented a simple heuristic method for simplification and $\mathcal{C}$-oriented schematization that iteratively reduces the complexity of a given set of territorial outlines. This algorithm maintains the exact area of the given outlines and is topologically safe. The schematic shapes obtained via this algorithm demonstrate that it maintains the most prominent geographic features of given territorial outlines. Furthermore, we showed that it can also be used in the context of building generalization. Our results suggest that the area-preservation constraint helps in finding good simplified or schematic shapes. This is, however, not a strict criterion for schematization. Can we quantify a tolerance for area distortion? In other words, how much area distortion may we cause with algorithms before this causes conflicts with an observer's mental map? Armed with such knowledge, we may proceed to design algorithms that take this tolerance for area distortion into account.

10.2 Looking forward

The chapters in this thesis describe various methods aimed at schematizing territorial outlines. The solutions in each chapter raise their own set of new questions. Besides these questions, a number of other challenges that involve cartographic schematization are worth mentioning. This includes questions both from a holistic perspective of schematic maps as well as from an atomic perspective that focuses on schematic territorial outlines. Here we consider these questions and suggest potential future research directions for cartographic schematization.

A holistic view. We studied the computation of schematic territorial outlines. However, this rarely constitutes a complete schematic map. To produce real schematic maps, we need a more holistic view that encompasses the entire schematization process.

A central question here is how different types of information integrate into a single map. If we for example consider the construction of a metro map, how do we combine network schematization algorithms with region schematization algorithms? For chorematic diagrams, how do we combine region schematization with the visual representations that illustrate the various processes? What visualization techniques developed in the field of geovisualization can be combined with cartographic schematization? We may consider computing the geometry for the various schematic elements in sequence, taking into consideration any previously computed geometry. For example, we may first compute a schematic outline and, within this outline, compute a schematized network. However, there is often an interplay between the various schematic elements, as the elements of a map should use a consistent spatial arrangement. Hence, the design and placement of the schematic shapes depend on each other. As a result, an approach that considers these in separate steps is limited. That is, better schematic maps can be computed by explicitly modeling the dependencies between the information.

Automated schematization has the potential to produce personalized maps on demand. This requires a system that is able to fully autonomously compute a schematic map based on simple queries. Unless the system is scoped to certain types of maps and data, it needs ways to deduce the map type, necessary data, map extent, level of detail and possibly other auxiliary information that is required to automatically compute a map. The system must then retrieve the data from geographic information systems and process them to produce a schematic map. We have introduced algorithms that can automatically derive schematized outlines from detailed geographic data. However, our current algorithms are not able to (consistently) meet the tight time constraints for on-demand maps. If the retrieved information is highly detailed, then near real-time processing is unlikely to be attainable even if the algorithms have good asymptotic bounds. How can we effectively preprocess information such that any query can be answered efficiently? The design of a schematic map depends on a number of factors, including the desired information, the intent of the map or a preferred geometric style. Because we may expect a large number of factors, it is not feasible to precompute information for all combinations. Therefore, such preprocessed information must still allow for various algorithms that are aimed at obtaining different schematic maps. A basic approach would be to simplify the information beforehand to a complexity level that is still significantly higher than the desired complexity of a schematic map. Is this sufficient to obtain near real-time production of maps? Moreover, how does such a preprocessing step affect the quality of a map? These questions must be answered to move to a system that can produce on-demand schematic maps of high quality.

Another important question is how we may deal with data that is updated over time. Small changes in data should ideally not result in drastic changes in a computed map. Can we ensure that our algorithms produce similar maps on similar data? This correlates to maintaining a stable mental map for a user: drastic changes in a map may cause the user to get lost. Therefore, such stability requirements are surely desirable in a scenario of changing data. How do possible preprocessing schemes, as suggested above, cope with changing data? Especially when data is updated frequently, time-consuming preprocessing strategies are no longer an option.

We have shown that our iterative schematization algorithm is also effective for generalization purposes. Can we find other problems in automated cartography for which schematization algorithms can be applied? We could widen the scope even further, to data of a nongeographic nature. The visual appeal of maps has in fact led to a map metaphor: nongeographic data is visualized in such a way that it visually resembles a cartographic map. Can methods, developed for cartographic schematization or automated cartography in general, contribute to such nonspatial maps? This results in interesting questions of how spatial concepts for schematization translate, via a nonspatial domain, to geometric relations in visualization.

An atomic view. Not just the holistic view on cartographic schematization poses interesting future challenges. Also in the atomic view, many questions can be posed for schematization for territorial outlines. We highlight some of the main open questions.

Schematic maps come in a variety of different geometric styles as well as in a range of different complexity levels. All current algorithms assume that the geometric style as well as the desired complexity (or some related value such as a distance threshold) are manually specified beforehand. To fully automate the production of a schematic map, we need some way of deciding on a style and complexity.

This decision may depend on geometric properties of the geographic shapes: sometimes hexilinear schematization is more suitable than octilinear, or vice versa. What are the geometric properties that lead to such differences? How can we compute these properties beforehand to avoid a trial-and-error approach to schematization? However, there are likely also nongeometric properties that affect the choice for map design. In particular, also the map intent may be an important factor. For example, $\mathcal{C}$-oriented schematization appears very strict and hence might fit an authoritarian map, one that should not leave room to question the map's content. On the other hand, general curved schematization tends to result in a much more liberal appearance, suggesting its use for maps that communicate thoughts and ideas rather than factual information. These are just suggestions of possible relations between map intent and geometric style. Do these relations in fact exist? If such relations are determined, we may exploit them to improve upon known methods or design new algorithms that explicitly take into account the map intent.

The desired complexity level of a schematic outline is likely related to how recognizable the map must be. Can we determine the relation between complexity and recognizability? The latter is affected by the observer's familiarity with the outlined region. The more familiar an observer is with a region, the easier it is to recognize it. Therefore, familiarity may allow for a lower complexity. On the other hand, familiarity may cause reluctance in accepting a schematic map. It may also simply vary per region, depending on how characteristic and unique its shape is. For example, it has been conjectured that 9 vertices are sufficient to obtain a recognizable shape for Australia [136].

This again brings us to recognizability. Throughout this thesis we assumed that recognizability corresponds directly to a quantification of geometric similarity. Though geometric similarity surely plays a significant role, this may not be the only factor involved. Distortions in the mental map of an observer need not be "uniform". This is often the case for certain landmarks or characteristic portions in a geographic shape (e.g. a major estuary or the position of a large city). These locations may be more pronounced or even exaggerated in a mental map, even though they are (scale-wise) relatively insignificant. Can we determine such characteristic features in an automated way? There are a number of methods developed in geographic information science that aim to derive such intangible characteristics. Such methods often use online data that is gathered (sometimes implicitly) by many users of web services. When we have such data available, how do we incorporate these nonuniform requirements in measuring similarity?

Even if we consider a purely geometric approach to similarity, it remains an open problem how to obtain a (computable) formulation that precisely captures the intuitive notion. Though this is unlikely to be possible in general, perhaps this is feasible in the specific context of $\mathcal{C}$-oriented schematization. We studied the Fréchet distance and considered improving the resulting matching to deal with outliers. In some ways, however, this can be seen as curing the symptom rather than the cause. Integral and average Fréchet distances are approaches that target the cause instead, but suffer from other undesirable behavior [44]. Normalization is one of the issues that arise. For schematization, however, we want to compare different schematic outlines to the same geographic outline. In particular, this geographic outline is significantly more detailed than the schematic ones. Therefore, a one-sided definition based on the geographic outline may yield a good measure for schematization purposes. This of course raises new algorithmic questions of how to use such a measure efficiently and effectively. However, to actually measure the efficacy of a new measure, we need to compare its predictions on similarity to what observers intuitively consider similar or recognizable.

On the other hand, we showed in Chapter 7 that using the symmetric difference in a local, incremental way may yield schematic outlines of high quality. The problem that we observed with the symmetric difference (Section 3.2) does not occur due to the local measurements. If we consider the "counterexample" we provided, we may identify the cause of the undesired behavior. Two overlapping regions correspond to conceptually different parts of the polygon. Therefore, this region of overlap should also increase the symmetric difference. Can we formally define a "topological" symmetric difference, a variant that takes such topological differences into account?

Looking forward. Suppose we manage to complement the results in this thesis with answers to the questions outlined above in a positive way. That is, we know exactly what shapes are perfectly recognizable, how to compute these in an instant, how to combine these with other map elements, et cetera. We may then develop a system that can efficiently produce effective schematic maps based on simple queries. This helps us cope with the large amounts of data that are being generated on a daily basis, by enabling visual analysis of the larger structures that lie hidden within.

References

[1] M.A. Abam, S. Daneshpajouh, L. Deleuran, and S. Ehsani. Computing homotopic line simplification. *Computational Geometry: Theory and Applications*, 47(7):728–739, 2014.

[2] P.K. Agarwal, R. Ben Avraham, H. Kaplan, and M. Sharir. Computing the discrete Fréchet distance in subquadratic time. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2013)*, pages 156–167, 2013.

[3] P.K. Agarwal, R. Ben Avraham, H. Kaplan, and M. Sharir. Computing the discrete Fréchet distance in subquadratic time. *SIAM Journal on Computing*, 43(2):429–449, 2014.

[4] P.K. Agarwal, S. Har-Peled, N.H. Mustafa, and Y. Wang. Near-linear time approximation algorithms for curve simplification. *Algorithmica*, 42(3–4):203–219, 2005.

[5] H.K. Ahn, C. Knauer, M. Scherfenberg, L. Schlipf, and A. Vigneron. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry and Applications*, 22(1):27–44, 2012.

[6] N. Ailon and B. Chazelle. Lower bounds for linear degeneracy testing. *Journal of the ACM*, 52(2):157–171, 2005.

[7] H. Alt. The computational geometry of comparing shapes. In *Efficient Algorithms*, LNCS 5760, pages 235–248, 2009.

[8] H. Alt and M. Buchin. Can we compute the similarity between surfaces? *Discrete and Computational Geometry*, 43(1):78–99, 2010.

[9] H. Alt, A. Efrat, G. Rote, and C. Wenk. Matching planar maps. *Journal of Algorithms*, 49:262–283, 2003.

[10] H. Alt and M. Godau. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry and Applications*, 5(1–2):78–99, 1995.

[11] H. Alt, C. Knauer, and C. Wenk. Comparison of distance measures for planar curves. *Algorithmica*, 38(1):45–58, 2003.

[12] J.H. Andrews. What was a map? The lexicographers reply. *Cartographica*, 33(4):1–12, 1996.

[13] L. Arge, J. Truelsen, and J. Yang. Simplifying massive planar subdivisions. In *Proceedings of the 16th Workshop on Algorithm Engineering and Experiments (ALENEX 2014)*, pages 20–30, 2014.

[14] E.M. Arkin, L.P. Chew, D.P. Huttenlocher, K. Kedem, and J.S.B. Mitchell. An efficiently computable metric for comparing polygonal shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(3):209–216, 1991.

[15] B. Aronov, S. Har-Peled, C. Knauer, Y. Wang, and C. Wenk. Fréchet distances for curves, revisited. In *Proceedings of the 14th European Symposium on Algorithms (ESA 2006)*, LNCS 4168, pages 52–63, 2006.

[16] S. Arora and B. Barak. *Computational complexity. A modern approach*. Cambridge University Press, Cambridge, United Kingdom, 2009.

[17] S. Avelar. *Schematic Maps on Demand. Design, Modeling and Visualization*. PhD thesis, Swiss Federal Institute of Technology Zürich, Switzerland, 2002.

[18] M. Bader, M. Barrault, and R. Weibel. Building displacement over a ductile truss. *International Journal of Geographic Information Science*, 19(8–9):915–936, 2005.

[19] I. Baran, E.D. Demaine, and M. Pătraşcu. Subquadratic algorithms for 3SUM. *Algorithmica*, 50(4):584–596, 2008.

[20] T. Barkowsky, L.J. Latecki, and K.F. Richter. Schematizing maps: Simplification of geographic shape by discrete curve evolution. In *Spatial Cognition II*, LNCS 1849, pages 41–53, 2000.

[21] M. Barrault, N. Regnauld, C. Duchêne, K. Haire, C. Baeijs, Y. Demazeau, P. Hardy, W.A. Mackaness, A. Ruas, and R. Weibel. Integrating multi-agent, object-oriented, and algorithmic techniques for improved automated map generalization. In *Proceedings of the 20th International Cartographic Conference (ICC 2001)*, pages 2110–2116, 2001.

[22] R. Bellman and R. Kalaba. On adaptive control processes. *IRE Transactions on Automatic Control*, 4(2):1–9, 1959.

[23] M. de Berg, O. Cheong, M. van Kreveld, and M.H. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, Germany, third edition, 2008.

[24] M. de Berg, A.F. Cook IV, and J. Gudmundsson. Fast Fréchet queries. *Computational Geometry: Theory and Applications*, 46(6):747–755, 2013.

[25] M. de Berg, M. van Kreveld, and S. Schirra. Topologically correct subdivision simplification using the bandwidth criterion. *Cartography and Geographic Information Science*, 25(4):243–257, 1998.

[26] A. Boffet and S.R. Serra. Identification of spatial structures within urban blocks for town characterisation. In *Proceedings of the 20th International Cartographic Conference (ICC 2001)*, pages 1974–1983, 2001.

[27] P. Bose, S. Cabello, O. Cheong, J. Gudmundsson, M. van Kreveld, and B. Speckmann. Area-preserving approximations of polygonal paths. *Journal of Discrete Algorithms*, 4(4):554–566, 2006.

[28] S. Brakatsoulas, D. Pfoser, R. Salas, and C. Wenk. On map-matching vehicle tracking data. In *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB 2005)*, pages 853–854, 2005.

[29] D. Bremner, T.M. Chan, E.D. Demaine, J. Erickson, F. Hurtado, J. Iacono, S. Langerman, M. Pătraşcu, and P. Taslakian. Necklaces, convolutions, and $X+Y$. *Algorithmica*, pages 1–21, 2012.

[30] K. Bringmann. Why walking the dog takes time: Fréchet distance has no strongly subquadratic algorithms unless SETH fails. *Computing Research Repository*, abs/1404.1448, 2014. 20 pages.

[31] M. Bruneau and L. Marcotte. Les états de l'Asie du sud-est continentale. *Mappe-Monde*, 91:4–8, 1991.

[32] K. Buchin, M. Buchin, and J. Gudmundsson. Constrained free space diagrams: a tool for trajectory analysis. *International Journal of Geographic Information Science*, 24(7):1101–1125, 2010.

[33] K. Buchin, M. Buchin, J. Gudmundsson, M. Löffler, and J. Luo. Detecting commuting patterns by clustering subtrajectories. *International Journal of Computational Geometry and Applications*, 21(3):253–282, 2011.

[34] K. Buchin, M. Buchin, C. Knauer, G. Rote, and C. Wenk. How difficult is it to walk the dog? In *Abstracts of the 23rd European Workshop on Computational Geometry (EuroCG 2007)*, pages 170–173, 2007.

[35] K. Buchin, M. Buchin, W. Meulemans, and W. Mulzer. Four Soviets walk the dog—with an application to Alt's conjecture. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2014)*, pages 1399–1413, 2014.

[36] K. Buchin, M. Buchin, W. Meulemans, and B. Speckmann. Locally correct Fréchet matchings. In *Proceedings of the 20th European Symposium on Algorithms (ESA 2012)*, LNCS 7501, pages 229–240, 2012.

[37] K. Buchin, M. Buchin, and A. Schulz. Fréchet distance of surfaces: Some simple hard cases. In *Proceedings of the 18th European Symposium on Algorithms (ESA 2010)*, LNCS 6347, pages 63–74, 2010.

[38] K. Buchin, M. Buchin, R. van Leusden, W. Meulemans, and W. Mulzer. Computing the Fréchet distance with a retractable leash. In *Proceedings of the 21st European Symposium on Algorithms (ESA 2013)*, LNCS 8125, pages 241–252, 2013.

[39] K. Buchin, M. Buchin, and Y. Wang. Exact algorithms for partial curve matching via the Fréchet distance. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2009)*, pages 645–654, 2009.

[40] K. Buchin, M. Buchin, and C. Wenk. Computing the Fréchet distance between simple polygons. *Computational Geometry: Theory and Applications*, 41(1–2):2–20, 2008.

[41] K. Buchin, V. Kusters, B. Speckmann, F. Staals, and B. Vasilescu. A splitting line model for directional relations. In *Proceedings of the 19th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems (ACM GIS 2011)*, pages 142–151, 2011.

[42] K. Buchin, W. Meulemans, and B. Speckmann. A new method for subdivision simplification with applications to urban-area generalization. In *Proceedings of the 19th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems (ACM GIS 2011)*, pages 261–270, 2011.

[43] K. Buchin and W. Mulzer. Delaunay triangulations in $O(\mathrm{sort}(n))$ time and more. *Journal of the ACM*, 58(2):Art. 6, 2011. 27 pages.

[44] M. Buchin. *On the Computability of the Fréchet Distance Between Triangulated Surfaces*. PhD thesis, Freie Universität Berlin, Germany, 2007.

[45] M. Buchin, A. Driemel, and B. Speckmann. Computing the Fréchet distance with shortcuts is NP-hard. *Computing Research Repository*, abs/1307.2097, 2013. 28 pages.

[46] D. Burghardt and A. Cecconia. Mesh simplification for building typification. *International Journal of Geographic Information Science*, 21(3):283–298, 2007.

[47] S. Cabello, M. de Berg, and M. van Kreveld. Schematization of networks. *Computational Geometry: Theory and Applications*, 30(3):223–238, 2005.

[48] E.W. Chambers, É.C. de Verdière, J. Erickson, S. Lazard, F. Lazarus, and S. Thite. Homotopic Fréchet distance between curves or, walking your dog in the woods in polynomial time. *Computational Geometry: Theory and Applications*, 43(3):295–311, 2010.

[49] T.M. Chan. A note on maximum indepedent set in rectangle intersection graphs. *Information Processing Letters*, 89:19–23, 2004.

[50] T.M. Chan. All-pairs shortest paths with real weights in $O(n^3/\log n)$ time. *Algorithmica*, 50(2):236–243, 2008.

[51] T.M. Chan. More algorithms for all-pairs shortest paths in weighted graphs. *SIAM Journal on Computing*, 39(5):2075–2089, 2010.

[52] W.S. Chan and F. Chin. Approximation of polygonal curves with minimum number of line segments. In *Proceedings of the 3rd International Symposium on Algorithms and Computation (ISAAC 1992)*, LNCS 650, pages 378–387, 1992.

[53] V. Chandru and M.R. Rao. Integer programming. In *Handbook on Algorithms and Theory of Computation*, pages 32/1–32/45. CRC Press, Boca Raton (FL), United States of America, 1998.

[54] B.M. Chazelle and D.T. Lee. On a circle placement problem. *Computing*, 36(1–2):1–16, 1986.

[55] T. Cheng and Z. Li. Toward quantitative measures for the semantic quality of polygon generalization. *Cartographica*, 41(2):487–499, 2006.

[56] D. de Chiara, V. del Fatto, R. Laurini, M. Sebillo, and G. Vitiello. A chorem-based approach for visually analyzing spatial data. *Journal of Visual Languages & Computing*, 22(3):173–193, 2011.

[57] T. Christ, D. Pálvölgyi, and M. Stojaković. Consistent digital line segments. In *Proceedings of the 26th Annual Symposium on Computational Geometry (SoCG 2010)*, pages 11–18, 2010.

[58] J. Christensen, J. Marks, and S. Shieber. An emperical study of algorithms for point-feature label placement. *ACM Transactions on Graphics*, 13(3):203–232, 1995.

[59] S. Cicerone and M. Cermignani. Fast and simple approach for polygon schematization. In *Proceedings of the 12th International Conference on Computational Science and Its Applications (ICCSA 2012)*, LNCS 7333, pages 267–279, 2012.

[60] A.F. Cook IV, A. Driemel, S. Har-Peled, J. Sherette, and C. Wenk. Computing the Fréchet distance between folded polygons. In *Proceedings of the 12th Symposium on Algorithms and Data Structures (WADS 2011)*, pages 267–278, 2011.

[61] A.F. Cook IV and C. Wenk. Geodesic Fréchet distance inside a simple polygon. *ACM Transactions on Algorithms*, 7(1):Art. 9, 2010.

[62] J. Damen, M. van Kreveld, and B. Spaan. High quality building generalization by extending morphological operators. In *Proceedings of the 11th ICA Workshop on Generalisation and Multiple Represenation*, 2008. 12 pages.

[63] S. Daneshpajouh, M. Ghodsi, and A. Zarei. Computing polygonal path simplification under area measures. *Graphical Models*, 74(5):283–289, 2012.

[64] D. Delling, A. Gemsa, M. Nöllenburg, and T. Pajor. Path schematization for route sketches. In *Proceedings of the 12th Scandinavian Workshop on Algorithm Theory (SWAT 2010)*, LNCS 6139, pages 285–296, 2010.

[65] S. van Dijk, D. Thierens, and M. de Berg. Using genetic algorithms for solving hard problems in GIS. *GeoInformatica*, 6(4):381–413, 2002.

[66] T.C. van Dijk, A. van Goethem, J.H. Haunert, W. Meulemans, and B. Speckmann. Accentuating focus maps via partial schematization. In *Proceedings of the 21st ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems (ACM GIS 2013)*, pages 438–441, 2013.

[67] T.C. van Dijk, A. van Goethem, J.H. Haunert, W. Meulemans, and B. Speckmann. An automated method for circular-arc metro maps. In *Abstracts of the 1st Schematic Mapping Workshop*, 2014. 2 pages.

[68] T.C. van Dijk, A. van Goethem, J.H. Haunert, W. Meulemans, and B. Speckmann. Map schematization with circular arcs. In *Proceedings of the 8th International Conference on Geographic Information Science (GIScience 2014)*, 2014. To appear.

[69] 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*, 10(2):112–122, 1973.

[70] A. Driemel and S. Har-Peled. Jaywalking your dog: Computing the Fréchet distance with shortcuts. *SIAM Journal on Computing*, 42(5):1830–1866, 2013.

[71] A. Driemel, S. Har-Peled, and C. Wenk. Approximating the Fréchet distance for realistic curves in near linear time. In *Proceedings of the 26th Annual Symposium on Computational Geometry (SoCG 2010)*, pages 365–374, 2010.

[72] R. Drysdale, G. Rote, and A. Sturm. Approximation of an open polygonal curve with a minimum number of circular arcs and biarcs. *Computational Geometry: Theory and Applications*, 41(1–2):31–47, 2008.

[73] C. Duchêne, S. Bard, X. Barillot, A. Ruas, J. Trévisan, and F. Holzapfel. Quantitative and qualitative description of building orientation. In *Proceedings of the 5th ICA Workshop on Progress in Automated Map Generalisation*, 2003. 10 pages.

[74] A. Efrat, L.J. Guibas, S. Har-Peled, J.S.B. Mitchell, and T.M. Murali. New similarity measures between polylines with applications to morphing and polygon sweeping. *Discrete and Computational Geometry*, 28(4):535–569, 2002.

[75] T. Eiter and H. Mannila. Computing discrete Fréchet distance. Technical Report CD-TR 94/65, Christian Doppler Laboratory, Austria, 1994. 8 pages.

[76] J. Erickson. Bounds for linear satisfiability problems. *Chicago Journal of Theoretical Computer Science*, 1999:Art. 8, 1999. 28 pages.

[77] R. Estkowski and J.S.B. Mitchell. Simplifying a polygonal subdivision while keeping it simple. In *Proceedings of the 17th Annual Symposium on Computational Geometry (SoCG 2001)*, pages 40–49, 2001.

[78] V. del Fatto. *Visual Summaries of Geographic Databases by Chorems*. PhD thesis, University of Salerna, Italy and INSA de Lyon, France, 2009.

[79] M. Fink, H. Haverkort, M. Nöllenburg, M. Roberts, J. Schuhmann, and A. Wolff. Drawing metro maps using Bézier curves. In *Proceedings of the 20th International Symposium on Graph Drawing (GD 2012)*, LNCS 7704, pages 463–474, 2013.

[80] M. Fink, M. Lechner, and A. Wolff. Concentric metro maps. In *Abstracts of the 1st Schematic Mapping Workshop*, 2014. 2 pages.

[81] M. Formann and F. Wagner. A packing problem with applications to lettering of maps. In *Proceedings of the 7th Annual Symposium on Computational Geometry (SoCG 1991)*, pages 281–288, 1991.

[82] D. Forrest. Causes and consequences of scale change in schematics maps: are users aware and do they care? In *Proceedings of the 1st Schematic Mapping Workshop*, 2014. 7 pages.

[83] M. Fréchet. Sur quelques points du calcul fonctionnel. *Rendiconti del Circolo Matematico di Palermo*, 22(1):1–72, 1906.

[84] M.L. Fredman. How good is the information theory bound in sorting? *Theoretical Computer Science*, 1(4):355–361, 1976.

[85] A. Gajentaan and M.H. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Computational Geometry: Theory and Applications*, 5(3):165–185, 1995.

[86] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York (NY), United States of America, 1979.

[87] K. Garland. *Mr Beck's Underground Map*. Capital Transport Publishing, Harrow, United Kingdom, 1994.

[88] A. Gemsa, M. Nöllenburg, T. Pajor, and I. Rutter. On d-regular schematization of embedded paths. In *Proceedings of the 37th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2011)*, LNCS 6543, pages 260–271, 2011.

[89] M. Godau. Die Fréchet-Metrik für Polygonzüge – Algorithmen zur Abstandsmessung und Approximation. Diplomarbeit, Freie Universität Berlin, Germany, 1991.

[90] M. Godau. *On the Complexity of Measuring the Similarity Between Geometric Objects in Higher Dimensions*. PhD thesis, Freie Universität Berlin, Germany, 1998.

[91] A. van Goethem, W. Meulemans, A.W. Reimer, H. Haverkort, and B. Speckmann. Topologically safe curved schematization. *The Cartographic Journal*, 50(3):276–285, 2013.

[92] A. van Goethem, W. Meulemans, B. Speckmann, and J. Wood. Exploring curved schematization. In *Proceedings of the 7th IEEE Pacific Visualization Symposium (PacificVis 2014)*, 2014. 8 pages.

[93] A. Grønlund and S. Pettie. Threesomes, degenerates, and love triangles. *Computing Research Repository*, abs/1404.0799, 2014. 13 pages.

[94] J. Gudmundsson and T. Wolle. Towards automated football analysis: Algorithms and data structures. In *Proceedings of the 10th Australasian Conference on Mathematics and Computers in Sport*, 2010. 8 pages.

[95] L.J. Guibas, J.E. Hershberger, J.S.B. Mitchell, and J.S. Snoeyink. Approximating polygons and subdivisions with minimum-link paths. *International Journal of Computational Geometry and Applications*, 3:383–415, 1993.

[96] S. Hahmann and D. Burghardt. How much information is geospatially referenced? Networks and cognition. *International Journal of Geographic Information Science*, 27(6):1171–1189, 2013.

[97] S. Har-Peled, A. Nayyeri, M. Salavatipour, and A. Sidiropoulos. How to walk your dog in the mountains with no magic leash. In *Proceedings of the 28th Annual Symposium on Computational Geometry (SoCG 2012)*, pages 121–130, 2012.

[98] S. Har-Peled and B. Raichel. The Fréchet distance revisited and extended. In *Proceedings of the 27th Annual Symposium on Computational Geometry (SoCG 2011)*, pages 448–457, 2011.

[99] S. Har-Peled and B. Raichel. The Fréchet distance revisited and extended. *Computing Research Repository*, abs/1202.5610, 2012. 28 pages.

[100] J.H. Haunert. A symmetry detector for map generalization and urban-space analysis. *ISPRS Journal of Photogrammetry and Remote Sensing*, 74:66–77, 2012.

[101] J.H. Haunert and B. Budig. An algorithm for map matching given incomplete road data. In *Proceedings of the 20th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems (ACM GIS 2012)*, pages 510–513, 2012.

[102] J.H. Haunert and A. Wolff. Area aggregation in map generalisation by mixed-integer programming. *International Journal of Geographic Information Science*, 24(12):1871–1897, 2010.

[103] J.H. Haunert and A. Wolff. Optimal and topologically safe simplification of building footprints. In *Proceedings of the 18th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems (ACM GIS 2010)*, pages 192–201, 2010.

[104] F. Hausdorff. *Grundzüge der Mengenlehre*. Verlag Von Veit & Comp., Leipzig, Germany, 1914.

[105] M. Heimlich and M. Held. Biarc approximation, simplification and smoothing of polygonal curves by means of Voronoi-based tolerance bands. *International Journal of Computational Geometry and Applications*, 18(3):221–250, 2008.

[106] M. Held and J. Eibl. Biarc approximation of polygons within asymmetric tolerance bands. *Computer-Aided Design*, 37(4):357–371, 2005.

[107] I. Heywood, S. Cornelius, and S. Carver. *An Introduction to Geographical Information Systems*. Prentice Hall, Upper Saddle River (NJ), United States of America, fourth edition, 2011.

[108] S.A. Hirsch. An algorithm for automatic name placement around point data. *The American Cartographer*, 9(1):5–17, 1982.

[109] H. Imai and M. Iri. Polygonal approximations of a curve—formulations and algorithms. *Computational Morphology*, pages 71–86, 1988.

[110] P. Indyk. Approximate nearest neighbor algorithms for Fréchet distance via product metrics. In *Proceedings of the 18th Annual Symposium on Computational Geometry (SoCG 2002)*, pages 102–106, 2002.

[111] International Cartographic Association. *A Strategic Plan for the International Cartographic Association 2003–2011*, 2003. 18 pages, available at `www.icaci.org/strategic-plan/`.

[112] M. Katz and M. Sharir. An expander-based approach to geometric optimization. *SIAM Journal on Computing*, 26(5):1384–1408, 1997.

[113] D.E. Knuth and A. Raghunathan. The problem of compatible representatives. *SIAM Journal of Discrete Mathematics*, 5:422–427, 1992.

[114] B. van de Kraats, M. van Kreveld, and M.H. Overmars. Printed circuit board simplification: Simplifying subdivisions in practice. In *Proceedings of the 11th Annual Symposium on Computational Geometry (SoCG 1995)*, pages 430–431, 1995.

[115] M. van Kreveld, T. Strijk, and A. Wolff. Point labeling with sliding labels. *Computational Geometry: Theory and Applications*, 13(1):21–47, 1999.

[116] S. Lamy, A. Ruas, Y. Demazeau, M. Jackson, W.A. Mackaness, and R. Weibel. The application of agents in automated map generalisation. In *Proceedings of the 19th International Cartographic Conference (ICC 1999)*, 1999. 7 pages.

[117] D. Lichtenstein. Planar formulae and their uses. *SIAM Journal of Computing*, 11:329–343, 1982.

[118] A. MacEachren. *How Maps Work: Representation, Visualization, and Design*. Guilford Publishing, New York (NY), United States of America, 1995.

[119] W.A. Mackaness and R.S. Purves. Automated displacement for large numbers of discrete map objects. *Algorithmica*, 30(2):302–311, 2001.

[120] A. Maheshwari, J.R. Sack, K. Shahbaz, and H. Zarrabi-Zadeh. Fréchet distance with speed limits. *Computational Geometry: Theory and Applications*, 44(2):110–120, 2011.

[121] A. Maheshwari, J.R. Sack, K. Shahbaz, and H. Zarrabi-Zadeh. Improved algorithms for partial curve matching. In *Proceedings of the 19th European Symposium on Algorithms (ESA 2011)*, LNCS 6942, pages 518–529, 2011.

[122] G. Maier. Optimal arc spline approximation. *Computer Aided Geometric Design*, 31(5):211–226, 2014.

[123] G. Maier and G. Pisinger. Approximation of a closed polygon with a minimum number of circular arcs and line segments. *Computational Geometry: Theory and Applications*, 46(3):263–275, 2013.

[124] A. Marzal and V. Palazón. Dynamic time warping of cyclic strings for shape matching. In *Proceedings of the 3rd International Conference on Advances in Pattern Recognition (ICAPR 2005)*, LNCS 3687, pages 644–652, 2005.

[125] H. Mayer. Scale-spaces for generalization of 3D buildings. *International Journal of Geographic Information Science*, 19(8):975–997, 2005.

[126] D. Merrick and J. Gudmundsson. Path simplification for metro map layout. In *Proceedings of the 14th International Symposium on Graph Drawing (GD 2006)*, LNCS 4372, pages 258–269, 2007.

[127] W. Meulemans, A. van Renssen, and B. Speckmann. Area-preserving subdivision schematization. In *Proceedings of the 6th International Conference on Geographic Information Science (GIScience 2010)*, LNCS 6292, pages 160–174, 2010.

[128] M. Monmonier. *How to Lie with Maps*. University of Chicago Press, Chicago (IL), United States of America, 1996.

[129] M.E. Munich and P. Perona. Continuous dynamic time warping for translation-invariant curve alignment with applications to signature verification. In *Proceedings of the 7th IEEE International Conference on Computer Vision (ICCV 1999)*, pages 108–115, 1999.

[130] N. Mustafa, E. Koutsofios, S. Krishnan, and S. Venkatasubramanian. Hardware-Assisted View-Dependent Map Simplification. In *Proceedings of the 17th Annual Symposium on Computational Geometry (SoCG 2001)*, pages 50–59, 2001.

[131] P. Newson and J. Krumm. Hidden markov map matching through noise and sparseness. In *Proceedings of the 17th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems (ACM GIS 2009)*, pages 336–343, 2009.

[132] G. Neyer. Line simplification with restricted orientations. In *Proceedings of the 6th Symposium on Algorithms and Data Structures (WADS 1999)*, LNCS 1663, pages 13–24, 1999.

[133] M. Nöllenburg and A. Wolff. Drawing and labeling high-quality metro maps by mixed-integer programming. *IEEE Transactions on Visualization and Computer Graphics*, 17(5):626–641, 2011.

[134] P. van Oosterom. The GAP-tree, an approach to 'on-the-fly' map generalization of an area partitioning. In *GIS and Generalisation: Methodology and Practice*, pages 120–132. Taylor & Francis, London, United Kingdom, 1995.

[135] P. van Oosterom and J. Stoter. 5D data modelling: Full integration of 2D/3D space, time and scale dimensions. In *Proceedings of the 6th International Conference on Geographic Information Science (GIScience 2010)*, LNCS 6292, pages 310–324, 2010.

[136] D. O'Sullivan and D.J. Unwin. *Geographic Information Analysis*. John Wiley & Sons, Inc., Hoboken (NJ), United States of America, 2003.

[137] M. Ovenden. *Metro Maps of the World*. Capital Transport Publishing, Harrow, United Kingdom, second edition, 2005.

[138] M. Pătraşcu. Towards polynomial lower bounds for dynamic problems. In *Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC 2010)*, pages 603–610, 2010.

[139] P.M. van der Poorten and C.B. Jones. Characterisation and generalisation of cartographic lines using delaunay triangulation. *International Journal of Geographic Information Science*, 16(8):773–794, 2002.

[140] J. van Putten and P. van Oosterom. New results with generalised area partitionings. In *Proceedings of the 8th International Symposium on Spatial Data Handling (SDH 1998)*, page 1998, 485495.

[141] M.A. Quddus, W.Y. Ochieng, and R.B. Noland. Current map-matching algorithms for transport applications: State-of-the art and future research directions. *Transportation Research Part C: Emerging Technologies*, 15(5):312–328, 2007.

[142] D. Rainsford and W.A. Mackaness. Template matching in support of generalisation of rural buildings. In *Proceedings of the 10th International Symposium on Spatial Data Handling (SDH 2002)*, pages 137–151, 2002.

[143] U. Ramer. An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing*, 1(3):244–256, 1972.

[144] P. Raposo. Scale-specific automated line simplification by vertex clustering on a hexagonal tessellation. *Cartography and Geographic Information Science*, 40(5):427–443, 2013.

[145] N. Regnauld. *Généralisation du bâti: Structure spatiale de type graphe et représentation cartographique*. PhD thesis, Laboratoire d'Informatique de Marseille, France, 1998.

[146] N. Regnauld. Contextual building typification in automated map generalization. *Algorithmica*, 30(2):37–66, 2007.

[147] N. Regnauld and R.B. McMaster. A synoptic view of generalisation operators. In *Generalisation of geographic information: cartographic modelling and applications*, pages 37–66. Elsevier, Oxford, United Kingdom, 2007.

[148] N. Regnauld and P. Revell. Automatic amalgamation of buildings for producing Ordnance Survey 1:50,000 scale maps. *The Cartographic Journal*, 44(3):239–250, 2007.

[149] A.W. Reimer. Understanding chorematic diagrams: Towards a taxonomy. *The Cartographic Journal*, 47(4):330–350, 2010.

[150] A.W. Reimer and W. Meulemans. Parallelity for chorematic territorial outlines. In *Proceedings of the 14th ICA Workshop on Generalisation and Multiple Representation*, 2011. 12 pages.

[151] M.J. Roberts. *Underground maps unravelled - Explorations in information design*. Self-published, 2012.

[152] G. Rote. Computing the Fréchet distance between piecewise smooth curves. *Computational Geometry: Theory and Applications*, 37(3):162–174, 2007.

[153] G. Rote. Lexicographic Fréchet matchings. In *Abstracts of the 30th European Workshop on Computational Geometry (EuroCG 2014)*, 2014.

[154] A. Ruas. Généralisation d'immeubles. Technical report, Ordnance Survey & Ecole Nationale des Sciences Géographiques, IGN, France, 1988.

[155] A. Ruas. *Modèle de généralisation de données géographiques à base de contraintes et d'autonomie*. PhD thesis, Université de Marne la Vallée, France, 1999.

[156] A. Saalfeld. Topologically consistent line simplification with the Douglas-Peucker algorithm. *Cartography and Geographic Information Science*, 27(1):7–18, 1999.

[157] M. Sester. Generalization based on least squares adjustment. *International Archives of Photogrammetry and Remote Sensing*, 33(B4):931–938, 2000.

[158] M. Sester. Optimization approaches for generalization and data abstraction. *International Journal of Geographic Information Science*, 19(8–9):871–897, 2005.

[159] M. Sharir and P.K. Agarwal. *Davenport-Schinzel Sequences and Their Geometric Applications*. Cambridge University Press, Cambridge, United Kingdom, 1995.

[160] J. Sherette and C. Wenk. Simple curve embedding. *Computing Research Repository*, abs/1303.0821, 2013. 16 pages.

[161] S. Steiniger, P. Taillandier, and R. Weibel. Utilising urban context recognition and machine learning to improve the generalisation of buildings. *International Journal of Geographic Information Science*, 24(2):253–282, 2010.

[162] S. Steiniger and R. Weibel. Relations among map objects in cartographic generalization. *Cartography and Geographic Information Science*, 34(3):175–197, 2007.

[163] J. Stott, P. Rodgers, J. Martínez-Ovando, and S. Walker. Automatic metro map layout using multicriteria optimization. *IEEE Transactions on Visualization and Computer Graphics*, 17(1):101–114, 2011.

[164] J. Swan, S. Anand, M. Ware, and M. Jackson. Automated schematization for web service applications. In *Proceedings of the 7th International Symposium on Web and Wireless Geographic Information Systems (W2GIS 2007)*, LNCS 4857, pages 216–226, 2007.

[165] M. Thorup. Randomized sorting in $O(n \log\log n)$ time and linear space using addition, shift, and bit-wise boolean operations. *Journal of Algorithms*, 42(2):205–230, 2002.

[166] D. Tutić and M. Lapaine. Area preserving cartographic line generalization. *Cartography and Geoinformation*, 8(11):84–100, 2009.

[167] B. Tversky. Distortions in memory for maps. *Cognitive psychology*, 13(3):407–433, 1981.

[168] B. Tversky. Cognitive maps, cognitive collages, and spatial mental models. In *Spatial Information Theory: A Theoretical Basis for GIS*, LNCS 716, pages 14–24, 1993.

[169] V. Vazirani. *Approximation Algorithms*. Springer-Verlag, Berlin, Germany, 2001.

[170] K. Verbeek. *Algorithms for Cartographic Visualization*. PhD thesis, Technische Universiteit Eindhoven, The Netherlands, 2012.

[171] M. Vermeij, P. van Oosterom, W. Quak, and T. Tijssen. Storing and using scale-less topological data efficiently in a client-server DBMS environment. In *Proceedings of the 7th International Conference on GeoComputation (GeoComputation 2003)*, 2003. 10 pages.

[172] M. Visvalingam and J.D. Whyatt. Line generalisation by repeated elimination of points. *The Cartographic Journal*, 30(1):46–51, 1993.

[173] P. Vujaković. The state as a ‘power container’: The role of news media cartography in contemporary geopolitical discourse. *The Cartographic Journal*, 51(1):11–24, 2013.

[174] F. Wagner and A. Wolff. A practical map labeling algorithm. *Computational Geometry: Theory and Applications*, 7(5–6):387–404, 1997.

[175] J.M. Ware, C.B. Jones, and G.L. Bundy. A triangulated spatial model for cartographic generalisation of areal objects. In *Proceedings of the 2nd International Conference On Spatial Information Theory (COSIT 1995)*, LNCS 988, pages 173–192, 1995.

[176] C. Wenk, R. Salas, and D. Pfoser. Addressing the need for map-matching speed: Localizing global curve-matching algorithms. In *Proceedings of the 18th International Conference on Scientific and Statistical Database Management (SSDBM 2006)*, pages 379–388, 2006.

[177] A. Wolff. Drawing subway maps: A survey. *Informatik—Forschung und Entwicklung*, 22(1):23–44, 2007.

[178] J. Wolf and R.A. Flather. Modelling waves and surges during the 1953 storm. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 363(1831):1359–1375, 2005.

[179] J.K. Wright. Map makers are human: Comments on the subjective in maps. *The Geographical Review*, 32(4):527–544, 1942.

[180] T.R. Wylie. *The Discrete Fréchet Distance with Applications*. PhD thesis, Montana State University, United States of America, 2013.

[181] T.R. Wylie and B. Zhu. A polynomial time solution for protein chain pair simplification under the discrete Fréchet distance. In *Proceedings of the 8th International Symposium on Bioinformatics Research and Applications (ISBRA 2012)*, LNBI 7292, pages 287–298, 2012.

[182] H. Yan, R. Weibel, and B. Yang. A multi-parameter approach to automated building grouping and generalization. *GeoInformatica*, 12(1):73–89, 2008.

[183] S. Zoraster. Integer programming applied to the map label placement problem. *Cartographica*, 23(3):16–27, 1986.

Summary

Similarity Measures and Algorithms for Cartographic Schematization

A schematic map visualizes information in its geographic context. It does so in a structured and organized manner, by using abstract and stylized shapes to represent the information and its context. The power of a schematic map lies in its emphasis on high-level structures and relations, which are often the important aspects of the information. It does not bother an observer with unnecessary details of geographic reality. Geographic accuracy is relinquished to improve upon the clarity of the map and to reduce the cognitive load required to mentally process the information.

Cartographic schematization is the process involved in making schematic maps. The advent of computers and the availability of digital geographic data gave rise to opportunities to automate the schematization process. This automation provides interesting *algorithmic* challenges which are investigated in this thesis. In particular, we focus on the schematization of territorial outlines such as country or province borders. Such elements are often used to convey information about a region or in conjunction with other visual elements to provide geographic context.

To model schematization as an algorithmic problem, we need to know what constitutes a good schematization of a territorial outline. We study the case in which the schematic map is represented by line segments. In this scenario we identify the following four criteria for a good schematic outline: (i) it uses only a few line segments; (ii) each line segment is oriented according to a specified set of allowed orientations; (iii) it maintains its geographic relations to other regions; (iv) it resembles the geographic outline. The first three criteria are comparatively easy to formalize. The fourth criterion is less straightforward: how do we quantify "resemblance"? To develop effective schematization algorithms, we require a suitable *similarity measure* to quantify resemblance.

In the first part of this thesis, we investigate existing similarity measures. We conclude that the Fréchet distance is most suitable to our problem. To compute this measure, we describe an algorithm that computes the Fréchet distance in $O(n^2\sqrt{\log n}(\log\log n)^{3/2})$ time. This is the first asymptotic improvement on the $O(n^2 \log n)$-time algorithm that was described in 1995. The Fréchet distance results in a single number that indicates the similarity between two curves. In many cases an actual description of the similarity (i.e., a matching) is desired. Many matchings result in the Fréchet distance. However, some matchings describe the similarity more accurately than others. To address this problem, we suggest a new criterion to distinguish between unintuitive and intuitive matchings. We call matchings that adhere to this criterion “locally correct”. We prove that any pair of curves admits a locally correct matching and show how one can be computed.

In the second part of this thesis, we investigate algorithms to schematize territorial outlines. First, we model the schematization problem under the Fréchet distance as another problem referred to as simple map matching. We prove that this leads to an NP-hard problem: it is unlikely that an efficient algorithm exists to compute the optimal schematization. We even prove that it is NP-hard to compute a close-to-optimal schematization. We show how to build an interval graph that can be used to solve the problem without a need for computing the Fréchet distance afterwards. By using a brute-force algorithm in combination with this graph, we demonstrate that instances can be solved nonetheless. The results show that the computed schematic outlines capture the prominent features of the geographic input.

We also present a heuristic iterative algorithm for schematization. First, the outline is changed such that all line segments adhere to the given set of orientations (criterion (ii)). Then, the algorithm reduces the complexity (criterion (i)) step by step, while maintaining geographic relations (criterion (iii)) and the orientations of line segments (criterion (ii)). This complexity reduction is based on greedily performing operations that cause the least change in similarity (criterion (iv)). The algorithm also preserves the exact enclosed area of an outline: this ensures that relative sizes do not change when schematizing multiple outlines simultaneously. The algorithm terminates when the desired level of complexity is reached. We prove that, for a single nonconvex outline, an operation always exists for the algorithm to perform. We conclude this part with an experimental evaluation of the schematization algorithm. The results show that the computed schematic outlines maintain the most salient geographic features.

In the third part of this thesis, we widen the scope of our schematization algorithms. We investigate the potential of our iterative algorithm for two different aspects of building generalization. The first aspect is building-wall squaring: we restore right-angle corners that have been distorted due to inaccuracy in the input data. The second aspect is the generalization of an urban area to make it appropriate for some predefined map scale: this requires the possibility to aggregate and eliminate buildings. We present simple extensions to the schematization algorithm such that it can be adequately applied for these problems in generalization.

Finally, we consider other geometric styles of schematization using our brute-force algorithm for the simple map-matching problem. We show that this approach can be used to provide good schematic outlines with concentric circular arcs. In addition, we suggest an interesting new geometric style: isothetic schematization.

Curriculum Vitae

Wouter Meulemans was born on the 22nd of January, 1987 in Breda, The Netherlands. He finished pre-university secondary education (*cum laude*) with the "Science and Engineering" profile in 2005 at the St.-Oelbert Gymnasium in Oosterhout (NB), The Netherlands. In 2008, he received his Bachelor of Science Degree (*cum laude*) in Computer Science from the Eindhoven University of Technology in Eindhoven, The Netherlands. At the same university, he completed his Master of Science Degree (*cum laude*) in Computer Science in 2010 within the Algorithms group. During his studies, he participated in the Master's Honors Programme. For his Master's thesis, he received the Best Final Project Award 2010 of the Department of Mathematics and Computer Science at the Eindhoven University of Technology. In September 2010, he started his PhD project at the Eindhoven University of Technology of which the results are presented in this dissertation. In the summer of 2012, he did an internship with Microsoft Corporation.

C.M. Gray. *Algorithms for Fat Objects: Decompositions and Applications*. Faculty of Mathematics and Computer Science, TU/e. 2008-19

J.R. Calamé. *Testing Reactive Systems with Data - Enumerative Methods and Constraint Solving*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-20

E. Mumford. *Drawing Graphs for Cartographic Applications*. Faculty of Mathematics and Computer Science, TU/e. 2008-21

E.H. de Graaf. *Mining Semi-structured Data, Theoretical and Experimental Aspects of Pattern Evaluation*. Faculty of Mathematics and Natural Sciences, UL. 2008-22

R. Brijder. *Models of Natural Computation: Gene Assembly and Membrane Systems*. Faculty of Mathematics and Natural Sciences, UL. 2008-23

A. Koprowski. *Termination of Rewriting and Its Certification*. Faculty of Mathematics and Computer Science, TU/e. 2008-24

U. Khadim. *Process Algebras for Hybrid Systems: Comparison and Development*. Faculty of Mathematics and Computer Science, TU/e. 2008-25

J. Markovski. *Real and Stochastic Time in Process Algebras for Performance Evaluation*. Faculty of Mathematics and Computer Science, TU/e. 2008-26

H. Kastenberg. *Graph-Based Software Specification and Verification*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-27

I.R. Buhan. *Cryptographic Keys from Noisy Data Theory and Applications*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-28

R.S. Marin-Perianu. *Wireless Sensor Networks in Motion: Clustering Algorithms for Service Discovery and Provisioning*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-29

M.H.G. Verhoef. *Modeling and Validating Distributed Embedded Real-Time Control Systems*. Faculty of Science, Mathematics and Computer Science, RU. 2009-01

M. de Mol. *Reasoning about Functional Programs: Sparkle, a proof assistant for Clean*. Faculty of Science, Mathematics and Computer Science, RU. 2009-02

M. Lormans. *Managing Requirements Evolution*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-03

M.P.W.J. van Osch. *Automated Model-based Testing of Hybrid Systems*. Faculty of Mathematics and Computer Science, TU/e. 2009-04

H. Sozer. *Architecting Fault-Tolerant Software Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-05

M.J. van Weerdenburg. *Efficient Rewriting Techniques*. Faculty of Mathematics and Computer Science, TU/e. 2009-06

H.H. Hansen. *Coalgebraic Modelling: Applications in Automata Theory and Modal Logic*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2009-07

A. Mesbah. *Analysis and Testing of Ajax-based Single-page Web Applications*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-08