Maastricht University | universiteit hasselt
KNOWLEDGE IN ACTION

2015 | Faculty of Sciences

DOCTORAL DISSERTATION

Qualitative similarity measures and map matching techniques for trajectory data

Doctoral dissertation submitted to obtain the degree of
Doctor of Science: Information Technology, to be defended by

Bart Moelans

Promoter: Prof. Dr Bart Kuijpers

Preface

This Ph.D. thesis contains the result of research undertaken at the Databases and Theoretical Computer Science Research Group of Hasselt University. This research was realized within the framework of three projects: project G.0344.05: *Knowledge Representation and Database Problems for Spatio-Temporal Databases* of Research Foundation Flanders (FWO-Vlaanderen), FET-IST project *Geographic Privacy-aware Knowledge Discovery and Delivery (GeoPKDD)* and FET-ICT project *Mobility, Data Mining, and Privacy (MODAP)*. Certainly, I would have never reached the point of finishing my dissertation without the help and support of others.

My research trajectory have been a challenging trip, with both ups and downs. Fortunately, I was not alone on this road, but accompanied by an extended team of experts, always willing to coach, sponsor, help, and motivate me. For this, I would like to kindly thank them. My most important coach throughout all these years was Bart Kuijpers. I am grateful to have you as my promoter, you have given me the opportunity and freedom to determine the direction of my research and to gain a lot of international experience. You guided me through the world of science and, probably the biggest challenge, thought me a lot about scientific writing.

Many thanks to my fellow researchers within Hasselt University and the three projects. I very much enjoyed our exchange of ideas and thoughts, and the fun moments together. Special thanks go to all other people who have worked with me over the years, in particular the co-authors Vania Bogorny, Luis Otavio Alvares, Walied Othman, Alejandro A. Vaisman and Nico Van de Weghe for many useful discussions, comments and suggestions.

Much gratitude goes also to Kristof Bamps, Kristof Ghys Dries Vangoidsenhoven and Jelle Vanhoof. Students I worked with as part of their master's thesis that helped me with software tools and critical questions to get new ideas for this dissertation.

Many thanks also to the members of my promotion committee and jury for spending time on evaluating my thesis, and for providing useful comments and suggestions.

I also appreciate the support of my (former) colleagues at PXL, CiBLiS and Antwerp University, for encouraging me to finish my PhD.

Thanks and love to my father John and my mother Alice, for their encouragement and support and giving me the opportunity to go to the university in the first place. Undoubtedly, my wife Annika deserves a special word of appreciation for her moral support, for her patience and love.

Contents

Introduction and summary

Polylines arise in Geographical Information Science (GIS) in a multitude of ways. One recent example comes from the collection of moving object data, where trajectories of moving objects (for instance, pedestrians, cars, animals, ...), that carry GPS-equipped devices, are collected in the form of time-space points that are registered at certain (ir)regular moments in time. The spatial trace of this movement is a collection of points in two-dimensional geographical space. There are several methods to extend the trajectory in between the measured sample points, of which linear interpolation is a popular method [GS05]. The resulting curve in the two-dimensional geographical space is a *polyline*.

Although most people use a GPS as a navigational tool, it can also be used for storing the position of moving objects for *(spatio-temporal) data analysis* (we refer to [GP08] for an overview of spatio-temporal data mining and analysis). For instance, we can analyse the routes followed by a person or a group of people and try to discover hidden patterns in this trajectory data. However, a major disadvantage of using GPS obtained coordinates is that they are not always very accurate and will not always match the road followed by, for instance, a car or a pedestrian. Therefore, most GPS devices, used in cars, account for these errors by mapping the measured location to the street that was followed, instead of just displaying the location information received from satellites. The general problem of matching GPS-recorded positions to a road network is called *map matching* and it will be the focus of Part II of this thesis (see below).

Another example of the use of polylines in GIS comes from *shape recognition and retrieval*, which arises in domains, such as computer vision, image analysis and GIS, in general. Here, closed polylines (where the starting point coincides with the end point) or polygons, often occur as the boundary of two-dimensional shapes or regions. Shape recognition and retrieval are central problems in this context.

In examples, such as the above, there are, roughly speaking, two very distinct approaches to deal with polylines, polygonal curves and shapes. On the one hand, there are the *quantitative* approaches and on the other hand there are the *qualitative* approaches. Initially, most research efforts have dealt with

the quantitative methods [Boo86, DM98, KM01, MM92]. Only afterwards, the qualitative approaches have gained more attention, mainly supported by research in cognitive science that provides evidence that qualitative models of shape representation are much more expressive than their quantitative counterpart and reflect better the way in which humans reason about their environment [Ger99]. Polygonal shapes and polygonal curves are very complex spatial phenomena and it is commonly agreed that a qualitative representation of space is more suitable to deal with them [Mea01].

Within the qualitative approaches to describe two-dimensional movement or shapes, two major approaches can be distinguished: the region-based and the boundary-based approach. The *region-based* approach, using concepts such as circularity, orientation with respect to the coordinate axis, can only distinguish between shapes with large dissimilarities [Sch96]. The *boundary-based*, using concepts such as extremes in and changes of curvature of the polyline representing the shape, gives more precise tools to distinguish polylines and polygons. Examples of the boundary-based approaches are found in [Got03, Jun93, KE03, LL00, Ley88, Mea01, Sch96].

The principles behind qualitative approaches to deal with polylines are also related to the field of spatial reasoning. *Spatial reasoning* has as one of its main objectives to present geographic information in a qualitative way to be able to reason about it (see, for example, Chapter 12 in [GP08], also for spatio-temporal reasoning) and it can be seen as the processing of information about a spatial environment that is immediately available to humans (or animals) through direct observation. The reason for using a *qualitative representation* is that the available information is often imprecise, partial and subjective [Fre92]. If we return to the example of trajectory data, we can see that if a person orients her- or himself at a certain location in a city and then moves away from this location, she or he remembers her or his current location by using a mental map that takes the relative turns into account with respect to the original starting point, rather than the precise metric information about her or his trajectory. For such navigational problems, a person will for instance remember: "I left the station and went straight; passing a church to my right; then taking two left turns; ..." This brings us to Part I of this thesis.

Part I: The double-cross description of polylines.

One of the formalisms to qualitatively describe polylines in the plane is given by the *double-cross calculus*. In this method, a *double-cross matrix* captures the relative position of any two line segments in a polyline by describing it with respect to a double cross based on the starting points of these line segments. The double-cross calculus was introduced as a formalism to qualitatively represent a configuration of vectors in the plane [Fre92, ZF96]. For

an overview of the use of constraint calculi in qualitative spatial reasoning, we refer to [RN07]. In the double-cross formalism, the relative position (or orientation) of two (located) vectors is encoded by means of a 4-tuple, whose entries come from the set $\{0, +, -\}$. Such a 4-tuple expresses the relative orientation of two vectors with respect to each other. The double-cross formalism is used, for instance, in the *qualitative trajectory calculus*, which, in turn, has been used to test polyline similarity with applications to query-by-sketch, indexing and classification [KpMdW06].

Two polylines are called *double-cross similar* if their double-cross matrices are identical. Two polylines, that are quite different from a quantitative or metric perspective, may have the same double-cross matrices. The idea is that they follow a similar pattern of relative turns, which reflects how humans visualize and remember movement patterns.

Algebraic and geometric characterizations of double-cross matrices of polylines. In Chapter 3, we provide an algebraic and geometric interpretation of the double-cross matrix of a polyline and of double-cross similarity of polylines. To start with, we give a collection of polynomial constraints (polynomial equalities and inequalities) on the coordinates of the measured points of a polyline (its vertices) that express the information contained in the double-cross matrix of a polyline. This algebraic characterisation can be used to effectively verify double-cross similarity of polylines and to generate double-cross similar polylines by means of tools from algebraic geometry, implemented, for instance, in software packages like MATHEMATICA [Wol]. This algebraic characterization of the double-cross matrix also allows us to prove a number of properties of double-cross matrices. As an example, we mention a high degree of symmetry in the double-cross matrix along its main diagonal.

Next, we turn to a geometrical interpretation of double-cross similarity of two polylines. This geometrical interpretation is based on local geometric information of the polyline in its vertices. This information is called the *local carrier order* and it uses (local) compass directions in the vertices of a polyline to locate the relative position of the other vertices. Our main result in this context says that two polylines are double-cross similar if and only if they have the same local carrier order structure.

From the definition of the double-cross matrix of a polyline it is clear that this matrix remains the same if, for instance, we translate or rotate the polyline in the two-dimensional plane. In a final part of this chapter, we identify the exact set of transformations of the two-dimensional plane that leave double-cross matrices invariant. Our main result states that the largest group of transformations of the plane, that is double-cross invariant consist of the *similarity* transformations of the plane onto itself. This result implies, for instance, that it is sufficient to prove any statement about double-cross

matrices of arbitrary polylines, only for polylines that start in the origin of the two-dimensional plane and have a unit length first line segment.

Algorithms to test double-cross similarity. In Chapter 4, we present an algorithm, based on the double-cross formalism, to test for polyline- (and polygon-) similarity. To determine the degree of similarity between two polylines (not necessary of the same size), the algorithm first computes their "generalized polylines," that consist of almost equally long line segments and that approximate the length of the given polylines within an ε-error margin. In a next step, the algorithm determines the double-cross matrices of the generalized polylines and the difference between these matrices is used as a basis to measure the degree of dissimilarity between the given polylines. We prove the termination of our algorithm and give its sequential time complexity.

This chapter ends with a number of applications. We apply our method to query-by-sketch, indexing of polyline databases, and classification of terrain features and show experimental results for each of these applications.

The double-cross matrix of polylines on a grid. In Chapter 5, we study properties of double-cross matrices of polylines that are situated on a grid in the two-dimensional plane. The grid lines are assumed to be parallel to the standard x- and y-axes of the plane. Polylines on grids may arise from trajectories on Manhattan-like road networks.

We give an effective characterization of what double-cross similarity means for polylines that are drawn on a grid. For a polyline on a grid, we call the vertical and horizontal straight lines through its vertices its *vertical and horizontal carriers* and we call the order in which they appear as we go through the polyline from start to end the V- and H-order of the polyline. We call two polylines *VH-equivalent* if they have the same V-order and the same H-order.

To a polyline on a grid, we also associate a *canonical polyline*, which is VH-equivalent to the original polyline and which has the same double-cross matrix as the original polyline. It turns out that VH-equivalence is the notion that captures the geometric information contained in the double-cross matrix: two polylines have the same matrix if and only if they are VH-equivalent (their last vectors may differ in length, though). We also give an algorithm that on input a double-cross matrix of size N by N, checks in $O(N^2)$ time whether it is realizable by a polyline on a grid. For polylines whose vertices are "snapped" to the grid, the above results can be improved on. Here, once the realizability of a matrix has been checked (in $O(N^2)$ time again), it can be realized in $O(N)$ time.

On the realizability of double-cross matrices. In Chapter 6, we address the problem of the realizability of double-cross (like) matrices. Not every $N \times N$ matrix of 4-tuples from $\{-, 0, +\}$ is the double-cross matrix of a polyline with $N + 1$ vertices. This gives rise to the following decision problem: Given

an $N \times N$ matrix of 4-tuples from $\{-, 0, +\}$, decide whether it is the double-cross matrix of a polyline (with $N + 1$ vertices), and if it is, given an example (or many examples) of a polyline that realizes the matrix.

From Chapter 3, we know a collection of polynomial (in)equalities on the coordinates of the vertices of a polyline, that express the information contained in the double-cross matrix of a polyline. Since first-order logic over the reals (or elementary geometry) is decidable ([Tar51]), it follows that our decision problem is also decidable. However, we are left with the question of its time complexity.

In computational algebraic geometry, the problem can be viewed as a satisfiability problem of a system of quadratic equations in $2(N + 1)$ variables. However, the best known algorithms to solve our problem (including the production of sample points) take exponential time. Our decision problem has many particularities (the polynomials are homogeneous of degree 2; they use few monomials and each of them uses only six variables), nevertheless the problem is known to be NP-hard.

In this chapter, we focus on subclasses of the above decision problem for which we can give polynomial time decision algorithms. A first subclass is obtained by restricting the attention to polylines in which consecutive line segments make angles that are multiples of 90°. For this sub-problem, we give a $O(N^2)$-time decision procedure. Next, we turn our attention to polylines in which consecutive line segments make angles that are multiples of 45°. To solve the more complicated case of 45°-polylines, we introduce the polar-coordinate representation of double-cross matrices. We give two-way translations between the Cartesian- and the polar-coordinate representations. Using polar coordinates, our decision problem can be reduced to a linear programming problem, with algebraic coefficients, however. Also here, we obtain a polynomial time decision procedure. This result has some implications on the convexity of the solution set consisting of all 45°-polylines that realise a matrix.

Part II: Map matching techniques for trajectory data.

A common problem in moving object databases (MOD) is the reconstruction of a trajectory from a trajectory sample. Trajectory samples are automatically collected using location-aware devices. Over the past decade, GPS-based navigation systems are becoming increasingly popular. More than often, object positions obtained using these location-aware devices fall outside a road or street network. Typically, when the position of a car or a pedestrian is monitored using GPS, around ninety-five percent of the recorded time-space points fall outside the actually followed road network. Besides the measurement errors, there are also other problems with real-world data. We think of traffic jams and gaps between measured points, produced by some interference

in the satellite signal (for instance, when moving in tunnels). Thus, matching the user's position to a location on the digital map is required. The general problem of matching GPS positions to a road network is called *map matching.*

Introduction to map matching. We give an almost comprehensive overview of the most important types of existing map matching algorithms in Chapter 7, where we also classify them in the operational way that they can be used (online versus offline; low versus high sampling rate; etc.). In this chapter, we also give an overview of characteristics of data sets.

An uncertainty-based map matching algorithm. Many algorithms have already been proposed to solve the map matching problem [BK98, WBK00], although none of them considers the uncertainty issue that is caused by the lack of information about the moving object's location in between measured locations. Linear interpolation between sample points, which assumes that objects move at constant minimal speed, is a classical solution. A more realistic model is based on the notion of *uncertainty.* An example of such model is the use of *space-time prisms* [Ege03, Häg70, HE02, PJ99, Mil05], that model the unknown, but possible positions of a moving object between sample points by using background information, like a local physical or law imposed speed limitation.

In this part of the thesis, we study the relationship between map matching and uncertainty, and propose an new algorithm that combines weighted k-shortest paths [Yen72] with space-time prisms to obtain a method that is applicable on a wide range of trajectory sample types and that outperforms existing algorithms [GKpM$^+$09]. We apply this algorithm to real-world cases and to computer generated trajectory samples, consisting of trajectory samples with a variety of properties. In some cases, observations are taken at small regular intervals. In other data sets, they are taken at larger and irregular intervals. This corresponds to the classical distinction between low sampling rate and high sampling rate, that is often discussed in the map matching literature. In addition, we compare the results of our new space-time prism and k-shortest path algorithm against a number of existing algorithms, on a variety of trajectory sample data with different characteristics. We show that the incorporation of uncertainty in the map matching process leads to more positive matchings. This positive outcome largely compensates the longer running times which, however, remain within reasonable limits.

The main contribution Chapter 8 is a novel map matching algorithm that uses space-time prisms in combination with a weighted k-shortest paths algorithm. An important component is the computation of the bounding box of the spatial projection of a space-time prism. This bounding box limits the number of candidate road segments in the map matching process.

Experimental evaluation of map matching algorithms.

In Chapter 9, we start by addressing the problem of accurately comparing the performance of map matching algorithms against each other. We give an overview of possible properties of trajectory samples and ways to obtain these properties in data sets are described in Section 9.2.

We also discuss a number of existing methods to measure the accuracy of a map matching algorithm on these different data types [LZZ+09, WBK00] and we propose a novel accuracy measure that does not suffer from the drawbacks of existing methods.

The remainder of Chapter 9 is devoted to experimental results. We present a number of tests of different map matching algorithms on a variety of trajectory sample data sets. The aim is to figure out which type of map matching algorithm works best on a certain type of trajectory samples. We have implemented a number of existing map matching algorithms and compare these algorithms with our own uncertainty-based map matching algorithm implementation. The experimental results indicate that our space-time prisms in combination with weighted k-shortest path algorithm is very robust and outperforms a variety of existing algorithms on very different types of trajectory samples.

In conclusion, we want to remark that the two topics of this thesis (map matching of trajectories and double-cross similarity of polylines) are both part of the broad area of *knowledge discovery in trajectory data*. Map matching can be seen as a preliminary data cleaning step that takes place before actual data mining algorithms are applied to the trajectory data. Indeed, map-matching algorithms "clean" the erroneous GPS-measured data by fitting it to the road network where the movement takes place. For data mining techniques, like clustering, that are applied to this cleaned trajectory data, the study and design of distance or similarity measures on trajectory data is necessary. This is where the double-cross similarity of polylines that are the traces of trajectories fits in the knowledge discovery process. Both quantitative and qualitative distance measures can be used for clustering (the traces of) trajectory data.

1

Definitions and preliminaries on trajectories and polylines

In this chapter, we give the definitions of a *trajectory*, a *trajectory sample*, a *polyline* and an α*-polyline*. We also illustrate these concepts. These definitions are used throughout this thesis.

1.1 Notation

In this thesis, we use the following notational conventions.

Let $\mathbf{N}$ and $\mathbf{R}$ denote the sets of the natural and the real numbers, respectively. For $n \in \mathbf{N}$, $\mathbf{R}^n$ denotes the n-dimensional real space.

Since we consider movement of objects in the two-dimensional real plane, $\mathbf{R}^2$, we denote the *time-space* space, in which we work, by $\mathbf{R} \times \mathbf{R}^2$. Here, the first dimension represents *time* and the latter two dimensions represent *space*. We prefer the notation $\mathbf{R} \times \mathbf{R}^2$ over the more correct $\mathbf{R} \times \mathbf{R} \times \mathbf{R}$ (or $\mathbf{R}^3$) to stress the distinction between time and space.

Typically, we use the characters $t, t_0, t_1, ...$ as variables that range over time points and $x, y, x_0, y_0, x_1, y_1, ...$ as variables that range over spatial coordinates. This means that a tuple (t, x, y) represents a variable time-space point in the space $\mathbf{R} \times \mathbf{R}^2$. Using this convention, we also refer to the time dimension as the t-axis and to the spatial dimensions as the (x, y)-plane.

To stress that some t, x, y-values (or other values) are constants, we use sans serif characters: $\mathsf{t}, \mathsf{x}, \mathsf{y}, \mathsf{t}_0, \mathsf{x}_0, \mathsf{y}_0, \mathsf{t}_1, \mathsf{x}_1, \mathsf{y}_1, \ldots$. So, $(\mathsf{t}, \mathsf{x}, \mathsf{y})$ represents some fixed time-space point in the space $\mathbf{R} \times \mathbf{R}^2$.

For variables that range over points in $\mathbf{R}^2$, we use the characters $p, p_0, p_1, \ldots$ and for constant points in $\mathbf{R}^2$, we use, again, sans serif characters $\mathsf{p}, \mathsf{p}_0, \mathsf{p}_1, \ldots$

1.2 Trajectories and trajectory samples

1.2.1 Trajectories

We now define what we mean by a *trajectory*, its *time domain* and its *trace*.

Definition 1.1. Let I be a closed and bounded interval in $\mathbf{R}$. A *trajectory* T is the graph of a continuous (with respect to t) mapping

$$\alpha : I \to \mathbf{R}^2 : t \mapsto \alpha(t) = (\alpha_x(t), \alpha_y(t)).$$

That is, $T = \{(t, \alpha_x(t), \alpha_y(t)) \in \mathbf{R} \times \mathbf{R}^2 \mid t \in I\}$. The set I is called the *time domain* of T. The *trace* of the trajectory T, denoted $\mathrm{tr}(T)$, is the range of the mapping α. That is,

$$\mathrm{tr}(T) = \{(\alpha_x(t), \alpha_y(t)) \in \mathbf{R}^2 \mid t \in I\}.$$

□

Alternatively, we can view the trace $\mathrm{tr}(T)$ of a trajectory T as the projection of T onto the (x, y)-plane. The time domain of a trajectory T corresponds to the projection of T onto the t-axis. Figure 1.1 gives an illustration of a trajectory, its time domain and its trace.

For the moment, we do not impose conditions on the finite representability of (the mappings α that determine) trajectories. An example of mappings α that are finitely representable is given by the case where the functions $\alpha_x(t)$ and $\alpha_y(t)$ are rational functions of t (in which both the numerator and the denominator are polynomials in t).

An example of this case is the trajectory

$$\left\{ \left(t, \frac{1-t^2}{1+t^2}, \frac{2t}{1+t^2} \right) \in \mathbf{R} \times \mathbf{R}^2 \;\mid\; -1 \leq t \leq 1 \right\},$$

which is illustrated in Figure 1.2. This is the trajectory of a moving object (or point) that moves at non-uniform speed on one half of the unit circle located in the half-plane $x \geq 0$ of the (x, y)-plane. Its time domain is the interval $[-1, 1]$ and its trace is the half circle described by the formula $x^2 + y^2 = 1 \wedge x \geq 0$.

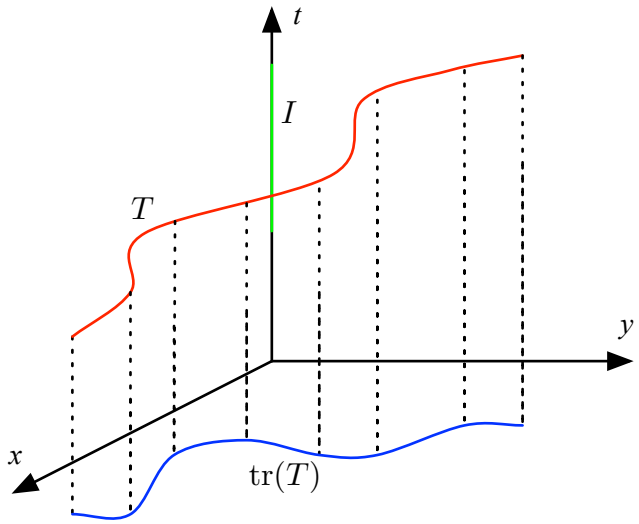


Figure 1.1: A trajectory T (in red) with its time domain I (in green) on the t-axis and its trace $\mathrm{tr}(T)$ (in blue) in the (x, y)-plane.

1.2.2 Trajectory samples

In practice, trajectories of moving objects are only known at discrete moments in time. As an example, we consider the trajectory of a person, measured and collected at certain intervals of time by a GPS-equipped device. This partial knowledge of trajectories is formalized by the notion of a *trajectory sample*. In its definition, we do not consider the uncertainty due to possible errors in the measurement of time and location.

Definition 1.2. A *trajectory sample* S is a finite set $\{(\mathsf{t}_0, \mathsf{x}_0, \mathsf{y}_0), (\mathsf{t}_1, \mathsf{x}_1, \mathsf{y}_1), \ldots, (\mathsf{t}_N, \mathsf{x}_N, \mathsf{y}_N)\}$ (with $N \in \mathbf{N}$) of time-space points of $\mathbf{R} \times \mathbf{R}^2$ with different time-coordinates (that is, $t_i \neq t_j$, for $0 \leq i < j \leq N$). □

If the order on time in S is given by $\mathsf{t}_0 < \mathsf{t}_1 < \cdots < \mathsf{t}_N$, then this order induces a natural order $(\mathsf{t}_0, \mathsf{x}_0, \mathsf{y}_0) < (\mathsf{t}_1, \mathsf{x}_1, \mathsf{y}_1) < \cdots < (\mathsf{t}_N, \mathsf{x}_N, \mathsf{y}_N)$ on the elements of the trajectory sample S.

For practical purposes, we assume that the tuples $(\mathsf{t}_i, \mathsf{x}_i, \mathsf{y}_i)$, for $0 \leq i \leq N$, of a trajectory sample contain rational values.

We now define what it means for a trajectory to be *consistent* with a trajectory sample.

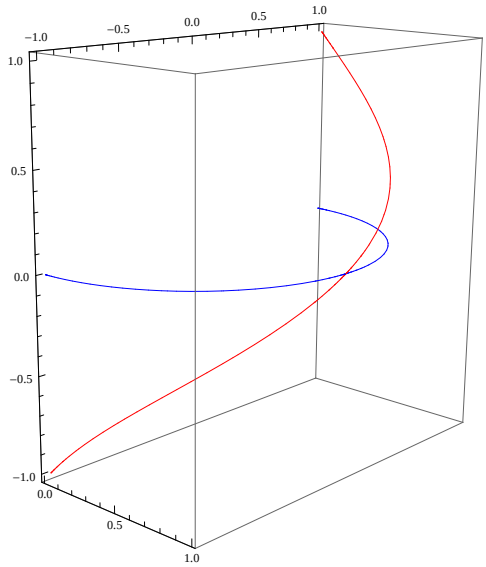


Figure 1.2: A trajectory on a half circle (in red) and its trace (in blue). This figure is produced in Mathematica [Wol] with the command `ParametricPlot3D`$[\{\{(1-t^2)/(1+t^2), 2t/(1+t^2), t\}, \{(1-t^2)/(1+t^2), 2t/(1+t^2), 0\}\}, \{t, -1, 1\}, \texttt{PlotStyle} \to \{\texttt{Red, Blue}\}]$.

Definition 1.3. Let $S = \{(\mathsf{t}_0, \mathsf{x}_0, \mathsf{y}_0), (\mathsf{t}_1, \mathsf{x}_1, \mathsf{y}_1), \ldots, (\mathsf{t}_N, \mathsf{x}_N, \mathsf{y}_N)\}$ be a trajectory sample and let $T = \{(t, \alpha_x(t), \alpha_y(t)) \in \mathbf{R} \times \mathbf{R}^2 \mid t \in I\}$ be a trajectory. We say that the trajectory T is *consistent* with the sample S, if $\mathsf{t}_0, \mathsf{t}_1, \ldots, \mathsf{t}_N \in I$ and $\alpha_x(\mathsf{t}_i) = \mathsf{x}_i$ and $\alpha_y(\mathsf{t}_i) = \mathsf{y}_i$ for $0 \leq i \leq N$. □

This definition explains in an intuitive way what consistency means, but we could express the condition for consistency, in short, by $S \subseteq T$.

1.2.3 The linear interpolation model

A classical and popular model to reconstruct a trajectory from a trajectory sample S is the *linear-interpolation model* (see, for example, Chapter 3 of [GS05]). Here, the unique trajectory, that is consistent with the sample S and that is obtained by assuming that the trajectory is run through at constant lowest speed between any two consecutive (in time) sample points, is constructed.

The following definition describes the linear-interpolation model.

Definition 1.4. For a trajectory sample $S = \{(\mathsf{t}_0, \mathsf{x}_0, \mathsf{y}_0), (\mathsf{t}_1, \mathsf{x}_1, \mathsf{y}_1), \ldots, (\mathsf{t}_N, \mathsf{x}_N, \mathsf{y}_N)\}$, with $\mathsf{t}_0 < \mathsf{t}_1 < \cdots < \mathsf{t}_N$, the trajectory $\mathrm{LIT}(S) :=$

$$\bigcup_{i=0}^{N-1} \left\{ \left(t, \frac{(\mathsf{t}_{i+1} - t)\mathsf{x}_i + (t - \mathsf{t}_i)\mathsf{x}_{i+1}}{\mathsf{t}_{i+1} - \mathsf{t}_i}, \frac{(\mathsf{t}_{i+1} - t)\mathsf{y}_i + (t - \mathsf{t}_i)\mathsf{y}_{i+1}}{\mathsf{t}_{i+1} - \mathsf{t}_i} \right) \mid \mathsf{t}_i \le t \le \mathsf{t}_{i+1} \right\}$$

is called the *linear-interpolation trajectory* of S. □

From the definition, it is clear that $\mathrm{LIT}(S)$ is consistent with S. The time domain of $\mathrm{LIT}(S)$ is the interval $[\mathsf{t}_0, \mathsf{t}_N]$. The functions describing the x- and y-coordinates of $\mathrm{LIT}(S)$ are continuous. They are everywhere differentiable except, possibly, at the moments $\mathsf{t}_0, \mathsf{t}_1, \ldots, \mathsf{t}_N$.

Physically seen, a linear-interpolation trajectory is not realistic. At the moments $\mathsf{t}_0, \mathsf{t}_1, \ldots, \mathsf{t}_{N-1}$ an infinite acceleration may be required to change direction instantly. Nevertheless, linear-interpolation trajectories are sufficient and useful for many practical purposes, especially when the sample points are frequent enough.

Figure 1.3 shows a trajectory sample and its linear-interpolation trajectory with its time domain and trace.

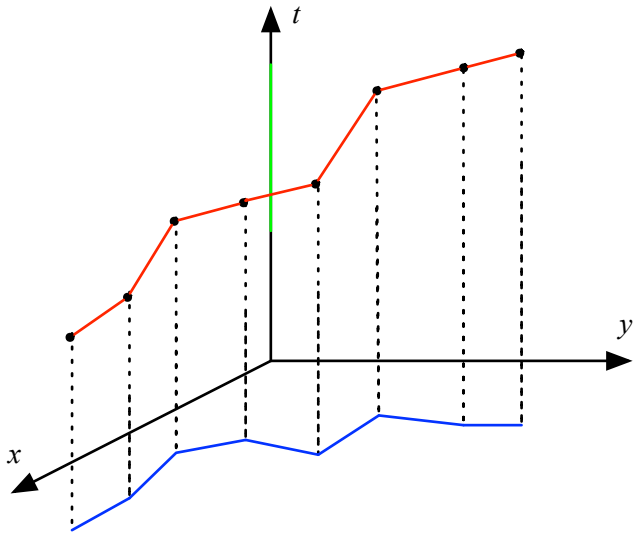


Figure 1.3: A trajectory sample (black dots) and its linear-interpolation trajectory (red line). Also the time domain on the t-axis (in green) and the trace in the (x, y)-plane (in blue) of the linear-interpolation trajectory are shown.

1.3 Polylines and α-polylines

1.3.1 Polylines

The following definitions specifies what *polygonal curves* and *polylines* are and how they are (finitely) represented. We also introduce some terminology about polylines.

Definition 1.5. Let T be a trajectory with time domain $I = [a, b]$. We call T *piecewise affine* if T is the graph of some continuous function $\alpha : [a, b] \to \mathbf{R}^2 : t \mapsto (\alpha_x(t), \alpha_y(t))$ for which there exist "division points" $a = \mathsf{t}_0 < \mathsf{t}_1 < \cdots < \mathsf{t}_N = b$ such that α_x and α_y are affine functions[1] on the intervals $[\mathsf{t}_i, \mathsf{t}_{i+1}]$, for $0 \leq i < N$.

The range of such function is called a *polygonal curve* (in $\mathbf{R}^2$). □

The linear-interpolation trajectory of a trajectory sample (see Definition 1.4) is an example of a trajectory given by piecewise-affine coordinate functions. Its trace, illustrated (in blue) in Figure 1.3, is a polygonal curve.

We remark that the division points in the above definition are not unique. For example, we can add arbitrary division points between any t_i and t_{i+1}, resulting in the same piecewise-affine trajectory and the same polygonal curve.

For the finite representation of polygonal curves, we reserve the name *polyline*.

Definition 1.6. Let $T = \{(t, \alpha_x(t), \alpha_y(t)) \in \mathbf{R} \times \mathbf{R}^2 \mid t \in I\}$ and $a = \mathsf{t}_0 < \mathsf{t}_1 < \cdots < \mathsf{t}_N = b$ be as in Definition 1.5 and let $\alpha(\mathsf{t}_i) = (\mathsf{x}_i, \mathsf{y}_i)$, for $0 \leq i \leq N$.

The ordered list $P = \langle(\mathsf{x}_0, \mathsf{y}_0), (\mathsf{x}_1, \mathsf{y}_1), , \ldots, (\mathsf{x}_N, \mathsf{y}_N)\rangle$ of *vertices* $(\mathsf{x}_i, \mathsf{y}_i)$, $0 \leq i \leq N$, is called a *polyline*. We say that N is the *size* of P. The vertices $(\mathsf{x}_0, \mathsf{y}_0)$ and $(\mathsf{x}_N, \mathsf{y}_N)$ are respectively called the *start* and *end vertex* of P. The line segments connecting the points $(\mathsf{x}_i, \mathsf{y}_i)$ and $(\mathsf{x}_{i+1}, \mathsf{y}_{i+1})$, for $0 \leq i < N$, are called the *(line) segments* of the polyline P. The trace $\mathrm{tr}(T)$ of T is called the semantics of P, denoted $\mathsf{sem}(P)$. □

So, the semantics $\mathsf{sem}(P)$ is the following union of line segments:

$$\bigcup_{i=0}^{N-1} \left\{(x, y) \in \mathbf{R}^2 \mid \exists \lambda \in [0, 1] : (x, y) = \lambda \cdot (\mathsf{x}_i, \mathsf{y}_i) + (1 - \lambda) \cdot (\mathsf{x}_{i+1}, \mathsf{y}_{i+1})\right\},$$

which is a polygonal curve in $\mathbf{R}^2$. Further on, we will loosely use the term polyline also to refer to the semantics of a polyline, although, *stricto sensu*, a polyline is a list of points in $\mathbf{R}^2$.

[1] A function $f : I \to \mathbf{R} : t \mapsto f(t)$ on an interval I of $\mathbf{R}$ is *affine* if it is of the form $f(t) = \mathsf{a}t + \mathsf{b}$, for all $t \in I$, for some constants $\mathsf{a}, \mathsf{b} \in \mathbf{R}$.

We remark that in a polyline $P = \langle(\mathsf{x}_0, \mathsf{y}_0), (\mathsf{x}_1, \mathsf{y}_1), , \ldots, (\mathsf{x}_N, \mathsf{y}_N)\rangle$, which is given as a list of its vertices, time is completely absent (except for the ordering of the vertices). We further remark that a polyline may be the trace of different piecewise-affine trajectories (having different speeds, for instance).

Figure 1.4 gives an example of two polylines with the same semantics, but a different number of vertices.

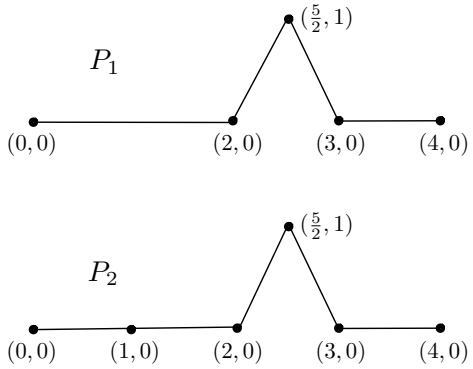


Figure 1.4: An example of a polyline $P_1 = \langle(0,0), (2,0), (\frac{5}{2}, 1), (3,0), (4,0)\rangle$ and a polyline $P_2 = \langle(0,0), (1,0), (2,0), (\frac{5}{2}, 1), (3,0), (4,0)\rangle$ Although they have a different vertex set and a different size, still $\mathsf{sem}(P_1) = \mathsf{sem}(P_2)$.

We also remark that the line segments, appearing in the semantics, may intersect in points which may or may be not vertices, as is illustrated by the polyline shown in Figure 1.5.

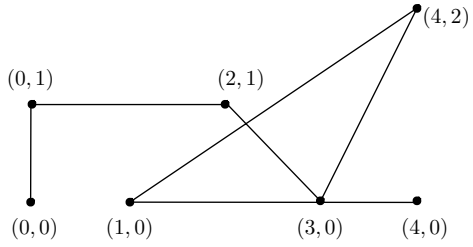


Figure 1.5: An example of a polyline $P = \langle(0,0), (0,1), (2,1), (3,0), (4,2), (1,0), (4,0)\rangle$ and its semantics $\mathsf{sem}(P)$. We see that two of the line segments of its semantics intersect in a point that is not a vertex. The last line segment of the polyline intersects two other line segments in a vertex.

It is reasonable to assume that polylines coming from GIS applications have vertices with rational coordinates. In the algorithms further on, we assume that these rational numbers are stored in some suitable bit representation.

Below, we stick to the notation introduced in the above definitions. Furthermore, as a standard, we abbreviate $(\mathsf{x}_i, \mathsf{y}_i)$ by p_i.

We also use the following notational conventions. The *(located) vector*[2] from p_i to p_j is denoted by $\overrightarrow{\mathsf{p}_i\mathsf{p}_j}$. The counter-clockwise angle (expressed in degrees) measured from $\overrightarrow{\mathsf{p}_i\mathsf{p}_j}$ to $\overrightarrow{\mathsf{p}_i\mathsf{p}_k}$ is denoted by $\angle(\overrightarrow{\mathsf{p}_i\mathsf{p}_j}, \overrightarrow{\mathsf{p}_i\mathsf{p}_k})$, as illustrated in Figure 1.6.

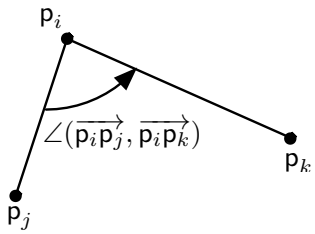


Figure 1.6: The counter-clockwise angle $\angle(\overrightarrow{\mathsf{p}_i\mathsf{p}_j}, \overrightarrow{\mathsf{p}_i\mathsf{p}_k})$ from $\overrightarrow{\mathsf{p}_i\mathsf{p}_j}$ to $\overrightarrow{\mathsf{p}_i\mathsf{p}_k}$.

1.3.2 α-Polylines

In the following chapters, we sometimes consider polylines in which the angles between two consecutive line segments come from a fixed finite set of angles. This is formalized in the following definition.

Definition 1.7. Let α, $0° < \alpha < 360°$, be an angle such that $\frac{360°}{\alpha} = k_\alpha$ is a natural number.

Let $P = \langle(\mathsf{x}_0, \mathsf{y}_0), (\mathsf{x}_1, \mathsf{y}_1), , \ldots, (\mathsf{x}_N, \mathsf{y}_N)\rangle$ be a polyline with $\mathsf{p}_i = (\mathsf{x}_i, \mathsf{y}_i)$, for $0 \leq i \leq N$.

We call P an *α-polyline* if all angles $\angle(\overrightarrow{\mathsf{p}_i\mathsf{p}_{i-1}}, \overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}})$ are multiples of α, for $0 < i < N$ (that is, $\angle(\overrightarrow{\mathsf{p}_i\mathsf{p}_{i-1}}, \overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}})$ is of the form $n_i\alpha$, with $n_i \in \{0, 1, ..., k_\alpha\}$). □

In the previous definition, the case where $n_i = 0$ corresponds to a polyline that, coming from p_{i-1}, reaches p_i, where it returns in the direction of p_{i-1} to reach p_{i+1}.

Figure 1.7 shows the 90°-polyline P_1 and the 45°-polylines P_1 and P_2. Indeed, in the polyline P_1, for instance, the consecutive angles are $90°, 90°, 270°$ and $270°$, assuming that the start vertex is at the left bottom.

[2]By the *located vector* from p to q, we mean an ordered pair (p, q) of points of $\mathbf{R}^2$, which we denote $\overrightarrow{\mathsf{p}\mathsf{q}}$. We use this concept to represent the oriented line segment between p and q.

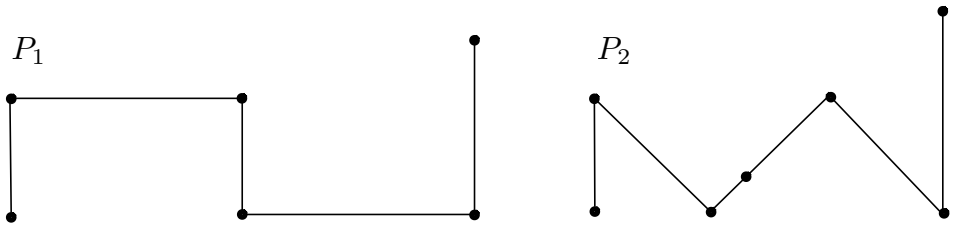


Figure 1.7: An example of a 90°-polyline (P_1) and two 45°-polylines (P_1 and P_2).

Part I

The double-cross description of polylines

2

Background on the double-cross formalism for polylines

In this chapter, we give the historical background of the double-cross formalism and we define the notions of the *double-cross matrix* of a polyline and of *double-cross similarity* of polylines.

2.1 Historical background on the double-cross calculus

Since the main topic of this thesis is qualitative distance measures based on the double-cross calculus, we give, in this section, some historical background on the double-cross formalism. The principles behind the double-cross formalism have emerged from the area of *spatial reasoning*. This field has as one of its main objectives to present geographic information in a qualitative way to be able to reason about it. In the following sections, we briefly discuss some relevant topics from the area of spatial reasoning.

2.1.1 Spatial reasoning

Spatial reasoning can be seen as the processing of information about a spatial environment that is immediately available to humans (or animals) through direct observation. It is this direct information that enables us to orient ourselves

in a spatial environment. This observational information can be transformed into a qualitative representation that allows us to compare environments with each other using some kind of algorithm. The reason for using a *qualitative representation* is that the available information is often imprecise, partial and subjective [Fre92].

For example, if a person orients her- or himself at a certain location in a city and then moves away from this location taking some right and left turns into streets, it is normal that this person remembers her or his current location by using a mental map that takes the relative turns into account with respect to the original starting point, rather than the precise metric information about her or his trajectory. For such navigational problems it is desirable to represent and process the spatial information in an appropriate way, namely rather qualitatively than quantitatively. Qualitative representations of spatial information are discussed in the following sections.

2.1.2 Qualitative representation

In this section, we discuss *qualitative representations* of geographical information. Since such representations aim to represent, in a natural and simple way, possibly inaccurate information, they need to fulfil some conditions [ZF96]:

1. The representation should be simple and extendable.

2. The formalism should allow different levels of granularity, both in the representation (e.g., if only imprecise knowledge is available) and in the choice of operations (e.g., faster computation of partial results should be possible under time constraints).

3. The approach should resemble some fundamental properties known about human spatial reasoning to be plausible from a cognitive point of view.

As will be explained in detail in Section 2.2, the *double-cross calculus* only uses three symbols, namely $-, 0$ and $+$. These symbols represent "away from", "left or right turn from" and "towards" a certain location. Several ways to extend this calculus have been suggested. For example, a method to take speed into account has been suggested [Hum10]. So, in some sense, the double-cross formalism fulfills the first of the above conditions. Also the second condition is fulfilled, since the double-cross approach allows different levels of precision and can handle missing data. The last condition, the resemblance with human reasoning, is a rather subjective one. But in the following sections, we will try to explain the motivations behind the double-cross formalism and map this to the way humans perform spatial reasoning.

We start with a simple example. Assume a person walks from point `a` to point `b`, and during his walk observes a third point `c`. If we represent the road

Figure 2.1: Walking from a to b, with c on the left.

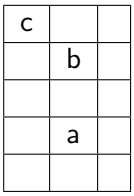


Figure 2.2: Walking from a to b, with c on the left (from [Fre92]).

from a to b as a vector $\overrightarrow{ab}$, it is simple and intuitive human observation to decide whether the point c is positioned to the left, to the right or on the line that contains the vector $\overrightarrow{ab}$. For instance, in Figures 2.1, 2.2 and 2.3, it is immediately clear (in different representations by different authors) that point c is to the left of the line determined by the vector $\overrightarrow{ab}$.

This observation can be extended or refined by drawing lines through a and b perpendicular to the line that contains $\overrightarrow{ab}$. Now the point c can be positioned on 15 different positions according to $\overrightarrow{ab}$. By working with these three lines, we can locate the point c in 15 positions (that are visualized in Figure 2.4) These 15 positions are divided in 6 regions ($a_1, \ldots, a_6$), 6 half lines ($\ell_1, \ldots, \ell_6$), 2 points (p_1 and p_2) and 1 line segment (p_1p_2).

These positions can be represented in several iconic ways. Zimmerman et al. present Figure 2.1 as in Figure 2.3 [ZF96] . For this section, we will use the representation of Freksa, illustrated in Figure 2.2 [Fre92].

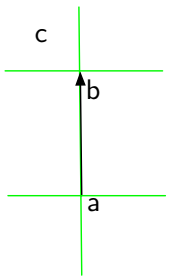


Figure 2.3: Walking from a to b, with c on the left (from [ZF96]).

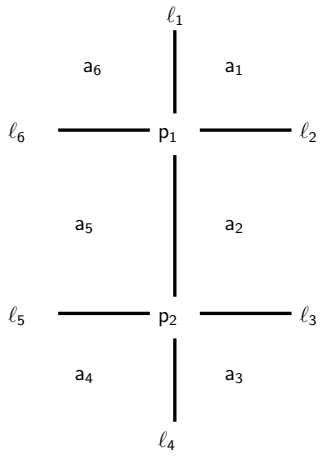


Figure 2.4: 15 possible positions according to a vector from p_1 to p_2.

Besides the fact that this notation is simple, a second advantage is the division of the space by perpendicular lines on a vector. Indeed, for humans it is more intuitive to estimate angle of 90° than angles of other sizes.

In Section 2.1.2.1, some extensions on this spatial representation are discussed (for instance, the composition of a spatial representation given two existing representations).

2.1.2.1 Operations

By applying operations on a spatial representation, several representations can be related to each other. In that way, we can get a more complete and general spatial representation. We give an example to illustrate the advantage of one of these operations.

Suppose a person is moving and passing, respectively, the spatial locations (points) a, b ,c and d. The position of this person observes c with respect to the first vector of his movement, $\overrightarrow{\mathsf{ab}}$, is already discussed in previous section. Similarly, we can look at how this person observes the position of the next point d with respect to her or his next direction of movement (the vector $\overrightarrow{\mathsf{bc}}$). If we now consider the vector $\overrightarrow{\mathsf{ac}}$, it would be nice and useful to predict the relative position of d using the relative position of c with respect to the vector $\overrightarrow{\mathsf{ab}}$ and the relative position of d with respect to the vector $\overrightarrow{\mathsf{bc}}$.

The operation which makes this possible is the *composition*, discussed in the following paragraph. Another operations, that we discuss is the *inversion*. Zimmerman et al. define two more operations: *homing* and *shortcut*, but these are irrelevant for our purpose [ZF96]. In the remainder of this section, we will abbreviate the relative position of c with respect to the vector $\overrightarrow{\mathsf{ab}}$ as $\mathsf{ab} : \mathsf{c}$.

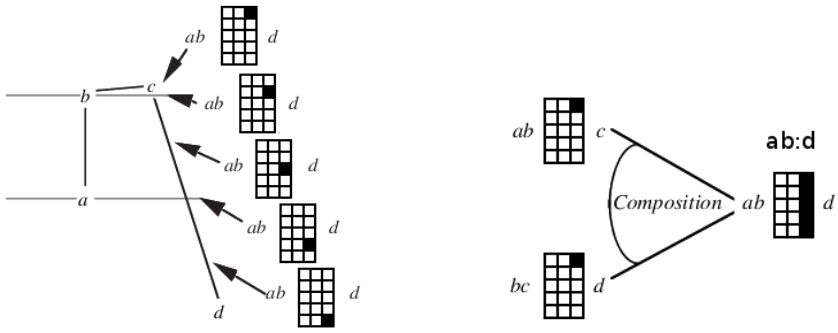


Figure 2.5: Composition (from [ZF96]).

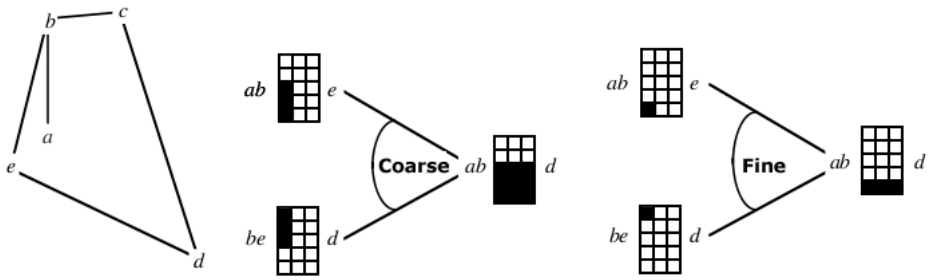


Figure 2.6: The difference between course and fine composition (from [ZF96]).

Composition In Figure 2.5(a) three line segments, ab, bc and cd and four points a, b, c and d are shown. If we transform the segments ab and bc into the vectors $\overrightarrow{\mathsf{ab}}$ and $\overrightarrow{\mathsf{bc}}$, we can, using the observations of the previous sector, represent $\mathsf{ab} : \mathsf{c}$ and $\mathsf{bc} : \mathsf{d}$. Now, we can calculate all possible positions for d according to $\overrightarrow{\mathsf{ab}}$. Because we can not infer anything about the length of line segment cd from $\mathsf{ab} : \mathsf{c}$ and $\mathsf{bc} : \mathsf{d}$, there are in this example five possible positions. If we combine these five positions, we can represent the composition as shown in Figure 2.5(b).

Furthermore, Zimmerman et al. distinguishes between two kind of compositions: *coarse* and *fine* composition [ZF96]. Coarse composition only uses positional information (like in Figure 2.5(b)), whereas fine composition also uses the length of the vectors. The difference between coarse and fine composition is illustrated in Figure 2.6.

Knowledge about this composition is necessary to get insight in realizability (see Chapter 6) of double-cross matrices (defined in Definition 2.3).

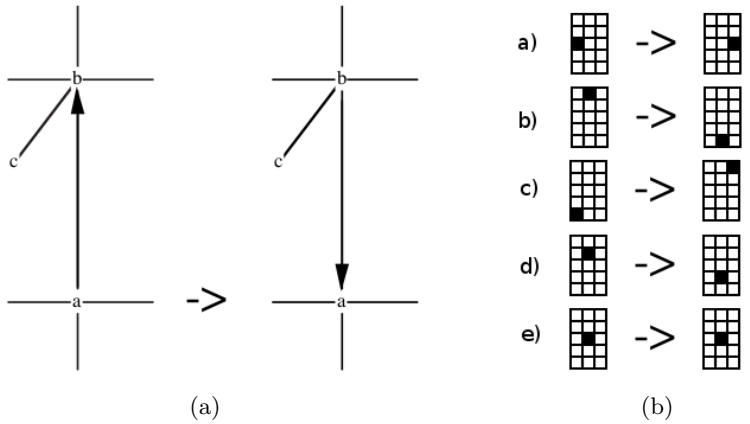


Figure 2.7: Example of inversion operator (from [ZF96]).

Inversion In contrast with composition, inversion is an unary operation. As the names indicates, inversion inverts the orientation of the vector $\overrightarrow{\mathsf{ab}}$ into $\overrightarrow{\mathsf{ba}}$ and the relation $\mathsf{ab} : \mathsf{c}$ to $\mathsf{ba} : \mathsf{c}$. The inversion operator is visualized in Figure 2.7. In Figure 2.7(a), the effect of this operation on $\overrightarrow{\mathsf{ab}}$ is shown. The effect of this inversion of relations is shown in Figure 2.7(b). This inversion operator is more formally discussed in Property 3.3 of Chapter 3.

2.2 The double-cross matrix

In this section, we define the *double-cross matrix* of a polyline.

From now on, we impose the following restriction on polylines (as defined in Chapter 1).

Assumption 1. We assume that no two *consecutive* vertices of a polyline are identical. □

2.2.1 The double-cross value of two (located) vectors

The double-cross calculus was introduced as a formalism to qualitatively represent a configuration of vectors in the plane $\mathbf{R}^2$ [Fre92, ZF96]. In this formalism, the relative position (or orientation) of two (located) vectors is encoded by means of a 4-tuple, whose entries come from the set $\{0, +, -\}$. Such a 4-tuple expresses the relative orientation of two vectors with respect to each other.

We associate to a polyline $P = \langle(\mathsf{x}_0, \mathsf{y}_0), (\mathsf{x}_1, \mathsf{y}_1), , \ldots, (\mathsf{x}_N, \mathsf{y}_N)\rangle$, with $\mathsf{p}_i = (\mathsf{x}_i, \mathsf{y}_i)$, the (located) vectors $\overrightarrow{\mathsf{p}_0\mathsf{p}_1}, \overrightarrow{\mathsf{p}_1\mathsf{p}_2}, \ldots, \overrightarrow{\mathsf{p}_{N-1}\mathsf{p}_N}$, representing the oriented line segments between the consecutive vertices of P. Because of Assumption 1,

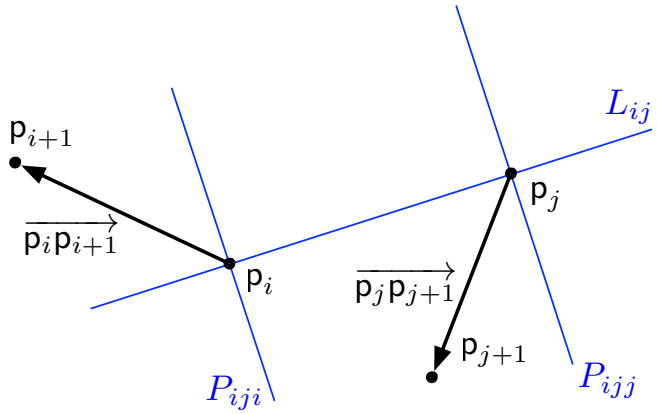


Figure 2.8: The double cross (in blue): the lines L_{ij}, P_{iji} and P_{ijj}.

the vectors $\overrightarrow{\mathsf{p}_0\mathsf{p}_1}, \overrightarrow{\mathsf{p}_1\mathsf{p}_2}, \ldots, \overrightarrow{\mathsf{p}_{N-1}\mathsf{p}_N}$ all have a strictly positive length. In the double-cross formalism, the relative orientation between $\overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}}$ and $\overrightarrow{\mathsf{p}_j\mathsf{p}_{j+1}}$ is given by means of a 4-tuple

$$(C_1\ C_2\ C_3\ C_4) \in \{-, 0, +\}^4.$$

We follow the traditional notation of this 4-tuple *without commas*.

To determine C_1, C_2, C_3 and C_4, for $\mathsf{p}_i \neq \mathsf{p}_j$, first of all, a *double cross* is defined for the vectors $\overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}}$ and $\overrightarrow{\mathsf{p}_j\mathsf{p}_{j+1}}$, determined by the following three lines:

- the line L_{ij} through p_i and p_j;
- the line P_{iji} through p_i, perpendicular on L_{ij}; and
- the line P_{ijj} through p_j, perpendicular on L_{ij}.

These three lines are illustrated in Figure 2.8. These three lines determine a cross at p_i and a cross at p_j. Hence the name "double cross." The entries C_1, C_2, C_3 and C_4 express in which quadrants or on which half lines p_{i+1} and p_{j+1} are located with respect to the double cross.

We now define this more formally and follow the historical use of the double cross (see, for instance, [Fre92, ZF96, dW04, dWKpBM05]). In this definition, an interval (a, b) of angles, represents the open interval between a and b on the counter-clockwise oriented circle.

Definition 2.1. Let $P = \langle(\mathsf{x}_0, \mathsf{y}_0), (\mathsf{x}_1, \mathsf{y}_1), , \ldots, (\mathsf{x}_N, \mathsf{y}_N)\rangle$ be a polyline, with $\mathsf{p}_i = (\mathsf{x}_i, \mathsf{y}_i)$, for $0 \leq i \leq N$, and with associated vectors $\overrightarrow{\mathsf{p}_0\mathsf{p}_1}, \overrightarrow{\mathsf{p}_1\mathsf{p}_2}, \ldots, \overrightarrow{\mathsf{p}_{N-1}\mathsf{p}_N}$.

For $\overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}}$ and $\overrightarrow{\mathsf{p}_j\mathsf{p}_{j+1}}$ with $0 \leq i, j < N$, $i \neq j$ and $\mathsf{p}_i \neq \mathsf{p}_j$, we define

$$\mathsf{DC}(\overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}}, \overrightarrow{\mathsf{p}_j\mathsf{p}_{j+1}}) = (C_1\ C_2\ C_3\ C_4)$$

as follows:

$$C_1 = \begin{cases} - & \text{if} \quad \angle(\overrightarrow{\mathsf{p}_i\mathsf{p}_j}, \overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}}) \in (-90°, 90°) \\ 0 & \text{if} \quad \angle(\overrightarrow{\mathsf{p}_i\mathsf{p}_j}, \overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}}) \in \{-90°, 90°\} \\ + & \text{else} \end{cases}$$

$$C_2 = \begin{cases} - & \text{if} \quad \angle(\overrightarrow{\mathsf{p}_j\mathsf{p}_i}, \overrightarrow{\mathsf{p}_j\mathsf{p}_{j+1}}) \in (-90°, 90°) \\ 0 & \text{if} \quad \angle(\overrightarrow{\mathsf{p}_j\mathsf{p}_i}, \overrightarrow{\mathsf{p}_j\mathsf{p}_{j+1}}) \in \{-90°, 90°\} \\ + & \text{else} \end{cases}$$

$$C_3 = \begin{cases} - & \text{if} \quad \angle(\overrightarrow{\mathsf{p}_i\mathsf{p}_j}, \overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}}) \in (0°, 180°) \\ 0 & \text{if} \quad \angle(\overrightarrow{\mathsf{p}_i\mathsf{p}_j}, \overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}}) \in \{0°, 180°\} \\ + & \text{else} \end{cases}$$

$$C_4 = \begin{cases} - & \text{if} \quad \angle(\overrightarrow{\mathsf{p}_j\mathsf{p}_i}, \overrightarrow{\mathsf{p}_j\mathsf{p}_{j+1}}) \in (0°, 180°) \\ 0 & \text{if} \quad \angle(\overrightarrow{\mathsf{p}_j\mathsf{p}_i}, \overrightarrow{\mathsf{p}_j\mathsf{p}_{j+1}}) \in \{0°, 180°\} \\ + & \text{else.} \end{cases}$$

For $\overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}}$ and $\overrightarrow{\mathsf{p}_j\mathsf{p}_{j+1}}$, with $\mathsf{p}_i = \mathsf{p}_j$, we define, for reasons of continuity,[1]

$$\mathsf{DC}(\overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}}, \overrightarrow{\mathsf{p}_j\mathsf{p}_{j+1}}) = (0\ 0\ 0\ 0).$$

□

So, we have $\mathsf{DC}(\overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}}, \overrightarrow{\mathsf{p}_j\mathsf{p}_{j+1}}) = (0\ 0\ 0\ 0)$ in particular, when $i = j$.

We remark that the values C_1 and C_3 describe the location of the point p_{i+1} or, equivalently, the orientation of the vector $\overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}}$ with respect to the cross at p_i (formed by the lines L_{ij} and P_{iji}). We see that each of the four quadrants and four half lines determined by the cross at p_i are determined by a unique combination of C_1 and C_3 values. Similarly, the values C_2 and C_4 describe the location of the point p_{j+1} or, equivalently, the orientation of the vector $\overrightarrow{\mathsf{p}_j\mathsf{p}_{j+1}}$ with respect to the cross at p_j (formed by the lines L_{ij} and P_{ijj}).

The quadrants and half lines where C_1, C_2, C_3 and C_4 take different values are graphically illustrated in Figure 2.9.

For example, the 4-tuple $\mathsf{DC}(\overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}}, \overrightarrow{\mathsf{p}_j\mathsf{p}_{j+1}})$ for the vectors $\overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}}$ and $\overrightarrow{\mathsf{p}_j\mathsf{p}_{j+1}}$, shown in Figure 2.8, is $(+\ -\ -\ -)$.

Further on, we will sometimes use the notation $\mathsf{DC}(\overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}}, \overrightarrow{\mathsf{p}_j\mathsf{p}_{j+1}})[k]$ to indicate C_k, for $k = 1, 2, 3, 4$. Obviously, this notation does not hide the dependence on i and j.

[1] This argumentation is given in [For90].

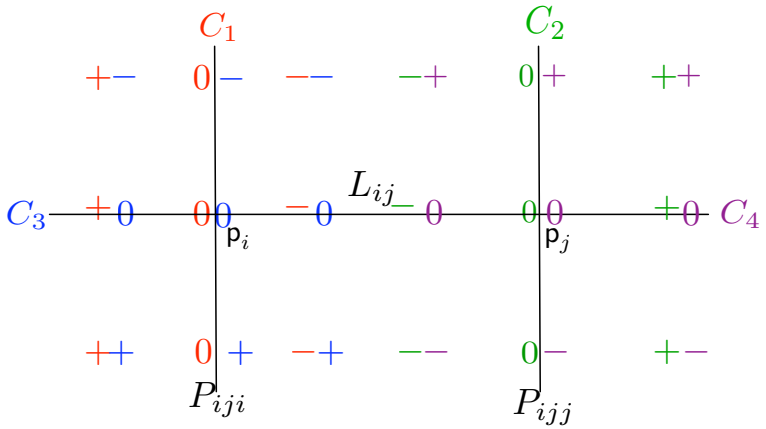


Figure 2.9: The quadrants and half lines where C_1, C_2, C_3 and C_4 take different values.

Remark 2.2. Since C_1, C_2, C_3 and C_4 take values from the set $\{-, 0, +\}$, it may seem that there are $3^4 = 81$ possible values for the tuples $(C_1\ C_2\ C_3\ C_4)$.

However, some combinations are not possible because of Assumption 1, that says that two consecutive vertices of a polyline have to be different. This means that C_1 and C_3 cannot be both 0 and that C_2 and C_4 cannot be both 0, in each case with the exception of C_1, C_2, C_3 and C_4 all being 0, that is $(C_1\ C_2\ C_3\ C_4) = (0\ 0\ 0\ 0)$. So, we have $81 - 8 - 8 = 65$ possible values for $(C_1\ C_2\ C_3\ C_4)$.

This number of 65 possible values for the tuples $(C_1\ C_2\ C_3\ C_4)$ can also be reached in another way. The point p_{i+1} can be in one of four quadrants around p_i or on one of four half lines starting in p_i. These are 8 possible locations for p_{i+1}. Similarly, we have 8 possible locations for p_{j+1} in the quadrants and half lines starting in p_j. This gives $8 \times 8 = 64$ possible combinations. Together with the case $(C_1\ C_2\ C_3\ C_4) = (0\ 0\ 0\ 0)$, we reach a total number of 65 possibilities. □

2.2.2 The double-cross matrix of a polyline

Based on the definition of $\mathsf{DC}(\overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}}, \overrightarrow{\mathsf{p}_j\mathsf{p}_{j+1}})$, we now define the double-cross matrix of a polyline.

Definition 2.3. Let $P = \langle(\mathsf{x}_0, \mathsf{y}_0), (\mathsf{x}_1, \mathsf{y}_1), , \ldots, (\mathsf{x}_N, \mathsf{y}_N)\rangle$ be a polyline, with $\mathsf{p}_i = (\mathsf{x}_i, \mathsf{y}_i)$, for $0 \leq i \leq N$, and with associated vectors $\overrightarrow{\mathsf{p}_0\mathsf{p}_1}, \overrightarrow{\mathsf{p}_1\mathsf{p}_2}, \ldots, \overrightarrow{\mathsf{p}_{N-1}\mathsf{p}_N}$. The *double-cross matrix* of P, denoted $\mathsf{DCM}(P)$, is the $N \times N$ matrix with the entries $\mathsf{DCM}(P)[i, j] = \mathsf{DC}(\overrightarrow{p_ip_{i+1}}, \overrightarrow{p_jp_{j+1}})$, for $0 \leq i, j < N$. □

	$\overrightarrow{p_0p_1}$	$\overrightarrow{p_1p_2}$	$\overrightarrow{p_2p_3}$	$\overrightarrow{p_3p_4}$	$\overrightarrow{p_4p_5}$
$\overrightarrow{p_0p_1}$	$(0\ 0\ 0\ 0)$	$(-\ -\ 0\ +)$	$(-\ +\ +\ -)$	$(-\ -\ +\ -)$	$(-\ +\ -\ +)$
$\overrightarrow{p_1p_2}$	$(-\ -\ +\ 0)$	$(0\ 0\ 0\ 0)$	$(-\ -\ 0\ +)$	$(-\ +\ +\ +)$	$(-\ -\ +\ +)$
$\overrightarrow{p_2p_3}$	$(-\ -\ -\ +)$	$(-\ -\ +\ 0)$	$(0\ 0\ 0\ 0)$	$(-\ +\ 0\ -)$	$(-\ -\ -\ +)$
$\overrightarrow{p_3p_4}$	$(-\ -\ -\ +)$	$(+\ -\ +\ +)$	$(+\ -\ -\ 0)$	$(0\ 0\ 0\ 0)$	$(-\ -\ 0\ +)$
$\overrightarrow{p_4p_5}$	$(+\ -\ +\ -)$	$(-\ -\ +\ +)$	$(-\ -\ +\ -)$	$(-\ -\ +\ 0)$	$(0\ 0\ 0\ 0)$

Table 2.1: The entries of the double-cross matrix of the polyline of Figure 2.10.

For example, the entries of the double-cross matrix of the polyline of Figure 2.10 are given in Table 2.1. This first example can be used to illustrate some properties of this matrix that are proven in Section 3.2. First, we observe that on the diagonal $(0\ 0\ 0\ 0)$ always appears. We also see that there is a certain degree of symmetry along the diagonal. If $\mathsf{DC}(\overrightarrow{p_ip_{i+1}}, \overrightarrow{p_jp_{j+1}}) = (C_1\ C_2\ C_3\ C_4)$, then we have $\mathsf{DC}(\overrightarrow{p_jp_{j+1}}, \overrightarrow{p_ip_{i+1}}) = (C_2\ C_1\ C_4\ C_3)$. These two observations imply that it suffices to know a double-cross matrix *above its diagonal*.

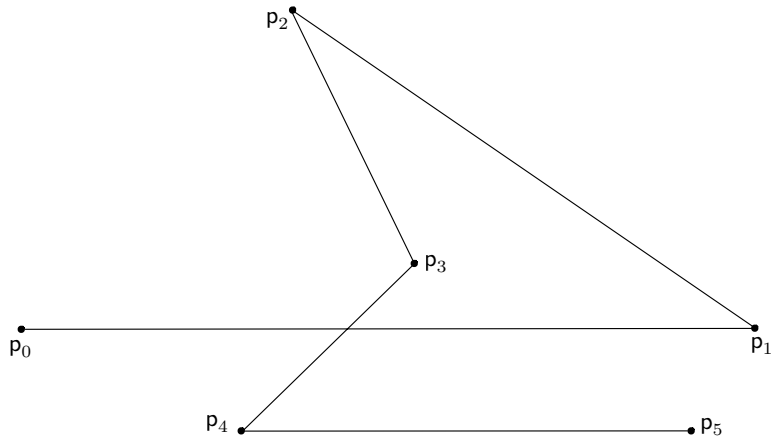


Figure 2.10: An example of a polyline.

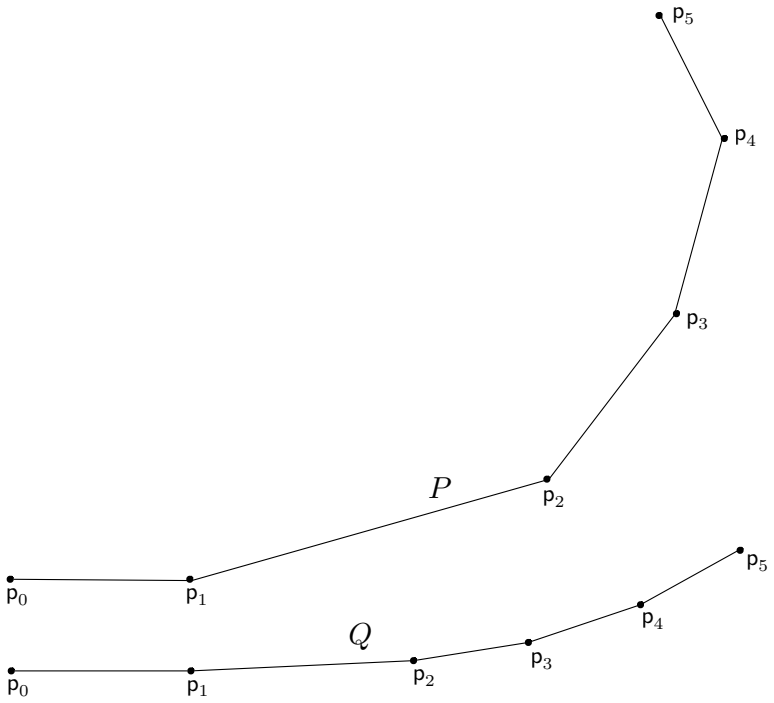


Figure 2.11: The polylines P and Q are double-cross similar.

2.3 Double-cross similarity of polylines

We now define *double-cross similarity* of two polylines of equal size.

Definition 2.4. Let P and Q be polylines of the same size. We say that P and Q are *double-cross similar* if $\mathsf{DCM}(P) = \mathsf{DCM}(Q)$. □

We stress that Definition 2.4 requires that the two polylines have to be of the same size before we can speak of their double-cross similarity.

Figure 2.11 depicts two polylines, P and Q, which are double-cross similar. The entries of their double-cross matrices are given in Table 2.2. In polyline P of Figure 2.11, at each vertex, the polyline bends around 10 degrees to the left. In polyline Q, this is only around 2 degrees. Nevertheless, all relative positions of oriented line segments remain the same. As the most extreme example, if we compare $\overrightarrow{\mathsf{p}_0\mathsf{p}_1}$ and $\overrightarrow{\mathsf{p}_4\mathsf{p}_5}$ in both polylines, we see that $\overrightarrow{\mathsf{p}_4\mathsf{p}_5}$ almost makes a 90° left angle with the central line of the double cross in the polyline P, whereas, this is only some 10° in the polyline Q. Still, both P and Q have the same double-cross entry for $\overrightarrow{\mathsf{p}_0\mathsf{p}_1}$ and $\overrightarrow{\mathsf{p}_4\mathsf{p}_5}$.

	$\overrightarrow{p_0p_1}$	$\overrightarrow{p_1p_2}$	$\overrightarrow{p_2p_3}$	$\overrightarrow{p_3p_4}$	$\overrightarrow{p_4p_5}$
$\overrightarrow{p_0p_1}$	(0 0 0 0)	(− + 0 +)	(− + + +)	(− + + +)	(− + + +)
$\overrightarrow{p_1p_2}$	(+ − + 0)	(0 0 0 0)	(− + 0 +)	(− + + +)	(− + + +)
$\overrightarrow{p_2p_3}$	(+ − + +)	(+ − + 0)	(0 0 0 0)	(− + 0 +)	(− + + +)
$\overrightarrow{p_3p_4}$	(+ − + +)	(+ − + +)	(+ − + 0)	(0 0 0 0)	(− + 0 +)
$\overrightarrow{p_4p_5}$	(+ − + +)	(+ − + +)	(+ − + +)	(+ − + 0)	(0 0 0 0)

Table 2.2: The entries of the double-cross matrix of the polylines of Figure 2.11.

We end this chapter with a variant of Definition 2.4. If we compare line segments of a polyline only with the next k line segments in a polyline, we get the following definition.

Definition 2.5. The *k-partial double-cross matrix of P*, denoted as $\mathsf{DCM}^k(P)$, is defined as $\mathsf{DCM}^k(P) = \{\mathsf{DCM}(P)[i,j] | i < j \leq max((i+k), N-1)\}$. Two polylines P and Q are called *k-double-cross similar* if $\mathsf{DCM}^k(P) = \mathsf{DCM}^k(Q)$. □

Clearly, for polylines consisting of N line segments, the notions of double-cross similarity and $(N-1)$-double-cross similarity coincide.

3

Algebraic and geometric characterizations of double-cross matrices of polylines

In this chapter, we first give an algebraic characterization of the double-cross matrix of a polyline and discuss some basic properties of double-cross matrices that can be derived from the algebraic characterization. Next, we give a geometric characterization of double-cross similarity of two polylines. To end this chapter, we identify the transformations of the plane that leave the double-cross matrix of all polylines in $\mathbf{R}^2$ invariant.

3.1 An algebraic characterization of the double-cross matrix of a polyline

In this section, we give an algebraic characterization of the entries in the double-cross matrix of a polyline.

Let $P = \langle(\mathsf{x}_0, \mathsf{y}_0), (\mathsf{x}_1, \mathsf{y}_1), , \ldots, (\mathsf{x}_N, \mathsf{y}_N)\rangle$ be a polyline and let $\mathsf{p}_i = (\mathsf{x}_i, \mathsf{y}_i)$, for $0 \leq i \leq N$. Theorem 3.2 gives algebraic expressions to calculate the entries $\mathsf{DC}(\overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}}, \overrightarrow{\mathsf{p}_j\mathsf{p}_{j+1}})$ of a double-cross matrix in terms of the x- and y-coordinates of the points $\mathsf{p}_i, \mathsf{p}_{i+1}, \mathsf{p}_j$ and p_{j+1}. Further on, we use this theorem extensively to prove properties of double-cross matrices.

Before stating and proving this theorem, we recall some elementary notations from algebra and some formula's in the following remark.

Remark 3.1. The well-known formula to calculate the (counter-clockwise) angle θ between two vectors[1] $\vec{a}$ and $\vec{b}$ in $\mathbf{R}^2$ (and also, in general, in $\mathbf{R}^n$) is

$$\cos\theta = \frac{\vec{a}\cdot\vec{b}}{|\vec{a}|\cdot|\vec{b}|}.$$

Here, the $\cdot$ in the numerator denotes the *inner product*[2] of two vectors and $|\vec{a}|$ is the norm or length of $\vec{a}$ (and the $\cdot$ in the denominator is the product of real numbers).

The above formula implies that we have $\cos\theta = 0$ if and only if $\vec{a}\cdot\vec{b} = 0$. And we have $\vec{a}\cdot\vec{b} = 0$ if and only if $\theta \in \{90°, -90°\}$. So, $\vec{a}\cdot\vec{b} = 0$ means that $\vec{a}$ is perpendicular to $\vec{b}$. On the other hand, we have $\cos\theta > 0$ and thus $\vec{a}\cdot\vec{b} > 0$, when $\theta \in (-90°, 90°)$. And finally $\vec{a}\cdot\vec{b} < 0$ is equivalent to $\theta \in (90°, 270°)$.

If $\vec{a} = (a, b) \in \mathbf{R}^2$, then $\vec{a}^{\perp} = (-b, a)$ is the unique vector, perpendicular to $\vec{a}$ and of the same length of $\vec{a}$, such that the (counter-clockwise) angle from $\vec{a}$ to $\vec{a}^{\perp}$ is 90°. □

In the following theorem, we use the function

$$\text{sign} : \mathbf{R} \to \{-, 0, +\} : x \mapsto \text{sign}(x) = \begin{cases} - & \text{if} \quad x < 0; \\ 0 & \text{if} \quad x = 0; \text{ and} \\ + & \text{if} \quad x > 0. \end{cases}$$

We also work with the following convention concerning signs: $--$ is $+$; -0 is 0; and $-+$ is $-$.

Theorem 3.2. *Let* $P = \langle(\mathsf{x}_0, \mathsf{y}_0), (\mathsf{x}_1, \mathsf{y}_1), , \ldots, (\mathsf{x}_N, \mathsf{y}_N)\rangle$ *be a polyline and let* $\mathsf{p}_i = (\mathsf{x}_i, \mathsf{y}_i)$*, for* $0 \le i \le N$*. Then,* $\mathsf{DC}(\overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}}, \overrightarrow{\mathsf{p}_j\mathsf{p}_{j+1}}) = (C_1\ C_2\ C_3\ C_4)$ *with*

$$\begin{array}{lcll} C_1 & = & - & \text{sign}((\mathsf{x}_j - \mathsf{x}_i)\cdot(\mathsf{x}_{i+1} - \mathsf{x}_i) + (\mathsf{y}_j - \mathsf{y}_i)\cdot(\mathsf{y}_{i+1} - \mathsf{y}_i)); \\ C_2 & = & & \text{sign}((\mathsf{x}_j - \mathsf{x}_i)\cdot(\mathsf{x}_{j+1} - \mathsf{x}_j) + (\mathsf{y}_j - \mathsf{y}_i)\cdot(\mathsf{y}_{j+1} - \mathsf{y}_j)); \\ C_3 & = & - & \text{sign}((\mathsf{x}_j - \mathsf{x}_i)\cdot(\mathsf{y}_{i+1} - \mathsf{y}_i) - (\mathsf{y}_j - \mathsf{y}_i)\cdot(\mathsf{x}_{i+1} - \mathsf{x}_i)); \textit{ and} \\ C_4 & = & & \text{sign}((\mathsf{x}_j - \mathsf{x}_i)\cdot(\mathsf{y}_{j+1} - \mathsf{y}_j) - (\mathsf{y}_j - \mathsf{y}_i)\cdot(\mathsf{x}_{j+1} - \mathsf{x}_j)). \end{array}$$

Proof. We have $\mathsf{p}_i = \mathsf{p}_j$ if and only if $\mathsf{x}_j - \mathsf{x}_i = 0$ and $\mathsf{y}_j - \mathsf{y}_i = 0$ and in this case the four (instantiated) polynomials in the statement of the theorem evaluate to zero.

Next, we assume $\mathsf{p}_i \neq \mathsf{p}_j$. We consider the following vectors in $\mathbf{R}^2$:

[1] Now, we are not talking about *located* vectors like before, but vectors in the common sense.

[2] The inner product is also called scalar product.

- $\overrightarrow{u_{ij}} = (\mathsf{x}_j - \mathsf{x}_i, \mathsf{y}_j - \mathsf{y}_i)$;
- $\overrightarrow{u_{ji}} = (\mathsf{x}_i - \mathsf{x}_j, \mathsf{y}_i - \mathsf{y}_j)$;
- $\overrightarrow{v_i} = (\mathsf{x}_{i+1} - \mathsf{x}_i, \mathsf{y}_{i+1} - \mathsf{y}_i)$; and
- $\overrightarrow{v_j} = (\mathsf{x}_{j+1} - \mathsf{x}_j, \mathsf{y}_{j+1} - \mathsf{y}_j)$.

We remark that $\overrightarrow{u_{ij}} = -\overrightarrow{u_{ji}}$ and that the vectors $\overrightarrow{u_{ij}}$, $\overrightarrow{v_i}$ and $\overrightarrow{v_j}$ (in the common sense of the word vector) are the (located) vectors $\overrightarrow{\mathsf{p}_i\mathsf{p}_j}$, $\overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}}$ and $\overrightarrow{\mathsf{p}_j\mathsf{p}_{j+1}}$ translated to the origin of $\mathbf{R}^2$.

• C_1: Now, we apply the above cosine-formula to $\overrightarrow{a} = \overrightarrow{u_{ij}}$ and $\overrightarrow{b} = \overrightarrow{v_i}$ to obtain an expression for C_1. Because C_1 is negative towards p_j, we get the minus-sign in the following expression for C_1:

$$\begin{aligned} C_1 &= -\text{sign}(\overrightarrow{u_{ij}} \cdot \overrightarrow{v_i}) \\ &= -\text{sign}((\mathsf{x}_j - \mathsf{x}_i) \cdot (\mathsf{x}_{i+1} - \mathsf{x}_i) + (\mathsf{y}_j - \mathsf{y}_i) \cdot (\mathsf{y}_{i+1} - \mathsf{y}_i)). \end{aligned}$$

• C_2: Next, we apply the cosine-formula to $\overrightarrow{a} = \overrightarrow{u_{ji}}$ and $\overrightarrow{b} = \overrightarrow{v_j}$ to obtain an expression for C_2. Again, because C_2 is negative towards p_i, we get the minus-sign in $C_2 = -\text{sign}(\overrightarrow{u_{ji}} \cdot \overrightarrow{v_j})$. This means that

$$\begin{aligned} C_2 &= \text{sign}(\overrightarrow{u_{ij}} \cdot \overrightarrow{v_j}) \\ &= \text{sign}((\mathsf{x}_j - \mathsf{x}_i) \cdot (\mathsf{x}_{j+1} - \mathsf{x}_j) + (\mathsf{y}_j - \mathsf{y}_i) \cdot (\mathsf{y}_{j+1} - \mathsf{y}_j)). \end{aligned}$$

• C_3: Here, we apply the cosine-formula to $\overrightarrow{a} = \overrightarrow{u_{ij}}^{\perp}$ and $\overrightarrow{b} = \overrightarrow{v_i}$ and get $C_3 = -\text{sign}(\overrightarrow{u_{ij}}^{\perp} \cdot \overrightarrow{v_i})$. We have a minus-sign here, because $C_3 = -$ in the direction of $\overrightarrow{u_{ij}}^{\perp}$. Since $\overrightarrow{u_{ij}}^{\perp} = (-(\mathsf{y}_j - \mathsf{y}_i), \mathsf{x}_j - \mathsf{x}_i)$, we get

$$\begin{aligned} C_3 &= -\text{sign}(\overrightarrow{u_{ij}}^{\perp} \cdot \overrightarrow{v_i}) \\ &= \text{sign}((\mathsf{y}_j - \mathsf{y}_i) \cdot (\mathsf{x}_{i+1} - \mathsf{x}_i) - (\mathsf{x}_j - \mathsf{x}_i) \cdot (\mathsf{y}_{i+1} - \mathsf{y}_i)). \end{aligned}$$

• C_4: Finally, we apply the cosine-formula to $\overrightarrow{a} = \overrightarrow{u_{ji}}^{\perp}$ and $\overrightarrow{b} = \overrightarrow{v_j}$. Since $C_4 = -$ in the direction of $\overrightarrow{u_{ji}}^{\perp}$, we have $C_4 = -\text{sign}(\overrightarrow{u_{ji}}^{\perp} \cdot \overrightarrow{v_j})$. Since $\overrightarrow{u_{ji}}^{\perp} = (\mathsf{y}_j - \mathsf{y}_i, -(\mathsf{x}_j - \mathsf{x}_i))$, we get

$$\begin{aligned} C_4 &= -\text{sign}(\overrightarrow{u_{ji}}^{\perp} \cdot \overrightarrow{v_j}) \\ &= \text{sign}(-(\mathsf{y}_j - \mathsf{y}_i) \cdot (\mathsf{x}_{j+1} - \mathsf{x}_j) + (\mathsf{x}_j - \mathsf{x}_i) \cdot (\mathsf{y}_{i+1} - \mathsf{y}_i)). \end{aligned}$$

This concludes the proof. □

In the following property, we show that the double-cross value (0 0 0 0), which, for reasons of continuity, is the value in the case $\mathsf{p}_i = \mathsf{p}_j$ (see Definition 2.1), can only occur in that exceptional case.

Property 3.1. Let $P = \langle(\mathsf{x}_0, \mathsf{y}_0), (\mathsf{x}_1, \mathsf{y}_1), , \ldots, (\mathsf{x}_N, \mathsf{y}_N)\rangle$ be a polyline and let $\mathsf{p}_i = (\mathsf{x}_i, \mathsf{y}_i)$. Then, $\mathsf{DC}(\overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}}, \overrightarrow{\mathsf{p}_j\mathsf{p}_{j+1}}) = (0\ 0\ 0\ 0)$ if and only if $\mathsf{p}_i = \mathsf{p}_j$.

Proof. As already observed in the proof of Theorem 3.2, $\mathsf{p}_i = \mathsf{p}_j$ implies $\mathsf{x}_j - \mathsf{x}_i = 0$ and $\mathsf{y}_j - \mathsf{y}_i = 0$ and in this case the four (instantiated) polynomials of Theorem 3.2 evaluate to zero. This implies that $\mathsf{DC}(\overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}}, \overrightarrow{\mathsf{p}_j\mathsf{p}_{j+1}}) = (0\,0\,0\,0)$.

For the converse, we have to show that if the four polynomials evaluate to zero, then $\mathsf{p}_i = \mathsf{p}_j$. We prove this by assuming $\mathsf{p}_i \neq \mathsf{p}_j$ and deriving a contradiction. If $\mathsf{p}_i \neq \mathsf{p}_j$, then $\mathsf{x}_j - \mathsf{x}_i \neq 0$ or $\mathsf{y}_j - \mathsf{y}_i \neq 0$. First, we consider the case $\mathsf{x}_j - \mathsf{x}_i \neq 0$.

As a first subcase, we consider the case $\mathsf{y}_j - \mathsf{y}_i = 0$. Then we get from the equations $C_1 = 0$ and $C_3 = 0$ that $(\mathsf{x}_j - \mathsf{x}_i) \cdot (\mathsf{x}_{i+1} - \mathsf{x}_i) = 0$ and $(\mathsf{x}_j - \mathsf{x}_i) \cdot (\mathsf{y}_{i+1} - \mathsf{y}_i) = 0$. Since $\mathsf{x}_j - \mathsf{x}_i \neq 0$ is assumed, this implies that $\mathsf{x}_{i+1} - \mathsf{x}_i = 0$ and $\mathsf{y}_{i+1} - \mathsf{y}_i = 0$. This contradicts Assumption 1, which says that no two consecutive vertices of a polyline are identical.

As a second subcase, we consider the case $\mathsf{y}_j - \mathsf{y}_i \neq 0$. Then we get from $C_1 = 0$ that

$$\mathsf{x}_{i+1} - \mathsf{x}_i = \frac{-(\mathsf{y}_j - \mathsf{y}_i) \cdot (\mathsf{y}_{i+1} - \mathsf{y}_i)}{\mathsf{x}_j - \mathsf{x}_i}.$$

From $C_3 = 0$, we get

$$\mathsf{x}_{i+1} - \mathsf{x}_i = \frac{(\mathsf{x}_j - \mathsf{x}_i) \cdot (\mathsf{y}_{i+1} - \mathsf{y}_i)}{\mathsf{y}_j - \mathsf{y}_i}.$$

Combined, these two equalities imply $((\mathsf{x}_j - \mathsf{x}_i)^2 + (\mathsf{y}_j - \mathsf{y}_i)^2) \cdot (\mathsf{y}_{i+1} - \mathsf{y}_i) = 0$. Since in this case $(\mathsf{x}_j - \mathsf{x}_i)^2 + (\mathsf{y}_j - \mathsf{y}_i)^2 > 0$, we conclude $\mathsf{y}_{i+1} - \mathsf{y}_i = 0$. But then, again using the equation for C_1, we get $(\mathsf{x}_j - \mathsf{x}_i) \cdot (\mathsf{x}_{i+1} - \mathsf{x}_i) = 0$, or $\mathsf{x}_{i+1} - \mathsf{x}_i = 0$. So, we have both $\mathsf{x}_{i+1} - \mathsf{x}_i = 0$ and $\mathsf{y}_{i+1} - \mathsf{y}_i = 0$, which again contradicts Assumption 1. We have contradiction in all cases and this concludes the proof of the first case. The case $\mathsf{y}_j - \mathsf{y}_i \neq 0$ has a completely analogous proof, now using $C_2 = 0$ and $C_4 = 0$ instead of $C_1 = 0$ and $C_3 = 0$. This concludes the proof. □

We end this section by remarking that all the factors appearing in the algebraic expressions, given by the theorem, that is $\mathsf{x}_j - \mathsf{x}_i$, $\mathsf{x}_{i+1} - \mathsf{x}_i$, $\mathsf{y}_j - \mathsf{y}_i$, $\mathsf{y}_{i+1} - \mathsf{y}_i$, $\mathsf{x}_{j+1} - \mathsf{x}_j$ and $\mathsf{y}_{j+1} - \mathsf{y}_j$ are differences in x-coordinate or differences in y-coordinate values. In Chapter 6, we come back to this remark.

3.2 Some properties of double-cross matrices that can be derived from their algebraic characterisation

In this section, we give some basic properties of double-cross matrices of polylines. In most cases, these properties can be derived from the algebraic charac-

terization of the entries of a double-cross matrix, that we presented in previous section.

3.2.1 Symmetry in the double-cross matrix of a polyline

In Section 2.2, we have already announced by the example polyline given in Figure 2.10 with its double-cross matrix given in Table 2.1, that a double-cross matrix exhibits symmetry properties. We prove these properties in this section. The first property is by definition, the second needs some inspection of polynomials. The conclusion is that it is enough to know a double-cross matrix *above its diagonal*.

Property 3.2. If $P = \langle \mathsf{p}_0, \mathsf{p}_1, , \ldots, \mathsf{p}_N \rangle$ is a polyline, then $\mathsf{DC}(\overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}}, \overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}})$ $= (0\ 0\ 0\ 0)$, for $0 \leq i < N$. □

The following property says how $\mathsf{DC}(\overrightarrow{\mathsf{p}_j\mathsf{p}_{j+1}}, \overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}})$ can be derived from $\mathsf{DC}(\overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}}, \overrightarrow{\mathsf{p}_j\mathsf{p}_{j+1}})$ in a straightforward way.

Property 3.3. Let $P = \langle \mathsf{p}_0, \mathsf{p}_1, , \ldots, \mathsf{p}_N \rangle$ be a polyline. If $\mathsf{DC}(\overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}}, \overrightarrow{\mathsf{p}_j\mathsf{p}_{j+1}}) =$ $(C_1\ C_2\ C_3\ C_4)$, then $\mathsf{DC}(\overrightarrow{\mathsf{p}_j\mathsf{p}_{j+1}}, \overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}}) = (C_2\ C_1\ C_4\ C_3)$.

Proof. Let $P = \langle (\mathsf{x}_0, \mathsf{y}_0), (\mathsf{x}_1, \mathsf{y}_1), , \ldots, (\mathsf{x}_N, \mathsf{y}_N) \rangle$ be a polyline and let $\mathsf{p}_i = (\mathsf{x}_i, \mathsf{y}_i)$. We use the polynomials given in Theorem 3.2 to prove this result. Essentially, what we do is to interchange the role of i and j. If $i = j$, nothing has to be shown. So, we assume $i \neq j$. If we interchange in

$$
\begin{array}{lcll}
C_1 & = & - & \mathrm{sign}((\mathsf{x}_j - \mathsf{x}_i) \cdot (\mathsf{x}_{i+1} - \mathsf{x}_i) + (\mathsf{y}_j - \mathsf{y}_i) \cdot (\mathsf{y}_{i+1} - \mathsf{y}_i)); \\
C_2 & = & & \mathrm{sign}((\mathsf{x}_j - \mathsf{x}_i) \cdot (\mathsf{x}_{j+1} - \mathsf{x}_j) + (\mathsf{y}_j - \mathsf{y}_i) \cdot (\mathsf{y}_{j+1} - \mathsf{y}_j)); \\
C_3 & = & - & \mathrm{sign}((\mathsf{x}_j - \mathsf{x}_i) \cdot (\mathsf{y}_{i+1} - \mathsf{y}_i) - (\mathsf{y}_j - \mathsf{y}_i) \cdot (\mathsf{x}_{i+1} - \mathsf{x}_i)); \text{ and} \\
C_4 & = & & \mathrm{sign}((\mathsf{x}_j - \mathsf{x}_i) \cdot (\mathsf{y}_{j+1} - \mathsf{y}_j) - (\mathsf{y}_j - \mathsf{y}_i) \cdot (\mathsf{x}_{j+1} - \mathsf{x}_j)).
\end{array}
$$

the role of i and j, we get $\mathsf{DC}(\overrightarrow{\mathsf{p}_j\mathsf{p}_{j+1}}, \overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}}) = (C'_1\ C'_2\ C'_3\ C'_4)$, with

$$
\begin{array}{lcll}
C'_1 & = & - & \mathrm{sign}((\mathsf{x}_i - \mathsf{x}_j) \cdot (\mathsf{x}_{j+1} - \mathsf{x}_j) + (\mathsf{y}_i - \mathsf{y}_j) \cdot (\mathsf{y}_{j+1} - \mathsf{y}_j)); \\
C'_2 & = & & \mathrm{sign}((\mathsf{x}_i - \mathsf{x}_j) \cdot (\mathsf{x}_{i+1} - \mathsf{x}_i) + (\mathsf{y}_i - \mathsf{y}_j) \cdot (\mathsf{y}_{i+1} - \mathsf{y}_i)); \\
C'_3 & = & - & \mathrm{sign}((\mathsf{x}_i - \mathsf{x}_j) \cdot (\mathsf{y}_{j+1} - \mathsf{y}_j) - (\mathsf{y}_i - \mathsf{y}_j) \cdot (\mathsf{x}_{j+1} - \mathsf{x}_j)); \text{ and} \\
C'_4 & = & & \mathrm{sign}((\mathsf{x}_i - \mathsf{x}_j) \cdot (\mathsf{y}_{i+1} - \mathsf{y}_i) - (\mathsf{y}_i - \mathsf{y}_j) \cdot (\mathsf{x}_{i+1} - \mathsf{x}_i)).
\end{array}
$$

It is easy to see that $C'_1 = C_2$, $C'_2 = C_1$, $C'_3 = C_4$ and $C'_4 = C_3$. □

These two properties implies that only the $\frac{N \cdot (N-1)}{2}$ entries above the diagonal of the double-cross matrix of a polyline are significant.

3.2.2 The double-cross value of consecutive line segments

The following property says what the entries in the double-cross matrix of two successive line segments $\overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}}$ and $\overrightarrow{\mathsf{p}_{i+1}\mathsf{p}_{i+2}}$ in a polyline $P = \langle \mathsf{p}_0, \mathsf{p}_1, \ldots, \mathsf{p}_N \rangle$ look like. These are the entries in de double-cross matrix just above its diagonal.

Property 3.4. Let $P = \langle \mathsf{p}_0, \mathsf{p}_1, , \ldots, \mathsf{p}_N \rangle$ be a polyline. In $\mathsf{DC}(\overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}}, \overrightarrow{\mathsf{p}_{i+1}\mathsf{p}_{i+2}})$ the entries C_1 and C_3 are fixed to $-$ and 0. That is,

$$\mathsf{DC}(\overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}}, \overrightarrow{\mathsf{p}_{i+1}\mathsf{p}_{i+2}}) = (-\ C_2\ 0\ C_4),$$

for any $0 \leq i < N - 1$.

Proof. Let $0 \leq i < N - 1$. We start with the entry $\mathsf{DC}(\overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}}, \overrightarrow{\mathsf{p}_{i+1}\mathsf{p}_{i+2}})[1] = -\text{sign}((\mathsf{x}_{i+1} - \mathsf{x}_i) \cdot (\mathsf{x}_{i+1} - \mathsf{x}_i) + (\mathsf{y}_{i+1} - \mathsf{y}_i) \cdot (\mathsf{y}_{i+1} - \mathsf{y}_i)) = -\text{sign}((\mathsf{x}_{i+1} - \mathsf{x}_i)^2 + (\mathsf{y}_{i+1} - \mathsf{y}_i)^2)$. Because of Assumption 1, we have $(\mathsf{x}_{i+1} - \mathsf{x}_i)^2 + (\mathsf{y}_{i+1} - \mathsf{y}_i)^2 > 0$ and we can conclude that $\mathsf{DC}(\overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}}, \overrightarrow{\mathsf{p}_{i+1}\mathsf{p}_{i+2}})[1] = -$.

For the third entry we have $\mathsf{DC}(\overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}}, \overrightarrow{\mathsf{p}_{i+1}\mathsf{p}_{i+2}})[3] = -\text{sign}((\mathsf{x}_{i+1} - \mathsf{x}_i) \cdot (\mathsf{y}_{i+1} - \mathsf{y}_i) - (\mathsf{y}_{i+1} - \mathsf{y}_i) \cdot (\mathsf{x}_{i+1} - \mathsf{x}_i)) = -\text{sign}(0) = 0$. □

The following property shows that more values depend on one another.

Property 3.5. Let $P = \langle \mathsf{p}_0, \mathsf{p}_1, , \ldots, \mathsf{p}_N \rangle$ be a polyline and let $1 \leq i < N - 1$. If $\mathsf{DC}(\overrightarrow{\mathsf{p}_{i-1}\mathsf{p}_i}, \overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}}) = (-\ C_2\ 0\ C_4)$, with $C_2 = +$ or 0, then $\mathsf{DC}(\overrightarrow{\mathsf{p}_{i-1}\mathsf{p}_i}, \overrightarrow{\mathsf{p}_{i+1}\mathsf{p}_{i+2}}) = (-\ C_2'\ C_4\ C_4')$, for some $C_2', C_4' \in \{-, 0, +\}$.

Proof. Let $\mathsf{DC}(\overrightarrow{\mathsf{p}_{i-1}\mathsf{p}_i}, \overrightarrow{\mathsf{p}_{i+1}\mathsf{p}_{i+2}})$ be $(C_1'\ C_2'\ C_3'\ C_4')$. We have the following expressions:

$$\begin{array}{rcl} C_2 & = & \text{sign}((\mathsf{x}_i - \mathsf{x}_{i-1}) \cdot (\mathsf{x}_{i+1} - \mathsf{x}_i) + (\mathsf{y}_i - \mathsf{y}_{i-1}) \cdot (\mathsf{y}_{i+1} - \mathsf{y}_i)) \\ C_4 & = & \text{sign}((\mathsf{x}_i - \mathsf{x}_{i-1}) \cdot (\mathsf{y}_{i+1} - \mathsf{y}_i) - (\mathsf{y}_i - \mathsf{y}_{i-1}) \cdot (\mathsf{x}_{i+1} - \mathsf{x}_i)) \\ C_1' & = & -\text{sign}((\mathsf{x}_{i+1} - \mathsf{x}_{i-1}) \cdot (\mathsf{x}_i - \mathsf{x}_{i-1}) + (\mathsf{y}_{i+1} - \mathsf{y}_{i-1}) \cdot (\mathsf{y}_i - \mathsf{y}_{i-1})) \\ C_3' & = & -\text{sign}((\mathsf{x}_{i+1} - \mathsf{x}_{i-1}) \cdot (\mathsf{y}_i - \mathsf{y}_{i-1}) - (\mathsf{y}_{i+1} - \mathsf{y}_{i-1}) \cdot (\mathsf{x}_i - \mathsf{x}_{i-1})) \end{array}$$

Let us abbreviate the first two expression as $C_2 = \text{sign}(c_2)$ and $C_4 = \text{sign}(c_4)$ and the latter two as $C_1' = -\text{sign}(c_1')$ and $C_3' = -\text{sign}(c_3')$.

Then we have $c_1' = ((\mathsf{x}_{i+1} - \mathsf{x}_i) + (\mathsf{x}_i - \mathsf{x}_{i-1})) \cdot (\mathsf{x}_i - \mathsf{x}_{i-1}) + ((\mathsf{y}_{i+1} - \mathsf{y}_i) + (\mathsf{y}_i - \mathsf{y}_{i-1})) \cdot (\mathsf{y}_i - \mathsf{y}_{i-1}) = (\mathsf{x}_i - \mathsf{x}_{i-1})^2 + (\mathsf{y}_i - \mathsf{y}_{i-1})^2 + c_2$. Since, by assumption $c_2 \geq 0$, it follows from Assumption 1 that $c_1' > 0$ and thus $C_1' = -\text{sign}(c_1') = -$.

Further, we have $c_3' = ((\mathsf{x}_{i+1} - \mathsf{x}_i) + (\mathsf{x}_i - \mathsf{x}_{i-1})) \cdot (\mathsf{y}_i - \mathsf{y}_{i-1}) - ((\mathsf{y}_{i+1} - \mathsf{y}_i) + (\mathsf{y}_i - \mathsf{y}_{i-1})) \cdot (\mathsf{x}_i - \mathsf{x}_{i-1}) = -c_4$. So, $C_3' = -\text{sign}(c_3') = -\text{sign}(-c_4) = C_4$. This concludes the proof. □

3.2.3 On the length of line segments of a polyline

The following properties shows that changing the length of segments in a polyline may or may not influence certain entries in its double-cross matrix. By "changing the length of a segment", we mean that the origin of the located vector, determined by the segment, is not changed, but only its destination vertex is scaled out.

Property 3.6. Let $\overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}}$ and $\overrightarrow{\mathsf{p}_j\mathsf{p}_{j+1}}$ be two located vectors.Changing the length of $\overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}}$ and $\overrightarrow{\mathsf{p}_j\mathsf{p}_{j+1}}$ does not influence the value of $\mathsf{DC}(\overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}}, \overrightarrow{\mathsf{p}_j\mathsf{p}_{j+1}})$.

Proof. Let $\overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}}$ and $\overrightarrow{\mathsf{p}_j\mathsf{p}_{j+1}}$ be two located vectors. And let $\mathsf{p}_k = (\mathsf{x}_k, \mathsf{y}_k)$, for $k \in \{i, i+1, j, j+1\}$.

If we take $\mathsf{DC}(\overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}}, \overrightarrow{\mathsf{p}_j\mathsf{p}_{j+1}}) = (C_1\ C_2\ C_3\ C_4)$ and $\mathsf{DC}(\overrightarrow{\mathsf{p}_i\mathsf{p}'_{i+1}}, \overrightarrow{\mathsf{p}_j\mathsf{p}'_{j+1}}) = (C'_1\ C'_2\ C'_3\ C'_4)$, where $\overrightarrow{\mathsf{p}_i\mathsf{p}'_{i+1}}$ is $\overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}}$ scaled by a factor c, with $c > 0$ and $\overrightarrow{\mathsf{p}_j\mathsf{p}'_{j+1}}$ is $\overrightarrow{\mathsf{p}_j\mathsf{p}_{j+1}}$ scaled by a factor d, with $d > 0$, then we first observe that $\mathsf{p}'_{i+1} = (x_i + c\cdot(x_{i+1} - x_i), y_i + c\cdot(y_{i+1} - y_i))$ and $\mathsf{p}'_{j+1} = (x_j + c\cdot(x_{j+1} - x_j), y_j + c\cdot(y_{j+1} - y_j))$. If we use the abbreviations c_1, c_2, c_3 and c_4 for the polynomials in Theorem 3.2, such that $C_1 = -\text{sign}(c_1)$, $C_2 = \text{sign}(c_2)$, $C_3 = -\text{sign}(c_3)$ and $C_4 = -\text{sign}(c_4)$, then it is easily verified that $C'_1 = -\text{sign}(c \cdot c_1) = -\text{sign}(c_1) = C_1$, since $c > 0$. Similarly, we get $C'_2 = \text{sign}(d \cdot c_2) = \text{sign}(c_2) = C_2$, $C'_3 = -\text{sign}(c \cdot c_3) = -\text{sign}(c_3) = C_3$ and $C'_4 = \text{sign}(d \cdot c_4) = \text{sign}(c_4) = C_4$, since also $d > 0$. This concludes the proof. □

The length of the last segment of a polyline does not influence the double-cross matrix. Only its direction matters. This follows straightforwardly from the definition.

Property 3.7. Let $P = \langle \mathsf{p}_0, \mathsf{p}_1, , \ldots, \mathsf{p}_N \rangle$ be a polyline. Changing the length of $\overrightarrow{\mathsf{p}_{N-1}\mathsf{p}_N}$ does not change $\mathsf{DCM}(P)$. □

For segments, that differ from the last, this is not the case, as the following property shows.

Property 3.8. Let $P = \langle \mathsf{p}_0, \mathsf{p}_1, , \ldots, \mathsf{p}_N \rangle$ be a polyline. Changing the length of $\overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}}$, for $0 \leq i < N - 1$, may change $\mathsf{DCM}(P)$.

Proof. Consider the polylines $P = \langle \mathsf{p}_0, \mathsf{p}_1, \mathsf{p}_2, \mathsf{p}_3, \mathsf{p}_4 \rangle$ and $Q = \langle \mathsf{q}_0, \mathsf{q}_1, \mathsf{q}_2, \mathsf{q}_3, \mathsf{q}_4 \rangle$ of Figure 3.1. They only differ in the length of their third segment. For P, we have $\mathsf{DC}(\overrightarrow{\mathsf{p}_0\mathsf{p}_1}, \overrightarrow{\mathsf{p}_3\mathsf{p}_4}) = (-\ -\ -\ -)$, whereas for Q, we have $\mathsf{DC}(\overrightarrow{\mathsf{q}_0\mathsf{q}_1}, \overrightarrow{\mathsf{q}_3\mathsf{q}_4}) = (+\ +\ -\ -)$. □

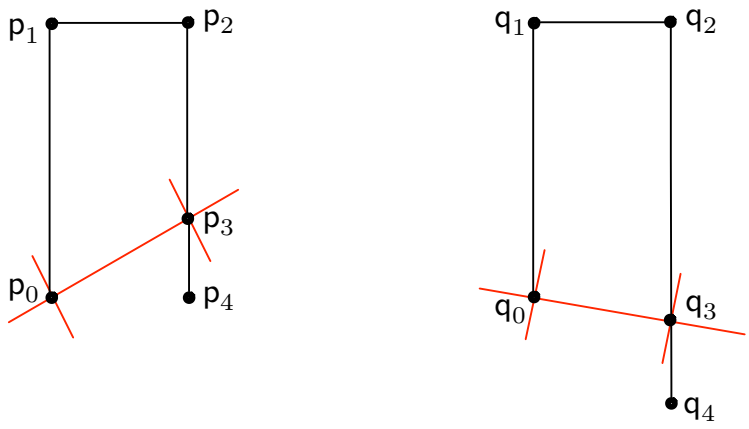


Figure 3.1: Two polylines that differ in the length of their third segment.

3.2.4 The quadrant of points of a polyline

From Section 3.4, we know that we can apply a similarity transformation to a polyline without changing its double-cross matrix. Without loss of generality, we may therefore assume that the first line segment of the polyline is the unit interval of the x-axis, that is, $\mathsf{p}_0 = (0,0)$ and $\mathsf{p}_1 = (1,0)$.

The following property states that we can derive the quadrants in which all the other points are located from the double-cross matrix.

Property 3.9. Let $P = \langle \mathsf{p}_0, \mathsf{p}_1, , \ldots, \mathsf{p}_N \rangle$ be a polyline and assume that $\mathsf{p}_0 = (0,0)$ and $\mathsf{p}_1 = (1,0)$. Let $\mathsf{p}_i = (\mathsf{x}_i, \mathsf{y}_i)$ for $2 \leq i \leq N$. From $\mathsf{DC}(\overrightarrow{\mathsf{p}_0\mathsf{p}_1}, \overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}})$, we can determine $\text{sign}(\mathsf{x}_i)$ and $\text{sign}(\mathsf{y}_i)$.

Proof. It is clear that $C_1 = -\text{sign}((\mathsf{x}_i - 0) \cdot 1 + \mathsf{y}_i \cdot 0) = -\text{sign}(\mathsf{x}_i)$ and that $C_3 = -\text{sign}(\mathsf{x}_i \cdot 0 + \mathsf{y}_i \cdot 1) = -\text{sign}(-\mathsf{y}_i)$. □

3.3 A geometric characterization of the double-cross similarity of two polylines

In this section, we define the *local carrier order* of a polyline. This is a geometric concept and the main result of this section is a characterization of double-cross similarity of two polylines in terms of their local carrier orders.

3.3.1 The local carrier order of a polyline

Here, we give the definition of the local carrier order of a polyline. First, we introduce some notation for half-lines.

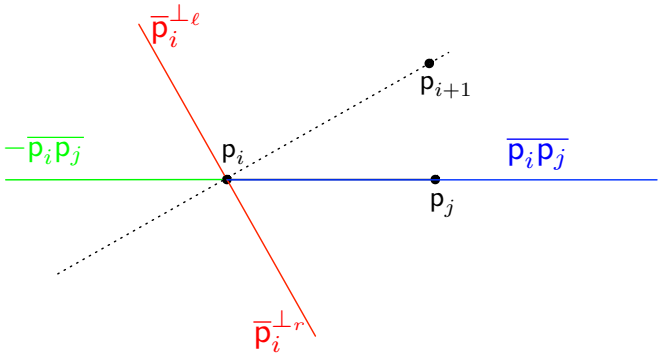


Figure 3.2: An example the half-lines $\overline{p_i p_j}$ (in blue), $-\overline{p_i p_j}$ (in green) and the two perpendicular half-lines $\overline{p_i}^{\perp_r}$ and $\overline{p_i}^{\perp_\ell}$ (in red) of Definition 3.3.

Definition 3.3. Let $P = \langle p_0, p_1, , \ldots, p_N \rangle$ be a polyline in $\mathbf{R}^2$ and let $0 \leq i < N$. If $p_i \neq p_j$, the (directed) half-line starting in p_i through p_j will be denoted by $\overline{p_i p_j}$. The half-line, also starting in p_i, but in the opposite direction is denoted $-\overline{p_i p_j}$. The half-lines $\overline{p_i p_j}$ and $-\overline{p_i p_j}$, for $0 \leq j \leq N$ with $j \neq i$ and $p_j \neq p_i$, are called the *carriers at* p_i.

The perpendicular half-line on $\overline{p_i p_{i+1}}$ starting in p_i directing to the right of $\overline{p_i p_{i+1}}$ (that is, making a 90° clockwise angle with $\overline{p_i p_{i+1}}$) as $\overline{p_i}^{\perp_r}$ and the opposite perpendicular half-line starting in p_i as $\overline{p_i}^{\perp_\ell}$. The half-lines $\overline{p_i}^{\perp_r}$ and $\overline{p_i}^{\perp_\ell}$ are called the *perpendiculars at* p_i. □

For $0 \leq i < N$, the point p_i has $2N$ carriers and 2 perpendiculars. For an illustration of the half-lines of and of this single cross between p_i and p_{i+1}, we refer to Figure 3.2.

Now, we define the local carrier order of a vertex p_i of a polyline P, for $0 \leq i < N$. This local carrier order consists of nine sets. One keeps track which p_j's are equal to p_i and the other eight are corresponding to eight directions of a compass card. We use the image of a 8-point compass with the northern cardinal direction in the direction of the half-line $\overline{p_i p_{i+1}}$ to name these sets.

In the following definition, we say that a half-line $\overline{\ell}$ is *strictly between* the two perpendicular half-lines $\overline{\ell_1}$ and $\overline{\ell_2}$, if they all have the same starting point and $\overline{\ell}$ is in the quadrant determined by $\overline{\ell_1}$ and $\overline{\ell_2}$ (following the counter-clockwise direction).

Definition 3.4. Let $P = \langle p_0, p_1, , \ldots, p_N \rangle$ be a polyline in $\mathbf{R}^2$. For $0 \leq i < N$, we define the following nine sets for the vertex p_i:

- $\mathsf{N}(p_i)$ is the set of $\overline{p_i p_j}$ equal to $\overline{p_i p_{i+1}}$;
- $\mathsf{NE}(p_i)$ is the set of $\overline{p_i p_j}$ strictly between $\overline{p_i p_{i+1}}$ and $\overline{p_i}^{\perp_r}$;

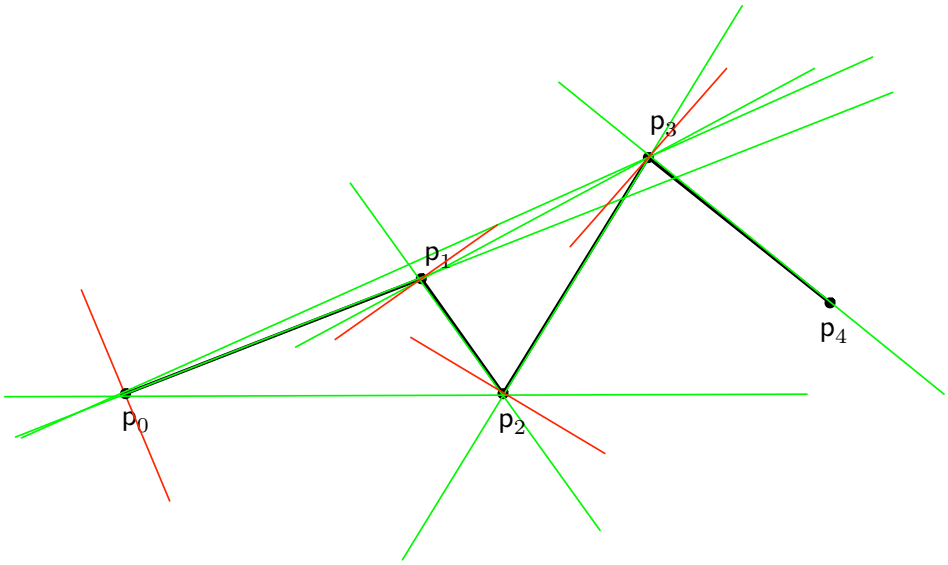


Figure 3.3: A polyline $P = \langle \mathsf{p}_0, \mathsf{p}_1, , \mathsf{p}_2, , \mathsf{p}_3, \mathsf{p}_4 \rangle$ with its carriers (in green) and its perpendiculars (in red).

- $\mathsf{E}(\mathsf{p}_i)$ is the set of $\overline{\mathsf{p}_i\mathsf{p}_j}$ equal to $\overline{\mathsf{p}_i}^{\perp_r}$;
- $\mathsf{SE}(\mathsf{p}_i)$ is the set of $\overline{\mathsf{p}_i\mathsf{p}_j}$ strictly between $\overline{\mathsf{p}_i}^{\perp_r}$ and $-\overline{\mathsf{p}_i\mathsf{p}_{i+1}}$;
- $\mathsf{S}(\mathsf{p}_i)$ is the set of $\overline{\mathsf{p}_i\mathsf{p}_j}$ equal to $-\overline{\mathsf{p}_i\mathsf{p}_{i+1}}$;
- $\mathsf{SW}(\mathsf{p}_i)$ is the set of $\overline{\mathsf{p}_i\mathsf{p}_j}$ strictly between $-\overline{\mathsf{p}_i\mathsf{p}_{i+1}}$ and$\overline{\mathsf{p}_i}^{\perp_\ell}$;
- $\mathsf{W}(\mathsf{p}_i)$ is the set of $\overline{\mathsf{p}_i\mathsf{p}_j}$ equal to $\overline{\mathsf{p}_i}^{\perp_\ell}$; and
- $\mathsf{NW}(\mathsf{p}_i)$ is the set of $\overline{\mathsf{p}_i\mathsf{p}_j}$ strictly between $\overline{\mathsf{p}_i}^{\perp_\ell}$ and $\overline{\mathsf{p}_i\mathsf{p}_{i+1}}$,

with $0 \leq j < i$ or $i < j < N$. Finally, $\mathsf{Eq}(\mathsf{p}_i)$ is the set of p_j that are equal to p_i. The *local carrier order of P in its vertex* p_i, for $0 \leq i < N$, denoted as $\mathsf{LCO}(P, \mathsf{p}_i)$, is the list of sets

$$\langle \mathsf{Eq}(\mathsf{p}_i), \mathsf{N}(\mathsf{p}_i), \mathsf{NE}(\mathsf{p}_i), \mathsf{E}(\mathsf{p}_i), \mathsf{SE}(\mathsf{p}_i), \mathsf{S}(\mathsf{p}_i), \mathsf{SW}(\mathsf{p}_i), \mathsf{W}(\mathsf{p}_i), \mathsf{NW}(\mathsf{p}_i) \rangle$$

and the *local carrier order of P* is the list

$$\langle \mathsf{LCO}(P, \mathsf{p}_0), \mathsf{LCO}(P, \mathsf{p}_1), ..., \mathsf{LCO}(P, \mathsf{p}_{N-1}) \rangle.$$

□

We remark that if $\mathsf{p}_j \in \mathsf{Eq}(\mathsf{p}_i)$, then the half-line $\overline{\mathsf{p}_i\mathsf{p}_j}$ makes no sense and therefore does not appear in any of the sets $\mathsf{N}(\mathsf{p}_i)$, ..., $\mathsf{NW}(\mathsf{p}_i)$.

As an illustration we use the polyline P depicted in Figure 3.3. Here, the local carrier orders in the vertices are given by:

- $\mathsf{LCO}(P, \mathsf{p}_0) = \langle\{\}, \{\overline{\mathsf{p}_0\mathsf{p}_1}\}, \{\overline{\mathsf{p}_0\mathsf{p}_2}, \overline{\mathsf{p}_0\mathsf{p}_4}\}, \{\}, \{\}, \{\}, \{\}, \{\}, \{\overline{\mathsf{p}_0\mathsf{p}_3}\}\rangle$
- $\mathsf{LCO}(P, \mathsf{p}_1) = \langle\{\}, \{\overline{\mathsf{p}_1\mathsf{p}_2}\}, \{\}, \{\}, \{\overline{\mathsf{p}_1\mathsf{p}_0}\}, \{\}, \{\}, \{\}, \{\overline{\mathsf{p}_1\mathsf{p}_3}, \overline{\mathsf{p}_1\mathsf{p}_4}\}\rangle$
- $\mathsf{LCO}(P, \mathsf{p}_2) = \langle\{\}, \{\overline{\mathsf{p}_2\mathsf{p}_3}\}, \{\overline{\mathsf{p}_2\mathsf{p}_4}\}, \{\}, \{\}, \{\}, \{\overline{\mathsf{p}_2\mathsf{p}_0}\}, \{\}, \{\overline{\mathsf{p}_2\mathsf{p}_1}\}\rangle$
- $\mathsf{LCO}(P, \mathsf{p}_3) = \langle\{\}, \{\overline{\mathsf{p}_3\mathsf{p}_4}\}, \{\overline{\mathsf{p}_3\mathsf{p}_2}\}, \{\}, \{\overline{\mathsf{p}_3\mathsf{p}_1}, \overline{\mathsf{p}_3\mathsf{p}_0}\}, \{\}, \{\}, \{\}, \{\}\rangle$

We now define the notion of *local-carrier-order similarity* of two polylines.

Definition 3.5. Let $P = \langle \mathsf{p}_0, \mathsf{p}_1, , \ldots, \mathsf{p}_N\rangle$ and $Q = \langle \mathsf{q}_0, \mathsf{q}_1, , \ldots, \mathsf{q}_N\rangle$ be polylines of equal size. We say that P and Q are *local-carrier-order similar* if $\mathsf{LCO}(P, \mathsf{p}_i) = \mathsf{LCO}(Q, \mathsf{q}_i)$ for all $i = 0, 1, ..., N-1$, that is, if $\mathsf{LCO}(P) = \mathsf{LCO}(Q)$ (always, modulo changing p_i in q_i) . □

3.3.2 An algebraic characterization of the local carrier order of a polyline

In this section, we give algebraic conditions to express the local carrier order of a polyline. Hereto, it suffices to give for each vertex p_i, with $0 \leq i < N$, in the polyline $P = \langle \mathsf{p}_0, \mathsf{p}_1, \ldots, \mathsf{p}_N\rangle$ characterizations of the sets in the list

$$\langle \mathsf{Eq}(\mathsf{p}_i), \mathsf{N}(\mathsf{p}_i), \mathsf{NE}(\mathsf{p}_i), \mathsf{E}(\mathsf{p}_i), \mathsf{SE}(\mathsf{p}_i), \mathsf{S}(\mathsf{p}_i), \mathsf{SW}(\mathsf{p}_i), \mathsf{W}(\mathsf{p}_i), \mathsf{NW}(\mathsf{p}_i)\rangle.$$

The following property gives this characterization. Obviously, the algebraic characterisation of $\mathsf{Eq}(\mathsf{p}_i)$ is given by equalities on the coordinates.

Property 3.10. Let $P = \langle(\mathsf{x}_0, \mathsf{y}_0), (\mathsf{x}_1, \mathsf{y}_1), , \ldots, (\mathsf{x}_N, \mathsf{y}_N)\rangle$ be a polyline and let $\mathsf{p}_i = (\mathsf{x}_i, \mathsf{y}_i)$, for $0 \leq i \leq N$. For $0 \leq j < i$ or $i < j < N$, the following table gives algebraic conditions for the halfline $\overline{\mathsf{p}_i\mathsf{p}_j}$ to belong to $\mathsf{X}(\mathsf{p}_i)$ with $\mathsf{X} \in \{\mathsf{N}, \mathsf{NE}, \mathsf{E}, \mathsf{SE}, \mathsf{S}, \mathsf{SW}, \mathsf{W}, \mathsf{NW}\}$:

$\mathsf{X} =$	$\overline{\mathsf{p}_i\mathsf{p}_j} \in \mathsf{X}(\mathsf{p}_i)$ is equivalent to
N	$-((\mathsf{x}_j - \mathsf{x}_i) \cdot (\mathsf{x}_{i+1} - \mathsf{x}_i) + (\mathsf{y}_j - \mathsf{y}_i) \cdot (\mathsf{y}_{i+1} - \mathsf{y}_i)) < 0$ and $-((\mathsf{x}_j - \mathsf{x}_i) \cdot (\mathsf{y}_{i+1} - \mathsf{y}_i) - (\mathsf{y}_j - \mathsf{y}_i) \cdot (\mathsf{x}_{i+1} - \mathsf{x}_i)) = 0$
NE	$-((\mathsf{x}_j - \mathsf{x}_i) \cdot (\mathsf{x}_{i+1} - \mathsf{x}_i) + (\mathsf{y}_j - \mathsf{y}_i) \cdot (\mathsf{y}_{i+1} - \mathsf{y}_i)) < 0$ and $-((\mathsf{x}_j - \mathsf{x}_i) \cdot (\mathsf{y}_{i+1} - \mathsf{y}_i) - (\mathsf{y}_j - \mathsf{y}_i) \cdot (\mathsf{x}_{i+1} - \mathsf{x}_i)) < 0$
E	$-((\mathsf{x}_j - \mathsf{x}_i) \cdot (\mathsf{x}_{i+1} - \mathsf{x}_i) + (\mathsf{y}_j - \mathsf{y}_i) \cdot (\mathsf{y}_{i+1} - \mathsf{y}_i)) = 0$ and $-((\mathsf{x}_j - \mathsf{x}_i) \cdot (\mathsf{y}_{i+1} - \mathsf{y}_i) - (\mathsf{y}_j - \mathsf{y}_i) \cdot (\mathsf{x}_{i+1} - \mathsf{x}_i)) < 0$
SE	$-((\mathsf{x}_j - \mathsf{x}_i) \cdot (\mathsf{x}_{i+1} - \mathsf{x}_i) + (\mathsf{y}_j - \mathsf{y}_i) \cdot (\mathsf{y}_{i+1} - \mathsf{y}_i)) > 0$ and $-((\mathsf{x}_j - \mathsf{x}_i) \cdot (\mathsf{y}_{i+1} - \mathsf{y}_i) - (\mathsf{y}_j - \mathsf{y}_i) \cdot (\mathsf{x}_{i+1} - \mathsf{x}_i)) < 0$
S	$-((\mathsf{x}_j - \mathsf{x}_i) \cdot (\mathsf{x}_{i+1} - \mathsf{x}_i) + (\mathsf{y}_j - \mathsf{y}_i) \cdot (\mathsf{y}_{i+1} - \mathsf{y}_i)) > 0$ and $-((\mathsf{x}_j - \mathsf{x}_i) \cdot (\mathsf{y}_{i+1} - \mathsf{y}_i) - (\mathsf{y}_j - \mathsf{y}_i) \cdot (\mathsf{x}_{i+1} - \mathsf{x}_i)) = 0$
SW	$-((\mathsf{x}_j - \mathsf{x}_i) \cdot (\mathsf{x}_{i+1} - \mathsf{x}_i) + (\mathsf{y}_j - \mathsf{y}_i) \cdot (\mathsf{y}_{i+1} - \mathsf{y}_i)) > 0$ and $-((\mathsf{x}_j - \mathsf{x}_i) \cdot (\mathsf{y}_{i+1} - \mathsf{y}_i) - (\mathsf{y}_j - \mathsf{y}_i) \cdot (\mathsf{x}_{i+1} - \mathsf{x}_i)) > 0$
W	$-((\mathsf{x}_j - \mathsf{x}_i) \cdot (\mathsf{x}_{i+1} - \mathsf{x}_i) + (\mathsf{y}_j - \mathsf{y}_i) \cdot (\mathsf{y}_{i+1} - \mathsf{y}_i)) = 0$ and $-((\mathsf{x}_j - \mathsf{x}_i) \cdot (\mathsf{y}_{i+1} - \mathsf{y}_i) - (\mathsf{y}_j - \mathsf{y}_i) \cdot (\mathsf{x}_{i+1} - \mathsf{x}_i)) > 0$
NW	$-((\mathsf{x}_j - \mathsf{x}_i) \cdot (\mathsf{x}_{i+1} - \mathsf{x}_i) + (\mathsf{y}_j - \mathsf{y}_i) \cdot (\mathsf{y}_{i+1} - \mathsf{y}_i)) < 0$ and $-((\mathsf{x}_j - \mathsf{x}_i) \cdot (\mathsf{y}_{i+1} - \mathsf{y}_i) - (\mathsf{y}_j - \mathsf{y}_i) \cdot (\mathsf{x}_{i+1} - \mathsf{x}_i)) > 0$

□

The proof of this property uses the same algebraic tools as the proof of Theorem 3.2 and we will skip the (straightforward) details.

3.3.3 A characterization of double-cross similarity of polylines in terms of their local carrier order

In this section, we give a geometric characterization of double-cross similarity of polylines in terms of their local carrier orders. The main theorem that we prove in this section is the following.

Theorem 3.6. *Let P and Q be polylines of equal size. Then, P and Q are double-cross similar if and only if they are local-carrier-order similar. That is*

$$\mathsf{DCM}(P) = \mathsf{DCM}(Q) \quad \textit{if and only if} \quad \mathsf{LCO}(P) = \mathsf{LCO}(Q).$$

The two directions of this theorem are proven in Lemma 3.7 and Lemma 3.9 (or, equivalently, in Corollary 3.8 and Corollary 3.10).

Lemma 3.7. Let $P = \langle \mathsf{p}_0, \mathsf{p}_1, , \ldots, \mathsf{p}_N \rangle$ be a polyline. For i, j with $0 \leq i \leq j < N$, we can derive the value of the 4-tuple $\mathsf{DCM}(P)[i,j] = (C_1\ C_2\ C_3\ C_4)$ from $\mathsf{LCO}(P, \mathsf{p}_i)$ and $\mathsf{LCO}(P, \mathsf{p}_j)$.

Proof. Let $P = \langle \mathsf{p}_0, \mathsf{p}_1, , \ldots, \mathsf{p}_N \rangle$ be a polyline of size N. If $\mathsf{p}_j \in \mathsf{Eq}(\mathsf{p}_i)$, then $\mathsf{DCM}(P)[i,j] = (0\ 0\ 0\ 0)$. This is in particular true if $i = j$.

If $\mathsf{p}_j \notin \mathsf{Eq}(\mathsf{p}_i)$, then the following twelve easily observable facts show how to determine C_1, C_2, C_3 and C_4 (for instance, by inspecting Figure 2.9).

C_1	is equivalent to
0	$\overline{\mathsf{p}_i\mathsf{p}_j} \in \mathsf{W}(\mathsf{p}_i) \cup \mathsf{E}(\mathsf{p}_i)$
+	$\overline{\mathsf{p}_i\mathsf{p}_j} \in \mathsf{SE}(\mathsf{p}_i) \cup \mathsf{S}(\mathsf{p}_i) \cup \mathsf{SW}(\mathsf{p}_i)$
−	$\overline{\mathsf{p}_i\mathsf{p}_j} \in \mathsf{NW}(\mathsf{p}_i) \cup \mathsf{N}(\mathsf{p}_i) \cup \mathsf{NE}(\mathsf{p}_i)$

C_2	is equivalent to
0	$\overline{\mathsf{p}_j\mathsf{p}_i} \in \mathsf{W}(\mathsf{p}_j) \cup \mathsf{E}(\mathsf{p}_j)$
+	$\overline{\mathsf{p}_j\mathsf{p}_i} \in \mathsf{SE}(\mathsf{p}_j) \cup \mathsf{S}(\mathsf{p}_j) \cup \mathsf{SW}(\mathsf{p}_j)$
−	$\overline{\mathsf{p}_i\mathsf{p}_j} \in \mathsf{NW}(\mathsf{p}_i) \cup \mathsf{N}(\mathsf{p}_i) \cup \mathsf{NE}(\mathsf{p}_i)$

C_3	is equivalent to
0	$\overline{\mathsf{p}_i\mathsf{p}_j} \in \mathsf{N}(\mathsf{p}_i) \cup \mathsf{S}(\mathsf{p}_i)$
+	$\overline{\mathsf{p}_i\mathsf{p}_j} \in \mathsf{SW}(\mathsf{p}_i) \cup \mathsf{W}(\mathsf{p}_i) \cup \mathsf{NW}(\mathsf{p}_i)$
−	$\overline{\mathsf{p}_i\mathsf{p}_j} \in \mathsf{NE}(\mathsf{p}_i) \cup \mathsf{E}(\mathsf{p}_i) \cup \mathsf{SE}(\mathsf{p}_i)$

C_4	is equivalent to
0	$\overline{\mathsf{p}_i\mathsf{p}_j} \in \mathsf{N}(\mathsf{p}_j) \cup \mathsf{S}(\mathsf{p}_j)$
+	$\overline{\mathsf{p}_i\mathsf{p}_j} \in \mathsf{SW}(\mathsf{p}_j) \cup \mathsf{W}(\mathsf{p}_j) \cup \mathsf{NW}(\mathsf{p}_j)$
−	$\overline{\mathsf{p}_i\mathsf{p}_j} \in \mathsf{NE}(\mathsf{p}_i) \cup \mathsf{E}(\mathsf{p}_i) \cup \mathsf{SE}(\mathsf{p}_i)$

This concludes the proof. □

This lemma has the following immediate corollary.

Corollary 3.8. Let P be a polyline in $\mathbf{R}^2$. Then, the matrix $\mathsf{DCM}(P)$ can be reconstructed from the local carrier order $\mathsf{LCO}(P)$.

Proof. Properties 3.2 and 3.3 show that it is sufficient to know a double-cross matrix of a polyline on and above its diagonal in order to complete it below

its diagonal. And Lemma 3.7 shows how the local carrier order gives the double-cross matrix on and above its diagonal. This concludes the proof. □

Now, we turn to the other implication of Theorem 3.6.

Lemma 3.9. Let $P = \langle \mathsf{p}_0, \mathsf{p}_1, , \ldots, \mathsf{p}_N \rangle$ be a polyline in $\mathbf{R}^2$ of size N. If $0 \leq i < j < N$, then $\mathsf{DCM}(P)[i,j]$ contains enough information to derived to which set of $\mathsf{LCO}(P, \mathsf{p}_i)$ the half-lines $\overline{\mathsf{p}_i\mathsf{p}_j}$ belong and to which set of $\mathsf{LCO}(P, \mathsf{p}_j)$ the half-lines $\overline{\mathsf{p}_j\mathsf{p}_i}$ belong.

Proof. Let $\mathsf{DCM}(P)[i,j] = (C_1\ C_2\ C_3\ C_4)$, for $0 \leq i < j < N$. Again, the following facts are easily observable (for instance, by inspecting Figure 2.9).

C_1	C_3	is equivalent to
$-$	$-$	$\overline{\mathsf{p}_i\mathsf{p}_j} \in \mathsf{NE}(\mathsf{p}_i)$
$-$	0	$\overline{\mathsf{p}_i\mathsf{p}_j} \in \mathsf{N}(\mathsf{p}_i)$
$-$	$+$	$\overline{\mathsf{p}_i\mathsf{p}_j} \in \mathsf{NW}(\mathsf{p}_i)$
0	$-$	$\overline{\mathsf{p}_i\mathsf{p}_j} \in \mathsf{E}(\mathsf{p}_i)$
0	0	$\mathsf{p}_i = \mathsf{p}_{i+1}$ is excluded
0	$+$	$\overline{\mathsf{p}_i\mathsf{p}_j} \in \mathsf{W}(\mathsf{p}_i)$
$+$	$-$	$\overline{\mathsf{p}_i\mathsf{p}_j} \in \mathsf{SE}(\mathsf{p}_i)$
$+$	0	$\overline{\mathsf{p}_i\mathsf{p}_j} \in \mathsf{S}(\mathsf{p}_i)$
$+$	$+$	$\overline{\mathsf{p}_i\mathsf{p}_j} \in \mathsf{SW}(\mathsf{p}_i)$

C_2	C_4	is equivalent to
$-$	$-$	$\overline{\mathsf{p}_j\mathsf{p}_i} \in \mathsf{NE}(\mathsf{p}_j)$
$-$	0	$\overline{\mathsf{p}_j\mathsf{p}_i} \in \mathsf{N}(\mathsf{p}_j)$
$-$	$+$	$\overline{\mathsf{p}_j\mathsf{p}_i} \in \mathsf{NW}(\mathsf{p}_j)$
0	$-$	$\overline{\mathsf{p}_j\mathsf{p}_i} \in \mathsf{E}(\mathsf{p}_j)$
0	0	$\mathsf{p}_j = \mathsf{p}_{j+1}$ is excluded
0	$+$	$\overline{\mathsf{p}_j\mathsf{p}_i} \in \mathsf{W}(\mathsf{p}_j)$
$+$	$-$	$\overline{\mathsf{p}_j\mathsf{p}_i} \in \mathsf{SE}(\mathsf{p}_j)$
$+$	0	$\overline{\mathsf{p}_j\mathsf{p}_i} \in \mathsf{S}(\mathsf{p}_j)$
$+$	$+$	$\overline{\mathsf{p}_j\mathsf{p}_i} \in \mathsf{SW}(\mathsf{p}_j)$

This concludes the proof. □

This lemma has the following immediate corollary.

Corollary 3.10. Given $\mathsf{DCM}(P)$, $\mathsf{LCO}(P, \mathsf{p}_i)$ can be constructed for all $0 \leq i < N$. □

Combined, Corollaries 3.8 and 3.10 prove Theorem 3.6.

3.4 A characterization of the double-cross invariant transformations of the plane

In this section, we identify the transformations[3] of the plane $\mathbf{R}^2$ that leave the double-cross matrix of all polylines invariant.

If $\alpha : \mathbf{R}^2 \to \mathbf{R}^2$ is a transformation and if p and q are points in $\mathbf{R}^2$, then we write $\alpha(\overrightarrow{\mathsf{pq}})$ for $\overrightarrow{\alpha(\mathsf{p})\alpha(\mathsf{q})}$.

What do we mean by applying a transformation of the plane to a polyline? The following definition says that we mean it to be the polyline formed by the transformed vertices.

[3] A transformation is a continuous, bijective mapping of the plane $\mathbf{R}^2$ onto itself.

Definition 3.11. Let $\alpha : \mathbf{R}^2 \to \mathbf{R}^2$ be a transformation. Let $P = \langle(\mathsf{x}_0, \mathsf{y}_0), (\mathsf{x}_1, \mathsf{y}_1), , \ldots, (\mathsf{x}_N, \mathsf{y}_N)\rangle$ be a polyline. We define $\alpha(P)$ to be the polyline $\langle\alpha(\mathsf{x}_0, \mathsf{y}_0), \alpha(\mathsf{x}_1, \mathsf{y}_1), , \ldots, \alpha(\mathsf{x}_N, \mathsf{y}_N)\rangle$. □

We remark that since a transformation α is a bijective function, Assumption 1, which says that no two consecutive vertices of a polyline are identical, will hold for $\alpha(P)$ if it holds for the polyline P.

We now define the notion of double-cross invariant transformation of the plane.

Definition 3.12. Let $\alpha : \mathbf{R}^2 \to \mathbf{R}^2$ be a transformation. Let P be a polyline. We say that α *leaves* P *invariant* if P and $\alpha(P)$ are double-cross similar, that is, if $\mathsf{DCM}(P) = \mathsf{DCM}(\alpha(P))$.

We say that α is a *double-cross invariant* transformation if it leaves all polylines invariant. A group of transformations of $\mathbf{R}^2$ is *double-cross invariant* if all its members are double-cross invariant transformations. □

The main aim of this section is to prove the following theorem, which says that the largest group of transformations that is double-cross invariant consists of the translations, rotations and homotecies (scalings)[4] of the plane. The elements of this group are called the *similarities* of $\mathbf{R}^2$.

Theorem 3.13. *The largest group of transformations of* $\mathbf{R}^2$*, that is double-cross invariant consist of the* similarity transformations *of the plane onto itself, that is, transformations of the form*

$$\alpha : \mathbf{R}^2 \to \mathbf{R}^2 : \begin{pmatrix} x \\ y \end{pmatrix} \mapsto \mathsf{c} \cdot \begin{pmatrix} \mathsf{a} & -\mathsf{b} \\ \mathsf{b} & \mathsf{a} \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \mathsf{d} \\ \mathsf{e} \end{pmatrix},$$

where $\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{e} \in \mathbf{R}$*,* $\mathsf{c} \neq 0$ *and* $\mathsf{a}^2 + \mathsf{b}^2 = 1$.

We remark that the condition $\mathsf{a}^2 + \mathsf{b}^2 = 1$ implies that a and b cannot be both zero. In fact, we can see a as $\cos\varphi$ and b as $\sin\varphi$, where φ is the angle of the rotation expressed by the matrix.

We prove this theorem by proving three lemmas. Lemma 3.14 proves *soundness* and Lemma 3.16 proves *completeness*. Lemma 3.15 is a purely technical lemma.

Lemma 3.14. The translations, rotations and homotecies of the plane (that is, the transformations given in Theorem 3.2) are double-cross invariant transformations.

[4] A homotecy of the plane is a transformation of the form $\alpha_{\mathsf{c}} : \mathbf{R}^2 \to \mathbf{R}^2 : (x, y) \mapsto \mathsf{c} \cdot (x, y)$, where $\mathsf{c} \neq 0$.

Proof. We consider the three types of transformations separately, since we can apply them one after the other. In all cases, we use the algebraic characterization, given by Theorem 3.2.

1. Translations. We have already remarked that all the factors appearing in the algebraic expressions, given by given by Theorem 3.2, that is $(\mathsf{x}_j - \mathsf{x}_i)$, $(\mathsf{x}_{i+1} - \mathsf{x}_i)$, $(\mathsf{y}_j - \mathsf{y}_i)$, $(\mathsf{y}_{i+1} - \mathsf{y}_i)$, $(\mathsf{x}_{j+1} - \mathsf{x}_j)$ and $(\mathsf{y}_{j+1} - \mathsf{y}_j)$ are differences in x-coordinates or differences in y-coordinates. A translation $\tau_{(\mathsf{d},\mathsf{e})} : \mathbf{R}^2 \to \mathbf{R}^2 : (x, y) \mapsto (x + \mathsf{d}, y + \mathsf{e})$, therefore leaves these differences unaltered. For instance, $(\mathsf{x}_j - \mathsf{x}_i)$ is transformed to $(\mathsf{x}_j + \mathsf{d} - (\mathsf{x}_i + \mathsf{d}))$, which is, of course, the original value $(\mathsf{x}_j - \mathsf{x}_i)$. None of the expressions given by Theorem 3.2 are therefore changed and the double-cross conditions remain the same.

2. Rotations. Let

$$\rho_{(\mathsf{a},\mathsf{b})} : \mathbf{R}^2 \to \mathbf{R}^2 : \begin{pmatrix} x \\ y \end{pmatrix} \mapsto \begin{pmatrix} \mathsf{a} & -\mathsf{b} \\ \mathsf{b} & \mathsf{a} \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix},$$

with $\mathsf{a}^2 + \mathsf{b}^2 = 1$, be a rotation (that fixes the origin).

The expression for C_1 is transformed to

$$\begin{aligned}(\mathsf{a} \cdot (\mathsf{x}_j - \mathsf{x}_i) - \mathsf{b} \cdot (\mathsf{y}_j - \mathsf{y}_i)) \cdot (\mathsf{a} \cdot (\mathsf{x}_{i+1} - \mathsf{x}_i) - \mathsf{b} \cdot (\mathsf{y}_{i+1} - \mathsf{y}_i)) + \\ (\mathsf{b} \cdot (\mathsf{x}_j - \mathsf{x}_i) + \mathsf{a} \cdot (\mathsf{y}_j - \mathsf{y}_i))) \cdot (\mathsf{b} \cdot (\mathsf{x}_{i+1} - \mathsf{x}_i) + \mathsf{a} \cdot (\mathsf{y}_{i+1} - \mathsf{y}_i)),\end{aligned}$$

which simplifies to $(\mathsf{a}^2 + \mathsf{b}^2) \cdot ((\mathsf{x}_j - \mathsf{x}_i) \cdot (\mathsf{x}_{i+1} - \mathsf{x}_i) + (\mathsf{y}_j - \mathsf{y}_i) \cdot (\mathsf{y}_{i+1} - \mathsf{y}_i))$, which is the original polynomial since $\mathsf{a}^2 + \mathsf{b}^2 = 1$. For C_2, C_3 and C_4, a similar straightforward computation shows that the polynomials remain the same.

3. Homotecies. A homotecy $\alpha_{\mathsf{c}} : \mathbf{R}^2 \to \mathbf{R}^2 : (x, y) \mapsto \mathsf{c} \cdot (x, y)$, transforms the differences $(\mathsf{x}_j - \mathsf{x}_i)$, $(\mathsf{x}_{i+1} - \mathsf{x}_i)$, $(\mathsf{y}_j - \mathsf{y}_i)$, $(\mathsf{y}_{i+1} - \mathsf{y}_i)$, $(\mathsf{x}_{j+1} - \mathsf{x}_j)$ and $(\mathsf{y}_{j+1} - \mathsf{y}_j)$ to $(\mathsf{c} \cdot \mathsf{x}_j - \mathsf{c} \cdot \mathsf{x}_i)$, $(\mathsf{c} \cdot \mathsf{x}_{i+1} - \mathsf{c} \cdot \mathsf{x}_i)$, $(\mathsf{c} \cdot \mathsf{y}_j - \mathsf{c} \cdot \mathsf{y}_i)$, $(\mathsf{c} \cdot \mathsf{y}_{i+1} - \mathsf{c} \cdot \mathsf{y}_i)$, $(\mathsf{c} \cdot \mathsf{x}_{j+1} - \mathsf{c} \cdot \mathsf{x}_j)$ and $(\mathsf{c} \cdot \mathsf{y}_{j+1} - \mathsf{c} \cdot \mathsf{y}_j)$. This means that the polynomials given by Theorem 3.2 are multiplied by c^2, which is strictly larger than zero, for $\mathsf{c} \neq 0$. The signs of these polynomials are therefore unaltered. And the double-cross value of the scaled polyline is the same as the original one. □

Before we can turn to completeness, we need the following technical lemma.

Lemma 3.15. Let $f : \mathbf{R} \to \mathbf{R} : t \mapsto f(t)$ be a strictly monotone increasing function. If

$$f(\frac{s+t}{2}) = \frac{f(s) + f(t)}{2}$$

for any $s, t \in \mathbf{R}$, then $f(t) = (f(1) - f(0)) \cdot t + f(0)$.

Proof. Suppose that f is a function as described and suppose that there is a $t_0 \in \mathbf{R}$ such that $f(t_0) \neq (f(1) - f(0)) \cdot t_0 + f(0)$. We remark that therefore t_0 cannot be 0 or 1.

If $f(-t_0) = (f(1) - f(0)) \cdot (-t_0) + f(0)$, then it follows from $2 \cdot f(0) = 2 \cdot f(\frac{t_0 - t_0}{2}) = f(t_0) + f(-t_0)$ that also $f(t_0) = (f(1) - f(0)) \cdot t_0 + f(0)$. We may therefore assume $0 < t_0$.

If $f(\frac{t_0}{2}) = (f(1) - f(0)) \cdot (\frac{t_0}{2}) + f(0)$, then it follows from $2 \cdot f(\frac{0+t_0}{2}) = f(0) + f(t_0)$ that also $f(t_0) = (f(1) - f(0)) \cdot t_0 + f(0)$. We may therefore assume $0 < t_0 < 1$.

Claim. For any $n \in \mathbf{N}$ and any k, with $0 \leq k \leq 2^n$, we have that

$$f(\frac{k}{2^n}) = (f(1) - f(0)) \cdot \frac{k}{2^n} + f(0).$$

We first prove this claim (by induction on n). For $n = 0$, and $k = 0, 1$, we respectively have $f(0) = (f(1) - f(0)) \cdot 0 + f(0)$ and $f(1) = (f(1) - f(0)) \cdot 1 + f(0)$.

Assume now that the claim is true for n. We prove it holds for $n + 1$. We consider $\frac{k}{2^{n+1}}$ and distinguish between the cases, $0 \leq k \leq 2^n$ and $k = k' + 2^n$ with $0 < k' \leq 2^n$. If $0 \leq k \leq 2^n$, then $f(\frac{k}{2^{n+1}}) = f(\frac{1}{2}(0 + \frac{k}{2^n})) = \frac{1}{2}(f(0) + f(\frac{k}{2^n}))$, which by the induction hypothesis equals $\frac{1}{2}(f(0) + (f(1) - f(0)) \cdot \frac{k}{2^n} + f(0))$ or $(f(1) - f(0)) \cdot \frac{k}{2^{n+1}} + f(0)$.

If $k = k' + 2^n$ with $0 < k' \leq 2^n$, then $f(\frac{2^n + k'}{2^{n+1}}) = f(\frac{1}{2}(1 + \frac{k'}{2^n})) = \frac{1}{2}(f(1) + f(\frac{k'}{2^n}))$, which by the induction hypothesis equals $\frac{1}{2}(f(1) + (f(1) - f(0)) \cdot \frac{k'}{2^n} + f(0))$ which equals $\frac{1}{2}(f(1) - f(0)) + (f(1) - f(0)) \cdot \frac{k'}{2^{n+1}} + f(0)$ or $(f(1) - f(0)) \cdot \frac{k' + 2^n}{2^{n+1}} + f(0)$ which is $(f(1) - f(0)) \cdot \frac{k}{2^{n+1}} + f(0)$. This concludes the proof of the claim. □

To conclude the proof, let $0 < t_0 < 1$ and assume first that $f(t_0) > (f(1) - f(0)) \cdot t_0 + f(0)$. This means that $t_0 < \frac{f(t_0) - f(0)}{f(1) - f(0)}$. We remark that since f is assumed to be strictly monotone, $f(1) - f(0) \neq 0$ and therefore the division is allowed. Choose k and n such that

$$\frac{k}{2^n} \leq t_0 < \frac{k+1}{2^n} < \frac{f(t_0) - f(0)}{f(1) - f(0)},$$

with $0 \leq k \leq 2^n$. Then $f(\frac{k+1}{2^n}) = (f(1) - f(0)) \cdot \frac{k+1}{2^n} + f(0) < f(t_0)$, although $t_0 < \frac{k+1}{2^n}$, which contradicts the fact that f is strictly monotone increasing.

If we assume $f(t_0) < (f(1) - f(0)) \cdot t_0 + f(0)$ on the other hand, we have $\frac{f(t_0) - f(0)}{f(1) - f(0)} < t_0$. Choose k and n such that

$$\frac{f(t_0) - f(0)}{f(1) - f(0)} < \frac{k}{2^n} < t_0 \leq \frac{k+1}{2^n},$$

with $0 \leq k \leq 2^n$. Then $f(\frac{k}{2^n}) = (f(1) - f(0)) \cdot \frac{k}{2^n} + f(0) > f(t_0)$, although $\frac{k}{2^n} < t_0$, which contradicts the fact that f is strictly monotone increasing. In both cases, we obtain a contradiction and this concludes the proof. □

The following lemma proves completeness.

Lemma 3.16. The similarity transformations of the plane (given in Theorem 3.2) are the only double-cross invariant transformations.

Proof. Let $\alpha : \mathbf{R}^2 \to \mathbf{R}^2$ be a double-cross invariant transformation.

(1) We consider polylines $P = \langle \mathsf{p}_0, \mathsf{p}_1, \mathsf{p}_2 \rangle$, where p_0, p_1 and p_2 are collinear points with p_1 between p_0 and p_2. By Assumption 1, p_1 should be *strictly* between p_0 and p_2 The only relevant entry in the double-cross matrix of this polyline is $\mathsf{DC}(\overrightarrow{\mathsf{p}_0\mathsf{p}_1}, \overrightarrow{\mathsf{p}_1\mathsf{p}_2})$ which is $(-\ +\ 0\ 0)$. In $\alpha(P)$, $\mathsf{DC}(\alpha(\overrightarrow{\mathsf{p}_0\mathsf{p}_1}), \alpha(\overrightarrow{\mathsf{p}_1\mathsf{p}_2}))$ should also be $(-\ +\ 0\ 0)$. This implies that $\alpha(\mathsf{p}_0)$, $\alpha(\mathsf{p}_1)$ and $\alpha(\mathsf{p}_2)$ should also be collinear, with $\alpha(\mathsf{p}_1)$ (strictly) between $\alpha(\mathsf{p}_0)$ and $\alpha(\mathsf{p}_2)$. This means that α preserves *collinearity* and *betweenness*.

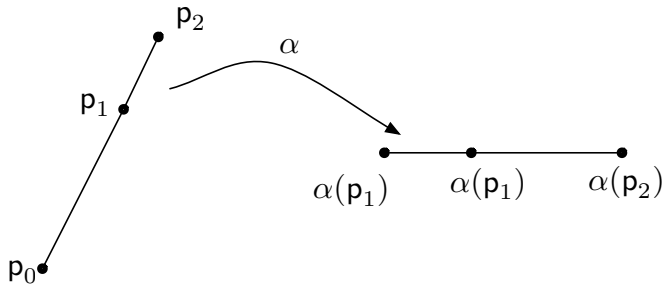


Figure 3.4: A collinearity and betweenness-preserving transformation of the plane.

(2) We consider polylines $P = \langle \mathsf{p}_0, \mathsf{p}_1, \mathsf{p}_2 \rangle$, where $\angle(\overrightarrow{\mathsf{p}_1\mathsf{p}_0}, \overrightarrow{\mathsf{p}_1\mathsf{p}_2}) = 90°$, that is, the polyline takes a right turn at p_1. The only relevant entry in the double-cross matrix of this polyline is again $\mathsf{DC}(\overrightarrow{\mathsf{p}_0\mathsf{p}_1}, \overrightarrow{\mathsf{p}_1\mathsf{p}_2})$ which is now $(-\ 0\ 0\ -)$. In $\alpha(P)$, $\mathsf{DC}(\alpha(\overrightarrow{\mathsf{p}_0\mathsf{p}_1}), \alpha(\overrightarrow{\mathsf{p}_1\mathsf{p}_2}))$ should also be $(-\ 0\ 0\ -)$. This means that α is a *right-turn-preserving* transformation. This is illustrated in Figure 3.5. Similarly, α is a *left-turn-preserving* transformation.

(3) We consider the polyline $P = \langle \mathsf{p}_0, \mathsf{p}_1, \mathsf{p}_2, \mathsf{p}_3, \mathsf{p}_4, \mathsf{p}_5, \mathsf{p}_6, \mathsf{p}_7, \mathsf{p}_8, \mathsf{p}_9, \mathsf{p}_{10}, \mathsf{p}_{11}, \mathsf{p}_{12} \rangle$, with $\mathsf{p}_0 = \mathsf{p}_4 = \mathsf{p}_7 = \mathsf{p}_{12}(0,0)$, $\mathsf{p}_1 = \mathsf{p}_5 = (0,1)$, $\mathsf{p}_2 = \mathsf{p}_{10} = (1,1)$, $\mathsf{p}_3 = \mathsf{p}_8 = (1,0)$ and $\mathsf{p}_6 = \mathsf{p}_9 = (\frac{1}{2}, \frac{1}{2})$, depicted in Figure 3.6. This polyline forms a square with its two diagonals after making six 90° right-turns and one 90° left-turn. It is also closed in the sense that its start and end vertex are

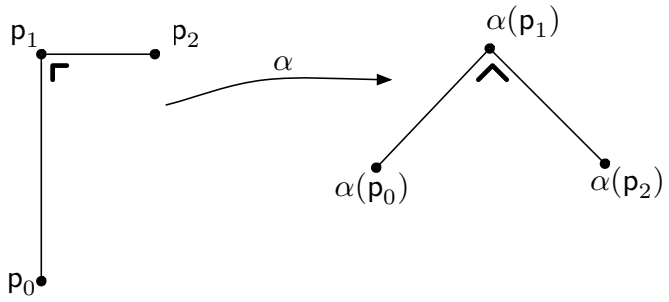


Figure 3.5: A right-turn-preserving transformation of the plane.

equal. The transformation α, which according to (2) preserves right and left turns, therefore has to map P again to a square with its diagonals. This means α is a *square-preserving* transformation. In particular, α preserves parallel line segments.

In P, coming from p_5, we have a 90° right-turn at p_6 to move to p_7. This means that in $\alpha(P)$, coming from $\alpha(\mathsf{p}_5)$, we must also have a 90° right-turn at $\alpha(\mathsf{p}_6)$ to move to $\alpha(\mathsf{p}_7)$. The possible locations of $\alpha(\mathsf{p}_6)$ inside the square determined by $\alpha(\mathsf{p}_0)$, $\alpha(\mathsf{p}_1)$, $\alpha(\mathsf{p}_2)$ and $\alpha(\mathsf{p}_3)$ that satisfy this requirement form a half-circle that is based on the line segment connecting $\alpha(\mathsf{p}_0)$ and $\alpha(\mathsf{p}_1)$ and that passes through the "centre" of the square.

Also in P, coming from p_8, we have a 90° right-turn at $\mathsf{p}_9 = \mathsf{p}_6$ to move to p_{10}. This means that in $\alpha(P)$, coming from $\alpha(\mathsf{p}_8)$, we must also have a 90° right-turn at $\alpha(\mathsf{p}_9) = \alpha(\mathsf{p}_6)$ to move to $\alpha(\mathsf{p}_{10})$. The possible locations of $\alpha(\mathsf{p}_6)$ inside the square determined by $\alpha(\mathsf{p}_0)$, $\alpha(\mathsf{p}_1)$, $\alpha(\mathsf{p}_2)$ and $\alpha(\mathsf{p}_3)$ that satisfy this requirement form a half-circle that is based on the line segment connecting $\alpha(\mathsf{p}_2)$ and $\alpha(\mathsf{p}_3)$ and that passes through the "centre" of the square.

This implies that there is only one location to satisfy both restrictions simultaneously, namely the unique intersection point of these two half circles. This implies that $\alpha(\mathsf{p}_6)$ should be the "centre" of the square determined by $\alpha(\mathsf{p}_0)$, $\alpha(\mathsf{p}_1)$, $\alpha(\mathsf{p}_2)$ and $\alpha(\mathsf{p}_3)$. This implies that, p_6, which is the midpoint between p_0 and p_2 is mapped to $\alpha(\mathsf{p}_6)$, which should be the midpoint between $\alpha(\mathsf{p}_0)$ and $\alpha(\mathsf{p}_2)$. This means α is also a *midpoint-preserving* transformation.

Suppose $\alpha(0,0) = (a,b)$. If $\tau_{(-a,-b)}$ is the translation $(x,y) \mapsto (x-a, y-b)$, then $\tau_{(-a,-b)} \circ \alpha(0,0) = (0,0)$. Suppose $\tau_{(-a,-b)} \circ \alpha(1,0) = (c,d)$. Let $\rho_{(c,d)}$ be the rotation with $(0,0)$ as center that brings (c,d) to the positive x-axis, that is, to $(\sqrt{c^2+d^2}, 0)$. We remark that (c,d) cannot be the origin since α is assumed to be a bijective function. So, also $\tau_{(-a,-b)} \circ \alpha$ is bijective.

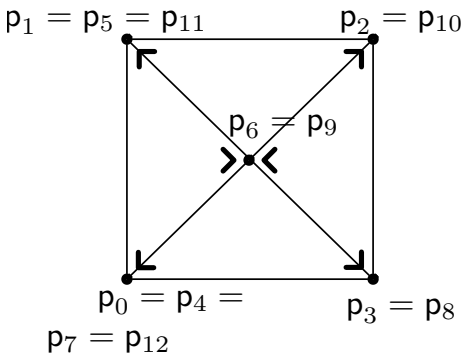


Figure 3.6: A polyline that is a square with its two diagonals. The six 90° right-turns are indicated in bold.

Furthermore, let $\sigma_{\sqrt{c^2+d^2}}$ be the scaling $(x, y) \mapsto \frac{1}{\sqrt{c^2+d^2}}(x, y)$ and let

$$\beta = \sigma_{\sqrt{c^2+d^2}} \circ \rho_{(c,d)} \circ \tau_{(-a,-b)} \circ \alpha.$$

Then we have that $\beta(0, 0) = (0, 0)$ and $\beta(1, 0) = (1, 0)$.

Since α is a double-cross invariant transformation, by assumption, and since $\sigma_{\sqrt{c^2+d^2}}$, $\rho_{(c,d)}$ and $\tau_{(-a,-b)}$ are double-cross invariant transformations by Lemma 3.14, also β is a double-cross invariant transformation. And β also inherits from α the properties of preserving betweenness, collinearity, right- and left turns, squares, parallel line segments and midpoints. Because β preserves squares, we also have $\beta(0, 1) = (0, 1)$.

We now claim the following.

Claim: The transformation β is the identity.

The proof of this claim finishes the proof. Indeed, then we have

$$\alpha = \sigma^{-1}_{\sqrt{c^2+d^2}} \circ \rho^{-1}_{(c,d)} \circ \tau^{-1}_{(-a,-b)},$$

which is of the required form.

Proof of the claim: First, we show that β is the identity on the x-axis and next we do the same for all lines perpendicular to the x-axis. Hereto, we consider the function

$$\beta_x : \mathbf{R} \to \mathbf{R} : x \mapsto \beta_x(x) := \pi_x(\beta(x, 0)),$$

where π_x is the projection on the first component, that is, $\pi_x(x, y) := x$. Since $\beta(0, 0) = (0, 0)$ and $\beta(1, 0) = (1, 0)$ and β preserves collinearity, β maps the x-axis onto the x-axis and we have $\beta_x(0) = 0$ and $\beta_x(1) = 1$. Furthermore, since β and hence β_x preserves betweenness, β_x is strictly monotone increasing.

Indeed, let $s, t \in \mathbf{R}$ with $s < t$. With respect to 0 and 1, we can consider the twelve possible locations of s and t: $s < t < 0$; $s < t = 0 < 1$; $s < 0 < t < 1$; $s < 0 < t = 1$; $s < 0 < 1 < t$; $s = 0 < t < 1$; $s = 0 < t = 1$; $s = 0 < 1 < t$; $0 < s < t = 1$; $0 < s < 1 < t$; $0 < s = 1 < t$; and $0 < 1 < s < t$. In all cases, except $s = 0 < t = 1$, we have three points. So, here we can use the fact that β preserves betweenness to show that $\beta_x(s) < \beta_x(t)$. In the case $s = 0 < t = 1$, we have $\beta_x(s) = \beta_x(0) = 0 < 1 = \beta_x(1) = \beta_x(t)$. Finally, since β preserves midpoints, also for β_x, we have

$$\beta_x(\frac{s+t}{2}) = \frac{\beta_x(s) + \beta_x(t)}{2},$$

for all $s, t \in \mathbf{R}$. All the conditions to apply Lemma 3.15 are therefore fulfilled. And we get $\beta_x(x) = (\beta_x(1) - \beta_x(0)) \cdot x + \beta_x(0) = (1 - 0) \cdot x + 0 = x$.

Now, we fix some $\mathsf{x}_0 \in \mathbf{R}$ and consider the function

$$\beta_{\mathsf{x}_0,y} : \mathbf{R} \to \mathbf{R} : y \mapsto \beta_{\mathsf{x}_0,y}(y) := \pi_y(\beta(\mathsf{x}_0, y)),$$

where $\pi_y(x, y) := y$. Since β preserves parallel line segments, $\beta_{\mathsf{x}_0,y}$ maps the line with equation $x = \mathsf{x}_0$ onto itself (since it maps the y-axis to itself). Since β also preserves the rectangle given by the polyline

$$P = \langle (0,0), (0,1), (1,1), (\mathsf{x}_0, 1), (\mathsf{x}_0, 0), (1,0), (0,0), (0,1) \rangle$$

(for $\mathsf{x}_0 = 1$, we can omit $(\mathsf{x}_0, 1)$ and $(\mathsf{x}_0, 0)$ from the list) onto itself, we have again have $\beta_{\mathsf{x}_0,y}(0) = 0$ and $\beta_{\mathsf{x}_0,y}(1) = 1$. The function $\beta_{\mathsf{x}_0,y}$ also inherits from β the property of preserving midpoints and is strictly monotonic increasing on the line $x = \mathsf{x}_0$. So, again we can apply Lemma 3.15 to show that $\beta_{\mathsf{x}_0,y}$ is the identity.

Since $\beta(x, y) = (\beta_x(x), \beta_{x,y}(y))$, we obtain that β is the identity transformation. This finishes the proof of the claim and also of the lemma. □

4

Algorithms to test double-cross similarity

In this chapter, we present an algorithm for polyline- (and polygon-) similarity testing that is based on the double-cross formalism. To determine the degree of similarity between two polylines (not necessary of the same size), the algorithm first computes their "generalized polylines," that consist of almost equally long line segments and that approximate the length of the given polylines within an ε-error margin. Next, the algorithm determines the double-cross matrices of the generalized polylines and the difference between these matrices is used as a measure of dissimilarity between the given polylines. We prove termination of our algorithm and give its sequential time complexity.

We apply our method to query-by-sketch, indexing of polyline databases, and classification of terrain features and show experimental results for each of these applications.

4.1 Generalizations of polylines

When we describe our algorithm to test double-cross similarity of polylines, we want to be able to apply it to test similarity of polylines that are not necessarily of the same size. But, in order to compare their double-cross matrices, they need to be of the same size. Further as shown in Figure 3.1, two polylines can have the same semantics, but a different double-cross matrix. To overcome this problem, we introduce the notion of "generalized polylines."

For a given polyline P, a sequence of *generalized polylines*

$$P^2, P^4, P^8, ..., P^{2^n}, ...$$

are defined that tend (as n grows) to consist of equally long line segments. To define these generalized polylines, we first need to define the distance along P from the start vertex of P to some point belonging to $\mathsf{sem}(P)$. Hereto, we introduce a function

$$\lambda_P : [0, \ell(P)] \to \mathbf{R}^2$$

with $\ell(P)$ the length of P as defined below, that maps a length τ to the unique point of $\mathsf{sem}(P)$ that is at distance τ from the start vertex as we travel along the polyline from the start to the end vertex. Let $P = \langle \mathsf{p}_0, \mathsf{p}_1, ..., \mathsf{p}_N \rangle$ be a polyline and let ℓ_i denote the length of the vector $\overrightarrow{\mathsf{p}_{i-1}\mathsf{p}_i}$ for $1 \leq i \leq N$.

So, for any input τ of the function λ_P there exists a k, $0 < k \leq N$, and a $0 \leq \tau' < \ell_k$, such that $\tau = \sum_{i=1}^{k-1} \ell_i + \tau'$. Then $\lambda_P(\tau)$ is the unique point at distance τ' from p_{k-1} on the line segment connecting p_{k-1} and p_k.

In the following, we abbreviate the *length of* P by $\ell(P)$. So, $\ell(P) = \ell_1 + \ell_2 + \cdots + \ell_N$.

Definition 4.1. Let $P = \langle (x_0, y_0), (x_1, y_1), ..., (x_N, y_N) \rangle$ be a polyline. We define the *generalized polylines of* P, denoted $P^2, P^4, P^8, ..., P^{2^n}, ...$ as follows. For $n \geq 1$, the polyline $P^{2^n} = \langle (u_0, v_0), (u_1, v_1), ..., (u_{2^n}, v_{2^n}) \rangle$ with (u_i, v_i) the unique point on $\mathsf{sem}(P)$ at distance $\frac{i}{2^n} \cdot \ell(P)$ from the start vertex of P along P, that is,

$$(u_i, v_i) = \lambda_P(\frac{i}{2^n} \cdot \ell(P)),$$

for $i = 0, 1, ..., 2^n$. □

We remark that, in general, $\mathsf{sem}(P)$ and $\mathsf{sem}(P^{2^n})$ are *not* the same. They may even be different for all n. An example of this fact is given in Figure 4.1. Here, the three lines that make up the polyline $P = \langle (0,1), (0,0), (1,0), (1,1) \rangle$ are all equally long. Consequently, the vertices $(0,0)$ and $(1,0)$ will never belong to $\mathsf{sem}(P^{2^n})$, since $\frac{1}{3} \neq \frac{i}{2^n}$ and $\frac{2}{3} \neq \frac{i}{2^n}$ for any n and any $0 \leq i \leq 2^n$.

4.2 An algorithm to test double-cross similarity of polylines

4.2.1 The algorithm DC-similar$_\Delta$

We now describe our algorithm for determining the similarity of polylines and polygonal shapes (we give it in pseudo-code). This algorithm returns a degree of similarity between two given polylines P_1 and P_2. It uses an error rate ε

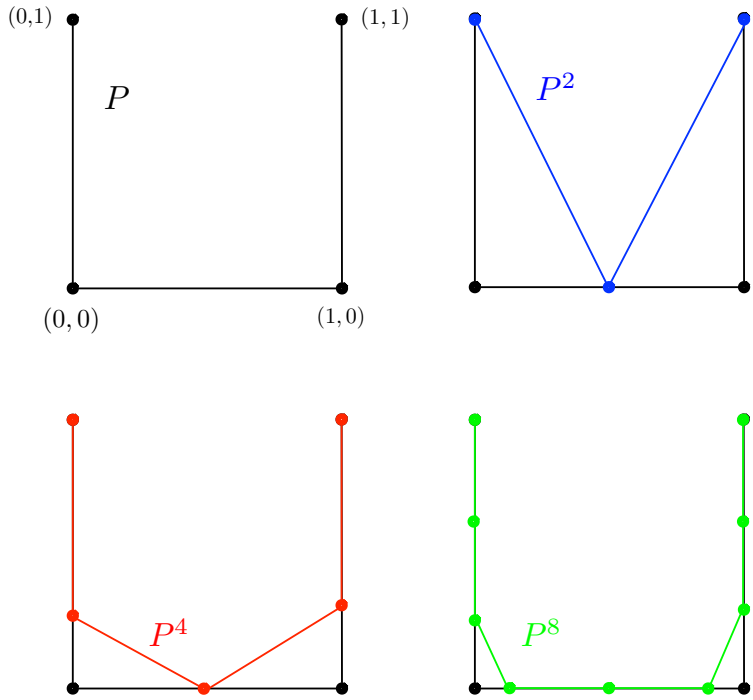


Figure 4.1: An example of a polyline $P = \langle (0,1), (0,0), (1,0), (1,1) \rangle$ (in black). Next, its first generalizations P^2, $P^4$and P^8 are shown in blue, in red and in green, respectively.

(with $0 \leq \varepsilon \leq 1$) and depends on a distance function Δ between double-cross matrices, which we consider a parameter of the algorithm.

Basically, the algorithm computes the generalized polylines of P_1 and P_2 until these approximate the length of P_1 and P_2 up to an error ε (a "small" percentage of the length). Then the double-cross matrices of the generalized polylines are computed and the distance between them is returned. By construction, and as can be seen in Figure 4.1, $P^{2^{n-1}} \in P^{2^n}$. So to speed up the calculation of P^{2^n} we recycle the points calculated in $P^{2^{n-1}}$ and only look at the original polyline P to calculate the extra points on $\mathsf{sem}(P)$.

Algorithm DC–SIMILAR$_\Delta$

input: trajectories P_1, P_2;
threshold $0 \leq \varepsilon \leq 1$.

set n:=1;
compute P_1^2 and P_2^2;

while $|\ell(P_1^{2^n}) - \ell(P_1)| \geq \varepsilon \cdot \ell(P_1)$ or $|\ell(P_2^{2^n}) - \ell(P_2)| \geq \varepsilon \cdot \ell(P_2)$
do
    n:=n+1;
    compute in parallel
      1. $P_1^{2^n}$ from $P_1^{2^{n-1}}$ and P_1; and
      2. $P_2^{2^n}$ from $P_2^{2^{n-1}}$ and P_2;
od

compute in parallel
     1. $M_1 := DCM(P_1^{2^n})$; and
     2. $M_2 := DCM(P_2^{2^n})$;
return $\Delta(M_1, M_2)$.

As mentioned, $\Delta(M_1, M_2)$ expresses a measure of difference or distance between the two double-cross matrices M_1 and M_2. There are many possibilities here, but in our experiments, we have used the distance function Δ_H (for an alternative function Δ_E, we refer to Section 4.4). To define the distance measure $\Delta_H(M_1, M_2)$ between double-cross matrices, we first construct, for both matrices M_1 and M_2, vectors $\gamma(M_1), \gamma(M_2) \in \mathbf{N}^{65}$ that counts for each of the 65 realizable 4-tuples of $-$, $+$ and 0, the number of times they occur in the matrices M_1 and M_2 (Remark 2.2 explains the number 65). Then, for matrices M_1 and M_2 of polylines of size N, we define

$$\Delta_H(M_1, M_2) := \frac{1}{N^2 - N} \sum_{i=1}^{65} |\gamma(M_1)[i] - \gamma(M_2)[i]| .$$

The function Δ_H counts the average difference between the 65 count values. We remark that the function Δ_H is not a distance function in the strict sense of the word. The reason is that there exists double-cross matrices M_1 and M_2 that are not equal but for which $\Delta_H(M_1, M_2) = 0$.

4.2.2 Basic properties of DC-SIMILAR$_\Delta$

We now give some basic properties of DC-similar$_\Delta$. To start, we show that the algorithm is guaranteed to terminate on any input. We remark that these properties are independent of the choice of Δ (as long as the evaluation of Δ terminates). First, we give a lemma.

Lemma 4.2. Let P be a polyline. For any $n \geq 1$, $\ell(P^{2^n}) \leq \ell(P)$ holds.

Proof. Or, $\mathsf{sem}(P^{2^n}) = \mathsf{sem}(P)$, in which case $\ell(P^{2^n}) = \ell(P)$, or, $\mathsf{sem}(P^{2^n}) \neq \mathsf{sem}(P)$, in which case P^{2^n} somewhere takes a shortcut (following a straight line) from P and $\ell(P^{2^n}) < \ell(P)$ by the triangle inequality. □

Proposition 4.3. The algorithm DC-SIMILAR$_\Delta$ terminates on any inputs P_1, P_2 and $0 < \varepsilon \leq 1$.

Proof. To prove this property, it suffices to show that for any polyline P and any $0 < \varepsilon \leq 1$, there exists an $n \geq 1$ such that $|\ell(P) - \ell(P^{2^n})| < \varepsilon \cdot \ell(P)$. We prove this by showing that $\lim_{n\to\infty} \ell(P^{2^n}) = \ell(P)$.

If we construct $P^{2^{n+1}}$ from P^{2^n} and P there are two possibilities: (1) if each vertex of $P^{2^{n+1}}$ is also an element of $\mathsf{sem}(P^{2^n})$, then $\mathsf{sem}(P^{2^n}) = \mathsf{sem}(P)$ and thus $\ell(P) = \ell(P^{2^n}) = \ell(P^{2^{n+1}})$ and the stop condition is satisfied; (2) there is at least one vertex p of $P^{2^{n+1}}$ that is not an element of $\mathsf{sem}(P^{2^n})$. Because of the triangle inequality and by construction we know that $\ell(P^{2^n}) < \ell(P^{2^{n+1}}) \leq \ell(P)$ (the last inequality is from Lemma 4.2). From these two cases we can deduce that $\lim_{n\to\infty} \ell(P^{2^n}) = \ell(P)$ and therefore there exists an $n \geq 1$ such that $|\ell(P) - \ell(P^{2^n})| < \varepsilon \cdot \ell(P)$. □

Proposition 4.4. When the algorithm DC-SIMILAR$_\Delta$, on input P_1, P_2 and $0 < \varepsilon \leq 1$, terminates for some n, then the standard deviation of the lengths of the line segments of $P_k^{2^n}$ tends to 0, more specifically, it is bounded by $\frac{\varepsilon}{2^n} \cdot \ell(P_k)$ $(k = 1, 2)$.

Proof. Let P be P_1 or P_2 and let $L_1, ..., L_{2^n}$ be the line segments of P^{2^n}. Let $\bar{\ell}$ denote the average length of the segments $L_1, ..., L_{2^n}$. The square of the standard deviation σ is then $\frac{1}{2^n}\sum_{i=1}^{2^n}(\ell(L_i) - \bar{\ell})^2$. By construction and the triangle inequality we know that $\ell(L_i) \leq \frac{\ell(P)}{2^n}$. We also know that

$$\ell(P^{2^n}) = \sum_{i=1}^{2^n} \ell(L_i) > \ell(P) \cdot (1 - \varepsilon).$$

Therefore,

$$\sigma^2 \leq \frac{1}{2^n}\sum_{i=1}^{2^n}(\frac{\ell(P)}{2^n} - \frac{\ell(P)(1-\varepsilon)}{2^n})^2 = (\frac{\ell(P)}{2^n} \cdot \varepsilon)^2$$

and thus $\sigma \leq \frac{\varepsilon}{2^n} \cdot \ell(P)$. □

4.2.3 Time complexity of DC-SIMILAR$_{\Delta_H}$

The following property gives an upper bound for the sequential time complexity of DC-SIMILAR$_{\Delta_H}$. We remark, that although large parts of DC-SIMILAR$_{\Delta_H}$ can be performed in parallel, this only gives a gain of a factor of 2.

Proposition 4.5. The algorithm DC-SIMILAR$_{\Delta_H}$, on input $P_1 = \langle(\mathsf{r}_0, \mathsf{s}_0), (\mathsf{r}_1, \mathsf{s}_1), ..., (\mathsf{r}_{N_1}, \mathsf{s}_{N_1})\rangle$, $P_2 = \langle(\mathsf{u}_0, \mathsf{v}_0), (\mathsf{u}_1, \mathsf{v}_1), ..., (\mathsf{u}_{N_2}, \mathsf{v}_{N_2})\rangle$ and $0 < \varepsilon \leq 1$, needs

$$O\left(\left(\frac{max(N_1, N_2)}{\varepsilon}\right)^2\right)$$

sequential time to return its output.

Proof. Let $P = \langle(\mathsf{x}_0, \mathsf{y}_0), (\mathsf{x}_1, \mathsf{y}_1), ..., (\mathsf{x}_N, \mathsf{y}_N)\rangle$ be P_1 or P_2. The difference between P and P^{2^n} is that line segments of P^{2^n} can short cut angles of P. Such a short-cut angle (or series of angles) of P has length $\frac{\ell(P)}{2^n}$ and there can at most be N cut angles. The difference in length between P and P^{2^n} is therefore bounded by $N \cdot \frac{\ell(P)}{2^n}$. We remark that this number will eventually, for increasing n, become smaller than $\varepsilon \cdot \ell(P)$, since $N \cdot \ell(P)$ is fixed. Therefore, the algorithm DC-SIMILAR$_{\Delta_H}$ terminates, on input P_1 and P_2, as soon as $N_1 \cdot \frac{\ell(P_1)}{2^n} < \varepsilon \cdot \ell(P_1)$ and $N_2 \cdot \frac{\ell(P_2)}{2^n} < \varepsilon \cdot \ell(P_2)$. This is true for $n = \lfloor log\left(\frac{M}{\varepsilon}\right)\rfloor + 1$ with $M = max(N_1, N_2)$. In this case, the while-loop of the algorithm has calculated maximally

$$2^{\lfloor log(\frac{M}{\varepsilon})\rfloor + 1} \leq 2\frac{M}{\varepsilon}$$

vertices on the generalized polylines and to compute the matrices M_1 and M_2 in the algorithm, we need in worst cast to calculate

$$\frac{(2\frac{M}{\varepsilon})^2 - 2\frac{M}{\varepsilon}}{2} \leq 2(\frac{M}{\varepsilon})^2$$

entries. The function Δ_H just scans each tuple of M_1 and M_2 in parallel once and needs maximum $O(\frac{M}{\varepsilon})$ sequential time. □

4.3 Experimental results

In this section, we discuss three experiments using Java-implementations of the algorithm DC-SIMILAR$_{\Delta_H}$. The run time results we mention are with respect to a system with an Intel ®Pentium ®M 1.86 GHz processor and 1GB RAM running Kubuntu as operating system.

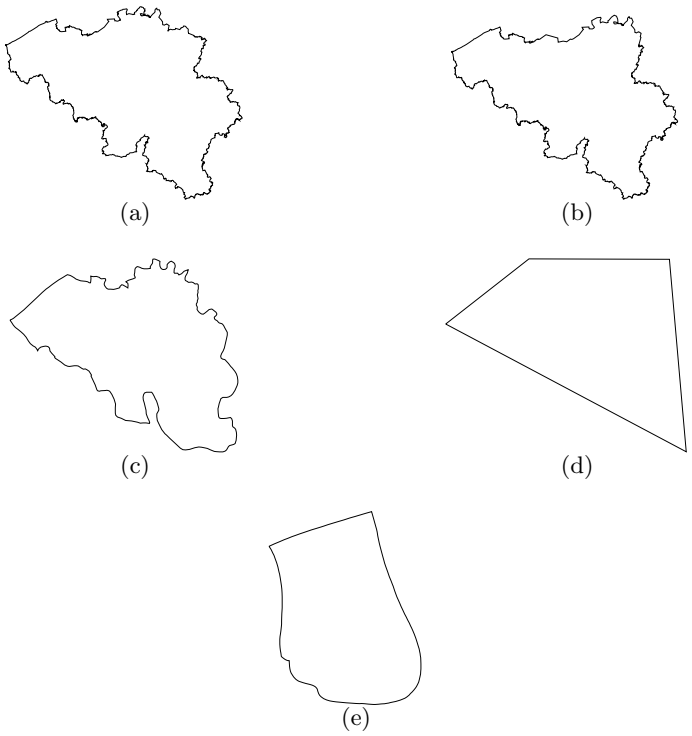


Figure 4.2: A map of Belgium and some sketches.

4.3.1 Experiment 1: Polygon similarity

For the first experiment, we used five figures representing Belgium (see Figure 4.2). Figure 4.2(a) is the correct representation of Belgium, consisting of 2047 line segments. Figure 4.2(b) is the same figure with the three bumps in the upper part moved to the right. The remaining three figures are sketches of Belgium made by respectively a geographer (Figure 4.2(c): real representation, Figure 4.2(d): abstract representation) and a non-specialist (Figure 4.2(e)).

We applied the algorithm DC-SIMILAR$_\Delta$ for $\Delta = \Delta_H$ to each two of these five figures with a threshold $\varepsilon = 0,05$. The results are given in Table 4.1. The measure Δ_H gives results that are very much in line with our intuition. For the above mentioned hardware, Δ_H takes ± 2.5 minutes to calculate the similarity of 2 shapes with 2^{15} line-segments (so $\pm 2^{29}$ entries). These experiments are in line with the theoretical complexity bound given by Proposition 4.5. Figure 4.2(b), with the three bumps moved to the right is 99% similar to the real map. The remaining three sketches of Belgium are ordered with decreasing similarity corresponding to our feeling.

Figure	4.2(a)	4.2(b)	4.2(c)	4.2(d)	4.2(e)
4.2(a)	100%	99%	87%	66%	60%
4.2(b)		100%	87%	67%	60%
4.2(c)			100%	80%	69%
4.2(c)				100%	84%
4.2(e)					100%

Table 4.1: Similarity between polygons of Figure 4.2 using Δ_H with threshold 5%.

We remark that Δ_H , in contradiction to Δ_E, because it is based on counting entries in the double-cross matrices, gives the same result, independent of the start vertex of polygonal input figures.

4.3.2 Experiment 2: Query by sketch

Because several experiments indicated that Δ_H gives good, start-vertex-independent results, we use the vector of 65 count values that is constructed by Δ_H as a method to *index* a database. This index is then used to support query-by-sketch. The 65-ary vector for a sketched figure is computed and, e.g., the four figures in the database that have a 65-ary vector closest to it are returned in order of descending best match.

Figure 4.3: The 43 cities and villages of Limburg.

More precisely, we have applied this method to perform a query-by-sketch on the villages and cities of Belgium. In this experiment we have calculated these index value for villages and cities, such that the generalized polygons of all cities approximate the length of the original polygon with an error smaller

than 2%. So in this case we do not use DC-SIMILAR$_\Delta$ to compare the figures side by side and thus the calculated double-cross matrices do not have the same size. Then for each of the villages and cities the 65-ary vector with count values of the 65 possible 4-tuples of $-$, $+$ and 0, is calculated as an index. Next, we queried for the presence of a sketched village. The sketched polygon is generalized up to the same level as the cities and villages in the database are; its index value is computed; and the four best fits (using the distance given by Δ_H) are returned.

For the sake of producing readable figures, we given an example of a query on a limited sample of cities in Belgium, namely the villages and cities in the province of Limburg, as depicted in Figure 4.3. This province consists of 43 villages and cities and it is not connected. Building the index for this sample costs 0.8 seconds on the above-mentioned configuration.

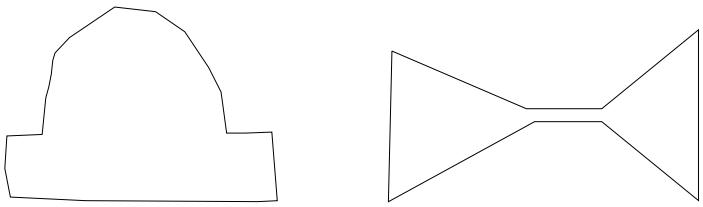


Figure 4.4: Bowler hat and bow tie sketch.

The first query is: *Give the 4 cities of Limburg looking the most as the bowler hat sketched on the left in Figure 4.4.*

The resulting cities are colored dark gray in Figure 4.5, with Lommel, the most similar city, shown in the north. We remark that Theorem 3.13 guarantees that similarity is tested invariant under translations, rotations and scalings, as can be seen from these results. Indeed, Lommel, looks like a the given bowler hat but 45° rotated.

Another query is: *Give the 4 cities of Limburg looking the most as the bow tie sketched on the right in Figure 4.4.*

The resulting cities are colored light gray in Figure 4.5. The best fit is Voeren, the village at the bottom right.

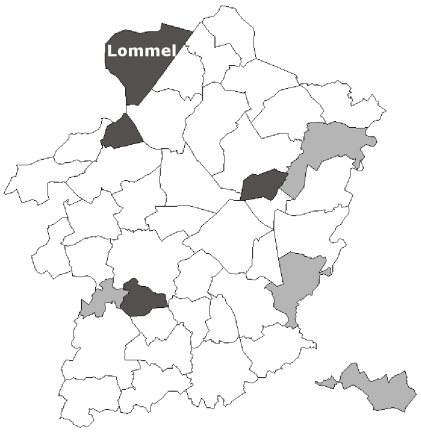


Figure 4.5: Result of bowler hat query (dark gray) and bow tie query (light gray).

4.3.3 Experiment 3: Classification of terrain features

Our last experiment deals with *terrain features*. The figures in this experiment are inspired by work of Kulik and Egenhofer [KE03]. We used the seven figures in Figure 4.6 as primitive terrain features to classify some silhouettes of terrains.

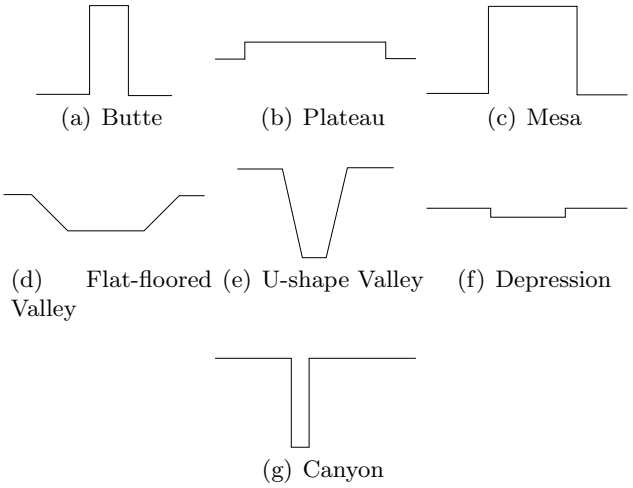
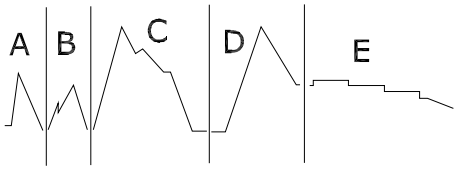


Figure 4.6: Basic shapes of terrain features.

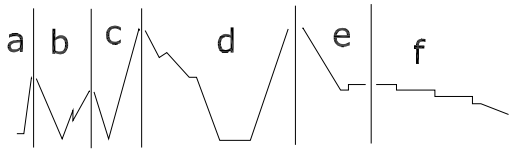
We use our Δ_H measure to classify the figures in Figure 4.7.
Our Δ_H algorithm classifies the three silhouettes of Figure 4.7, as (a) Mesa,

(a) (b) (c)

Figure 4.7: Some silhouettes of terrains.



(a)



(b)

Figure 4.8: Figure 4.7(c) divided by his maxima (left) and minima (right).

(b) U-Valley and (c) Mesa (or U Valley) respectively (see Tables 4.2, 4.3 and 4.4). For (a) and (b) this corresponds to our visual observations. Figure 4.7(c) is more complicated and needs more attention. We divided this figure according to its local maxima and minima (see Figure 4.8(a)). The resulting classifications, summarized in Tables 4.3 and 4.4, give a more precise description of these terrains.

	Fig. 4.7(a)	Fig. 4.7(b)	Fig. 4.7(c)
Butte	58%	52%	58%
Plateau	62%	62%	64%
Mesa	**71%**	64%	**70%**
Flat Valley	48%	56%	50%
U Valley	63%	**77%**	**68%**
Depression	47%	49%	50%
Canyon	42%	50%	43%

Table 4.2: Classification by Δ_H of Figure 4.7 using the primitives sketched in Figure 4.6.

	A	B	C	D	E
Butte	**67%**	62%	68%	**71%**	52%
Plateau	49%	58%	57%	40%	**62%**
Mesa	**68%**	**71%**	**78%**	65%	**62%**
Flat Valley	40%	40%	44%	46%	38%
U Valley	48%	54%	56%	40%	49%
Depression	38%	41%	42%	30%	54%
Canyon	39%	42%	40%	30%	47%

Table 4.3: Classification by Δ_H of Figure 4.8(a) using the primitives sketched in Figure 4.6.

	a	b	c	d	e	f
Butte	52%	49%	49%	57%	52%	58%
Plateau	50%	40%	48%	41%	50%	**66%**
Mesa	53%	49%	53%	53%	53%	**68%**
Flat Valley	**71%**	54%	45%	62%	**71%**	47%
U Valley	54%	**63%**	**61%**	**69%**	68%	56%
Depression	43%	43%	41%	34%	46%	56%
Canyon	31%	37%	52%	54%	43%	50%

Table 4.4: Classification by Δ_H of Figure 4.8(b) using the primitives sketched in Figure 4.6.

4.4 An alternative Δ, which is more suited for polylines

We have given a number of basic properties and described the time complexity of the algorithm DC-SIMILAR$_\Delta$ for testing polyline similarity. This algorithm depends on a function Δ that measures the difference between double-cross matrices. We have experimented with a number of Δ's and Δ_H, used throughout the paper gives the best experimental results. As stated, one reason for this might be that it is independent from the choice of start vertex of a polygon. For polylines, that are not polygons, it might be preferable to work with a Δ that compares corresponding line segments in the two polylines more directly. We here give an example of a Δ, namely Δ_E, that is more appropriate for polylines. First, we define a distance δ between elements of $\{-, 0, +\}$:

- $\delta(-,-) := 0$,
- $\delta(+,+) := 0$,
- $\delta(0,0) := 0$,
- $\delta(-,0) := 1$,
- $\delta(+,0) := 1$,
- $\delta(-,+) := 2$ and
- $\delta(x,y) := \delta(y,x)$.

Next, we define a distance $\bar{\delta}$ between entries of the double-cross matrices:

$$\bar{\delta}((C_1C_2C_3C_4),(C_1'C_2'C_3'C_4')) := \sum_{i=1}^{4} \delta(C_i, C_i').$$

Finally, if P_1 and P_2 are polylines with N edges, then we define

$$\Delta_E(DCM(P_1), DCM(P_2)) := \frac{1}{4(N-1)^2} \sum_{1 \le i < j \le N} \bar{\delta}(\mathsf{DCM}(P_1)[i,j], \mathsf{DCM}(P_2)[i,j]).$$

The function Δ_E measures the differences between the two matrices entry per entry. In a double-cross matrix, there are $\frac{N^2-N}{2}$ meaningful entries of which $N-1$ (the ones just above the diagonal) have maximal distance 4 and the other ones have maximal distance 8. In total, the maximal distance can reach $4(N-1)^2$, which explains the factor at the start of the above formula.

The proposed algorithm DC-SIMILAR$_\Delta$ might also be improved in other ways. For instance, termination conditions based on the Hausdorff distance between a polyline and its generalized polylines might be considered.

5

The double-cross matrix of polylines on a grid

In this chapter, we study properties of double-cross matrices of polylines that are situated on a grid. Such polylines may arise from trajectories on Manhattan-like road networks.

5.1 Polylines on a grid

To start with, we define the class of grids that we will work with. Let $\mathbf{Z}$ denote the set of the integers.

Definition 5.1. The *complete infinite grid* is defined to be the set $(\mathbf{Z} \times \mathbf{R}) \cup (\mathbf{R} \times \mathbf{Z})$. We call the elements of $\mathbf{Z} \times \mathbf{Z}$ the *crossings* of this grid. A *grid* is a subset of the complete infinite grid of the form $(A \times \mathbf{R}) \cup (\mathbf{R} \times B)$, with A, B finite subsets of $\mathbf{Z}$. We call the elements of $A \times B$ the *crossings* of this grid.

We say that a polyline P *is on a grid* G, if the semantics of P is contained in G, that is, if $\mathsf{sem}(P) \subset G$. □

In Figure 5.1, a grid G and polyline P on G are shown. In this example, not all vertices of P are located on crossings of G. For the case where all the vertices are crossings, we have the following definition.

Definition 5.2. Let G be a grid and let $P = \langle(\mathsf{x}_0, \mathsf{y}_0), \ldots, (\mathsf{x}_N, \mathsf{y}_N)\rangle$ be a polyline on G. We say that P *is snapped to* G if all vertices of P are crossings of G. Furthermore, if all crossings of G that belong to $\mathsf{sem}(P)$ are vertices of P, we say that P *is completely snapped to* G. □

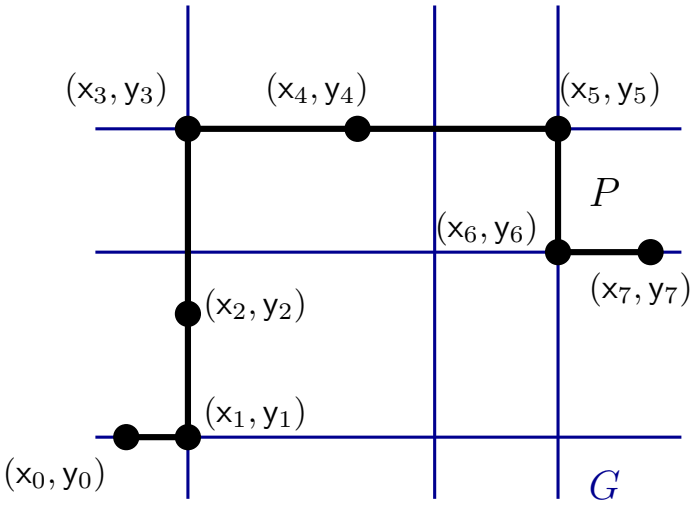


Figure 5.1: An example of a polyline (in black) on a grid (blue).

We remark that (completely) snapped polylines have vertices with integer-valued coordinates. But, since we assume rational coordinates, we can always find a suitable point-scaling that maps an arbitrary polyline on a grid to a snapped polyline on a sufficiently large grid, such that the resulting polyline has vertices with integer-valued coordinates. We refer to Property 5.12 for details.

Later on, for an arbitrary polyline P on a grid, we define a canonical polyline $\mathsf{can}(P)$, which has the same double-cross matrix as P, that is snapped to the complete infinite grid and that is minimal in some sense. To prove the existence of $\mathsf{can}(P)$, we will first need some definitions.

Definition 5.3. Given a polyline $P = \langle(x_0, y_0), \ldots, (x_N, y_N)\rangle$ on a grid, the lines V_i, given by the equation $x = x_i$, and H_i, given by the equation $y = y_i$, $0 \leq i \leq N$, are called the *vertical*, respectively *horizontal carriers of* P. □

Figure 5.2 shows a polyline $P = \langle(x_0, y_0), \ldots, (x_7, y_7)\rangle$ on a grid (not shown) and its vertical and horizontal carriers. We remark that some of these carriers may coincide, as illustrated by $H_0 = H_5 = H_6$ and $V_0 = V_1$ in the example.

We now define two lists of lists that capture the order of the carriers of a polyline on a grid.

Definition 5.4. Let $P = \langle(x_0, y_0), \ldots, (x_N, y_N)\rangle$ be a polyline on a grid. The *V-order of* P, denoted as $V(P)$, is a list $(A_1, \ldots, A_K)$ of lists $A_i = (a_{i1}, \ldots, a_{ik_i})$ such that each a_{ij} is an element of $\{0, 1, \ldots, N\}$ and appears exactly once in one of the lists of $V(P)$. Within each A_i, the elements appear

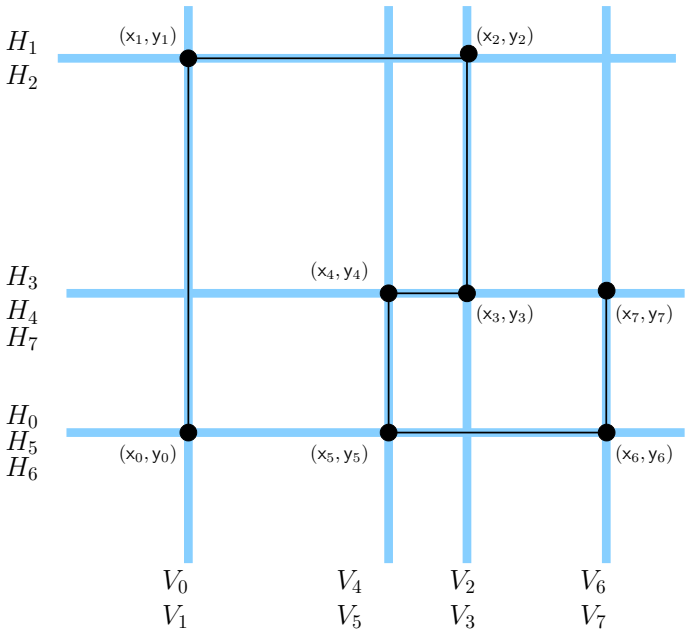


Figure 5.2: A polyline and its horizontal and vertical carriers (in blue).

in increasing order, and for $a, a' \in A_i$, we have $\mathsf{x}_a = \mathsf{x}_{a'}$. For $a \in A_i$ and $a' \in A_j$, with $i < j$, we have $\mathsf{x}_a < \mathsf{x}_{a'}$.

The *H-order of* P, denoted as $H(P)$, is a list $(B_1, \ldots, B_L)$ of lists $B_i = (b_{i1}, \ldots, b_{il_i})$ such that each b_{ij} is an element of $\{0, 1, \ldots, N\}$ and appears exactly once in one of the lists of $H(P)$. Within each B_i, the elements appear in increasing order, and for $b, b' \in B_i$, we have $\mathsf{y}_b = \mathsf{y}_{b'}$. For $b \in B_i$ and $b' \in B_j$, with $i < j$, we have $\mathsf{y}_b < \mathsf{y}_{b'}$. □

For the polyline P in Figure 5.2, $V(P) = ((0,1),(4,5),(2,3),(6,7))$ and $H(P) = ((0,5,6),(3,4,7),(1,2))$. We remark that $V(P)$ and $H(P)$ are invariant under translations and scalings but not under rotations of the plane $\mathbf{R}^2$. For instance, if ρ is a rotation over 180°, then $V(\rho(P)) = ((6,7),(2,3),(4,5),(0,1))$ and $H(\rho(P)) = ((1,2),(3,4,7),(0,5,6))$.

The following property follows immediately from the definition.

Proposition 5.5. Let $P = \langle(\mathsf{x}_0, \mathsf{y}_0), \ldots, (\mathsf{x}_N, \mathsf{y}_N)\rangle$ be a polyline on a grid. Given only $V(P)$, we can decide whether $\mathsf{x}_i < \mathsf{x}_j$, $\mathsf{x}_i = \mathsf{x}_j$ or $\mathsf{x}_j < \mathsf{x}_i$ for any $1 \le i < j \le N$. Given only $H(P)$, we can decide whether $\mathsf{y}_i < \mathsf{y}_j$, $\mathsf{y}_i = \mathsf{y}_j$ or $\mathsf{y}_j < \mathsf{y}_i$ for any $1 \le i < j \le N$. □

We now define equivalence of polylines on a grid in terms of their V- and H-order.

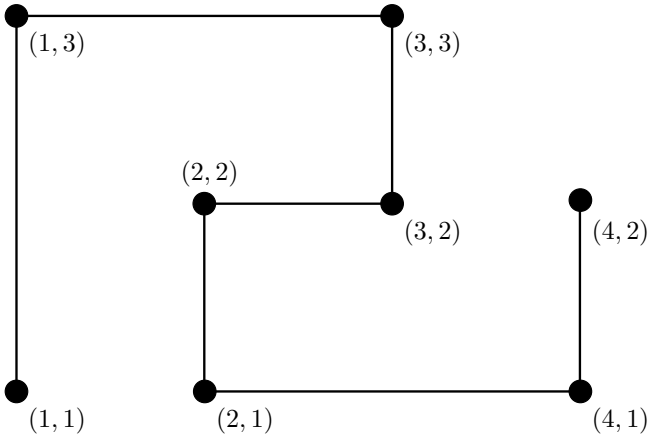


Figure 5.3: The canonical polyline of the polyline of Figure 5.2.

Definition 5.6. Let P and Q be two polylines of the same size on a grid G. We say that P *and* Q *are VH-equivalent*, denoted $P \equiv_{VH} Q$, if $V(P) = V(Q)$ and $H(P) = H(Q)$. □

Given a polyline on a grid, we now associate a *canonical polyline* on the complete infinite grid to it.

Definition 5.7. Given a polyline $P = \langle(\mathsf{x}_0, \mathsf{y}_0), \ldots, (\mathsf{x}_N, \mathsf{y}_N)\rangle$ on a grid, with $V(P) = (A_1, \ldots, A_K)$ and $H(P) = (B_1, \ldots, B_L)$, we define the *canonical polyline of* P, denoted $\mathsf{can}(P)$, to be the polyline $\langle(\mathsf{x}'_0, \mathsf{y}'_0), \ldots, (\mathsf{x}'_N, \mathsf{y}'_N)\rangle$, that is snapped to the complete infinite grid, such that for all $i = 0, ..., N$, we have if i belongs to A_j, then $\mathsf{x}'_i = j$ and if i belongs to B_j, then $\mathsf{y}'_i = j$. □

For the polyline P of Figure 5.2, $\mathsf{can}(P) = \langle(1,1),(1,3),(3,3),(3,2),(2,2),(2,1),(4,1),(4,2)\rangle$ is shown in Figure 5.3. We remark that if P is a polyline on a grid, its canonical polyline, *can*(P) is a snapped polyline on the complete infinite grid. In fact, $\mathsf{can}(P)$ can be viewed as a polyline on the "small" grid $(\{1, \ldots, K\} \times \mathbf{R}) \cup (\mathbf{R} \times \{1, \ldots, L\})$.

The following property follows straight from the definition.

Proposition 5.8. Let P and Q be of the same size on a grid G. We have $P \equiv_{VH} Q$ if and only if $\mathsf{can}(P) = \mathsf{can}(Q)$. □

5.1.1 Properties of double-cross matrices of polylines on a grid

In this section, we discuss some properties of the double-cross matrix of polylines on a grid. First, we remark that only 33 4-tuples $(C_1\ C_2\ C_3\ C_4) \in \{-, 0, +\}^4$ can appear in the double-cross matrix of a polyline on a grid, namely, those of the form

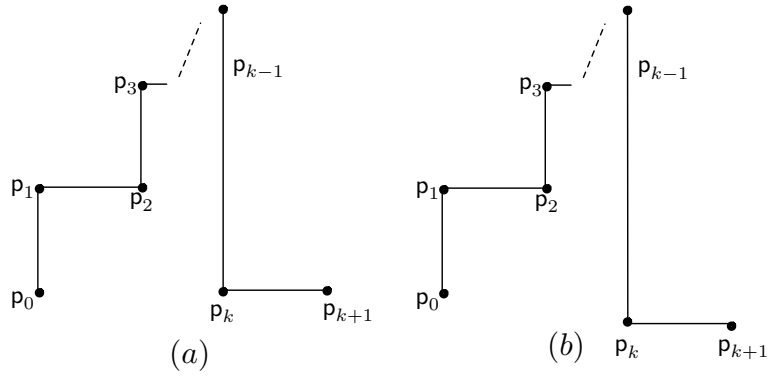


Figure 5.4: Example polylines for the proof of Proposition 5.9.

- $(C_1\ C_2\ C_3\ C_4)$,
- $(C_1\ C_2\ 0\ 0)$,
- $(C_1\ 0\ 0\ C_4)$,
- $(0\ 0\ C_3\ C_4)$,
- $(0\ C_2\ C_3\ 0)$ and
- $(0\ 0\ 0\ 0)$,

with $C_1, C_2, C_3, C_4 \in \{+, -\}$.

The following property shows that a k-partial double-cross matrix[1], in general, is not enough to know the complete double-cross matrix. This property holds on grids and hence in general.

Proposition 5.9. Let $k \geq 1$. For polylines (on a grid or not), the $(k+1)$-partial double-cross matrix $\mathsf{DCM}^{k+1}(P)$ cannot be derived from the k-partial double-cross matrix $\mathsf{DCM}^{k}(P)$.

Proof. The polylines in (a) and (b) of Figure 5.4 have the same DCM^k but they do not have the same DCM^{k+1}. A polyline like in Figure 5.4 can be generated for each $k \geq 2$. For the case $k = 1$, just consider $P_1 = \langle(0,0),(0,2),(0,1),(1,1)\rangle$ and $P_2 = \langle(0,0),(0,2),(0,-1),(1,-1)\rangle$. We have $\mathsf{DCM}^1(P_1) = \mathsf{DCM}^1(P_2)$ but $\mathsf{DCM}^2(P_1) \neq \mathsf{DCM}^2(P_2)$. Since polylines on a grid are a subset of polylines on the real plane, this also proves the general case. □

[1]Introduced in Definition 2.5

This property implies that, given a double-cross matrix M, we need to look at all cells (above the diagonal) of the matrix M to be able to reconstruct a polyline P such that $\mathsf{DCM}(P) = M$.

Proposition 5.10. If P is a polyline on a grid, then $\mathsf{DCM}(P) = \mathsf{DCM}(\mathsf{can}(P))$.

Proof. Let $P = \langle(\mathsf{x}_0, \mathsf{y}_0), \ldots, (\mathsf{x}_N, \mathsf{y}_N)\rangle$ be a polyline on a grid G and let $\mathsf{can}(P) = \langle(\mathsf{x}'_0, \mathsf{y}'_0), \ldots, (\mathsf{x}'_N, \mathsf{y}'_N)\rangle$ be its canonical polyline.

Suppose that $\mathsf{DCM}(P)[i,j] = \mathsf{DC}(\overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}}, \overrightarrow{\mathsf{p}_j\mathsf{p}_{j+1}}) = (C_1\ C_2\ C_3\ C_4)$ and $\mathsf{DCM}(\mathsf{can}(P))[i,j] = (D_1\ D_2\ D_3\ D_4)$, with $0 \leq i < j < N$. We have to show that $C_k = D_k$, for $k = 1, 2, 3, 4$. If we look at the algebraic expression for C_1, C_2, C_3 and C_4 of Theorem 3.2, we can see they are all of the form $\pm\text{sign}((t_{11} - t_{12}).(t_{21} - t_{22}) \pm (t_{31} - t_{32}).(t_{41} - t_{42}))$ and that t_{k1} and t_{k2} are x-coordinates or y-coordinates of vertices, with $k \in \{1, 2, 3, 4\}$. Let us focus on C_1 and D_1. We have $C_1 = -\text{sign}((\mathsf{x}_j - \mathsf{x}_i).(\mathsf{x}_{i+1} - \mathsf{x}_i) - (\mathsf{y}_j - \mathsf{y}_i).(\mathsf{y}_{i+1} - \mathsf{y}_i))$ and $D_1 = -\text{sign}((\mathsf{x}'_j - \mathsf{x}'_i).(\mathsf{x}'_{i+1} - \mathsf{x}'_i) - (\mathsf{y}'_j - \mathsf{y}'_i).(\mathsf{y}'_{i+1} - \mathsf{y}'_i))$. Since P is a polyline on a grid, we have that $\mathsf{x}_i = \mathsf{x}_{i+1}$ and $\mathsf{y}_i \neq \mathsf{y}_{i+1}$ or $\mathsf{x}_i \neq \mathsf{x}_{i+1}$ and $\mathsf{y}_i = \mathsf{y}_{i+1}$. Assume $\mathsf{x}_i = \mathsf{x}_{i+1}$ and $\mathsf{y}_i \neq \mathsf{y}_{i+1}$ (the other case is similar). By definition of $V(P)$ and $\mathsf{can}(P)$ we know when $\mathsf{x}_i = \mathsf{x}_{i+1}$ then $\mathsf{x}'_i = \mathsf{x}'_{i+1}$, thus we can rewrite the expression for C_1 and D_1 as follows: $C_1 = -\text{sign}((\mathsf{y}_j - \mathsf{y}_i).(\mathsf{y}_{i+1} - \mathsf{y}_i))$ and $D_1 = -\text{sign}((\mathsf{y}'_j - \mathsf{y}'_i).(\mathsf{y}'_{i+1} - \mathsf{y}'_i))$. By definition of $\mathsf{can}(P)$, we can see that $\mathsf{y}_i < \mathsf{y}_j$ if and only if $\mathsf{y}'_i < \mathsf{y}'_j$. Also, $\mathsf{y}_i < \mathsf{y}_{i+1}$ if and only if $\mathsf{y}'_i < \mathsf{y}'_{i+1}$. So, we have $C_1 = D_1$. We can show in a similar way that $C_2 = D_2$, $C_3 = D_3$ and $C_4 = D_4$. □

In the next theorem, we use the following notation. If $P = \langle(\mathsf{x}_0, \mathsf{y}_0), (\mathsf{x}_1, \mathsf{y}_1), \ldots, (\mathsf{x}_N, \mathsf{y}_N)\rangle$ is a polyline, then we denote by P_i the polyline $\langle(\mathsf{x}_0, \mathsf{y}_0), \ldots, (\mathsf{x}_i, \mathsf{y}_i)\rangle$, for $0 \leq i \leq N$. So, in particular $P = P_N$.

We need the following mapping to align the start vectors of two polylines. Let $P = \langle\mathsf{p}_0, \mathsf{p}_1, \ldots, \mathsf{p}_N\rangle$ and $Q = \langle\mathsf{q}_0, \mathsf{p}_1, \ldots, \mathsf{q}_M\rangle$ be two polylines and let α_{PQ} be the following unique orientation-preserving affinity of $\mathbf{R}^2$ that maps $\overrightarrow{\mathsf{p}_0\mathsf{p}_1}$ to $\overrightarrow{\mathsf{q}_0\mathsf{q}_1}$. The transformation α_{PQ} can be decomposed into a translation τ_{PQ} that maps the start vertex of P to the start vertex of Q; a counter-clockwise rotation ρ_{PQ} that aligns $\tau_{PQ}(\overrightarrow{\mathsf{p}_0\mathsf{p}_1})$ with $\overrightarrow{\mathsf{q}_0\mathsf{q}_1}$; and finally a point-scaling (or homotecy) σ_{PQ} that maps $\rho_{PQ}(\tau_{PQ}(\overrightarrow{\mathsf{p}_0\mathsf{p}_1}))$ onto $\overrightarrow{\mathsf{q}_0\mathsf{q}_1}$.

Theorem 5.11. *Let P and Q be two polylines of size N on a grid. Then P and Q are double-cross similar if and only if*

$$\alpha_{PQ}(P_{N-1}) \equiv_{VH} Q_{N-1}$$

and the last vector of $\alpha_{PQ}(P)$ and Q have the same direction with respect to their one but last vector, that is,

$$\mathsf{DC}(\overrightarrow{\mathsf{p}_{N-2}\mathsf{p}_{N-1}}, \overrightarrow{\mathsf{p}_{N-1}\mathsf{p}_N}) = \mathsf{DC}(\overrightarrow{\mathsf{q}_{N-2}\mathsf{q}_{N-1}}, \overrightarrow{\mathsf{q}_{N-1}\mathsf{q}_N}).$$

Proof. From Property 5.10, we know that $\mathsf{DCM}(P) = \mathsf{DCM}(Q)$ if and only if $\mathsf{DCM}(\mathsf{can}(P)) = \mathsf{DCM}(\mathsf{can}(Q))$ and from Property 5.8 that $\alpha_{PQ}(P_{N-1}) \equiv_{VH} Q_{N-1}$ if and only if $\mathsf{can}(\alpha_{PQ}(P_{N-1})) = \mathsf{can}(Q_{N-1})$. Therefore, it is enough to prove that $\mathsf{DCM}(\mathsf{can}(P)) = \mathsf{DCM}(\mathsf{can}(Q))$ if and only if $\mathsf{can}(\alpha_{PQ}(P_{N-1})) = \mathsf{can}(Q_{N-1})$ and the last vector of $\mathsf{can}(\alpha_{PQ}(P_N))$ and $\mathsf{can}(Q_N)$ have the same direction with respect to their one but last vector. Since $\mathsf{can}(\alpha_{PQ}(P_{N-1})) = \mathsf{can}(Q_{N-1}) = \alpha_{\mathsf{can}(P)\mathsf{can}(Q)}(\mathsf{can}(P_{N-1}))$ (where $\alpha_{\mathsf{can}(P)\mathsf{can}(Q)}$ is basically a rotation over 0°, 90°, 180°, or 270° with center $(1,1)$), it suffices to prove the theorem for canonical polylines with VH-equivalence replaced by equality.

So, from now, we simplify the notation and assume that P and Q are *canonical*. We will therefore write P and Q, meaning, $\mathsf{can}(P)$ and $\mathsf{can}(Q)$.

First, we prove the if-direction. We distinguish between $N = 2$ and $N > 2$. For $N = 2$, we have to show that if $\alpha_{PQ}(P_1) = Q_1$ and the last vector of $\alpha_{PQ}(P_2)$ and Q_2 have the same direction with respect to their one but last vector (in this case the first vector), then $\mathsf{DCM}(P_2) = \mathsf{DCM}(Q_2)$. This is trivial since the direction of the second segment with respect to the first is uniquely determines the right upper element in the double-cross matrix.

Let us now assume $N > 2$. And we assume $\mathsf{DC}(\overrightarrow{\mathsf{p}_{N-2}\mathsf{p}_{N-1}}, \overrightarrow{\mathsf{p}_{N-1}\mathsf{p}_N}) = \mathsf{DC}(\overrightarrow{\mathsf{q}_{N-2}\mathsf{q}_{N-1}}, \overrightarrow{\mathsf{q}_{N-1}\mathsf{q}_N})$ together with $Q_{N-1} = \alpha_{PQ}(P_{N-1})$. The former implies that $\mathsf{DCM}(P)[N-1, N] = \mathsf{DCM}(Q)[N-1, N]$. The latter implies that $\mathsf{DCM}(P)[i, j] = \mathsf{DCM}(Q)[i, j]$ for $1 \leq i < N$ and $1 \leq j < N$, by induction hypothesis. But since the end vertex of Q_{N-1} is equal to the end vertex of $\alpha_{PQ}(P_{N-1})$, we also have $\mathsf{DCM}(P)[i, N] = \mathsf{DCM}(Q)[i, N]$ for $i = 1, ..., N-2$.

For the only-if direction, let us assume that $\mathsf{DCM}(P) = \mathsf{DCM}(Q)$. From this assumption $\mathsf{DCM}(P)[N-1, N] = \mathsf{DCM}(Q)[N-1, N]$ follows and therefore the last vector of $\alpha_{PQ}(P)$ and Q have the same direction with respect to their one but last vector. Let $P = \langle(\mathsf{x}_0, \mathsf{y}_0), ..., (\mathsf{x}_N, \mathsf{y}_N)\rangle$. For each pair of vertices $(\mathsf{x}_i, \mathsf{y}_i)$ and $(\mathsf{x}_j, \mathsf{y}_j)$, with $1 \leq i < j < N$, it can be determined, from $\mathsf{DCM}(P)$, whether $\mathsf{x}_i < \mathsf{x}_j$, $\mathsf{x}_i = \mathsf{x}_j$ or $\mathsf{x}_i > \mathsf{x}_j$ and also $\mathsf{y}_i < \mathsf{y}_j$, $\mathsf{y}_i = \mathsf{y}_j$ or $\mathsf{y}_i > \mathsf{y}_j$. Indeed, suppose $\mathsf{DCM}(P)[i, j] = (C_1\ C_2\ C_3\ C_4)$. By looking at Theorem 3.2, one can see that if $C_3 = 0$ that $(\mathsf{x}_j, \mathsf{y}_j)$ is located on $\overrightarrow{\mathsf{p}_{i-1}\mathsf{p}_i}$. If $C_3 = -$, $(\mathsf{x}_j, \mathsf{y}_j)$ is located right of $\overrightarrow{\mathsf{p}_{i-1}\mathsf{p}_i}$, otherwise $(\mathsf{x}_j, \mathsf{y}_j)$ is located left of $\overrightarrow{\mathsf{p}_{i-1}\mathsf{p}_i}$. In other words, the V- and H-order of P_{N-1} can be derived from $\mathsf{DCM}(P)$. Since $\mathsf{DCM}(P) = \mathsf{DCM}(Q)$, the canonical polylines of P_{N-1} and Q_{N-1} are therefore equal. This concludes the proof. □

Proposition 5.12. If P is a polyline on a grid G and $G' = (A \times \mathbf{R}) \cup (\mathbf{R} \times B)$ is a grid with A and B finite subsets of $\mathbf{R}$ and with $|A| \geq |V(P)|$ and $|B| \geq |H(P)|$, then there exists a snapped polyline Q on G' such that $\mathsf{DCM}(P) = \mathsf{DCM}(Q)$.

Proof. We construct Q on G' as we constructed the canonical polyline of P on the complete infinite grid (see Definition 5.7). Then by construction $P \equiv_{VH} Q$ and the last vector from P and Q have the same direction with respect to the one but last vector. By Theorem 5.11, it follows that $\mathsf{DCM}(P) = \mathsf{DCM}(Q)$. □

Using the previous properties, we can see that two double-cross similar polylines differ in horizontal and vertical compressions and dilatations from the canonical polyline (apart from their last vector, whose length is arbitrary).

Corollary 5.13. Let P and Q be two polylines of size N on a grid (not necessarily the same grid). If P and Q are double-cross similar, there exists a transformation, preserving the V- and H-order, that maps P_{N-1} onto Q_{N-1} and that preserves the direction of the last vector of P with respect to the one but last vector of P. □

5.1.2 Constructing example polylines on a grid from a given a double-cross matrix

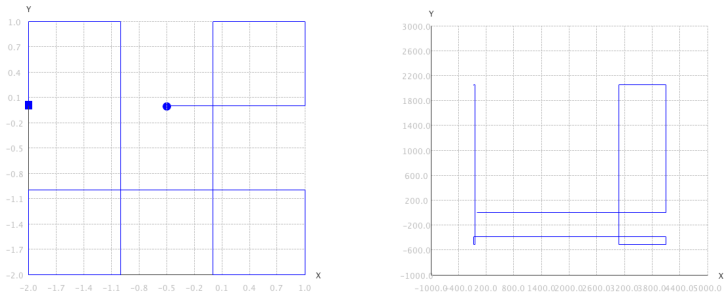
We now give an algorithm Reconstruct_DCM that, given a double-cross matrix M of size $N \times N$ of a polyline on a grid, generates a snapped polyline P on an exponentially large grid with $M = \mathsf{DCM}(P)$ in time $O(N^2)$. As will be explained later on, the algorithm Reconstruct_DCM can be modified to discover if the input is a matrix that is (not) realizable by a polyline on a grid.

From the output of Reconstruct_DCM, it is straightforward to produce the V- and H-order of the generated polyline. These orders can be used to generate many more example polylines that satisfy the given double-cross matrix, in particular the canonical polyline.

Listing 5.1: Algorithm Reconstruct_DCM

```
input: a N × N matrix M;

N² := size(M);

L1.start := (0,0);
lastX := 2^N;
lastY := 0;
L1.end := (lastX, lastY);
P := L1;

for(i:=2; i<N−1; i++) {
  Li.start := Li−1.end;
```

```
 compute direction d of p_i p_{i+1}
    using direction of p_{i-1} p_i and M[i-1,i];
 switch(d){
  case Horizontal Right:
    newY := lastY;
    based on V(P), find the interval I
       such that ∀x ∈ I and
      ℓ = (lastX, lastY), (x, newY), and
      ∀j ∈ [1, i-1] : DC(p_j p_{j+1}, ℓ) = M[j,i]
    newX := min(x ∈ I)+ 2^{N-i};
    break;
  case ...
  /* do similar constructions
   * for the other three directions */
 }//end switch
 L_i.end := (newX,newY);
 lastX := newX;
 lastY := newY;
P.addLast(L_i);
}//endfor

L_N.start := L_{N-1}.end;
 compute direction d of p_{N-1} p_N
   using direction of p_{N-2} p_{N-1} and M[N-2,N-1];
 switch(d){
  case Horizontal Right:
    newY := lastY;
    newX :=lastX + 1;
    break;
  case ...
  /* do similar constructions
   * for the other three directions */
    }//end switch
 L_N.tail := (newX,newY);
 P.addLast(L_N);

Return P;
```



(a) (b)

Figure 5.5: (a) A polyline on a grid. The circle is the start vertex and the square is the end vertex. (b) The reconstructed snapped polyline from the double-cross matrix in Table 5.1.

Theorem 5.14. *The algorithm* Reconstruct_DCM*, on input a double-cross matrix M of size $N \times N$ of a polyline on a grid, correctly generates, in $O(N^2)$ time, a snapped polyline P of size N with* $\mathsf{DCM}(P) = M$*. This algorithm can also be used to check , with the same time complexity, whether a matrix of size $N \times N$ is realizable with a polyline on a grid.*

Proof. The length of the interval I in the algorithm is in the i-th step maximal 2^{N-i+1}. Therefore, the new vertex can always be located in I. The correctness of the rest of the algorithm can be proven using Theorem 5.11 and Property 5.12.

For what concerns checking, we observe that the algorithm cannot find an interval I if and only if the matrix is not realizable. □

The algorithm Reconstruct_DCM produces a snapped polyline on a grid of exponential size. However, we remark that the algorithm only outputs the vertices of the polyline, which can be described in linear size space (using the bit representation of integers).

In Figure 5.5(a), we can see a polyline on a grid. Its double-cross matrix is given in Table 5.1. In Figure 5.5(b), we see the snapped polyline generated using the algorithm Reconstruct_DCM with as input the double-cross matrix in Table 5.1.

Table 5.1: The double-cross matrix of the snapped polyline in Figure 5.5(a).

−00+	− − ++	− − +−	− + −+	− − −+	− − −−	+ + −+	+ − −+	+ − −−	+ + ++	+ − ++
	−00+	− − ++	+ − ++	+ − 00	+00−	+ + ++	+ − ++	+ − +−	− + ++	− − ++
		−00+	− − ++	0 − +0	00 + −	− + ++	− − ++	− − +−	− + 00	−00+
			−00+	− − ++	− − +−	− + −+	− − −+	− − −−	0 + −0	00 − +
				−00+	− − ++	+ − ++	+ − 00	+00−	+ + ++	+ − ++
					−00+	− − ++	0 − +0	00 + −	− + ++	− − ++
						−00+	− − ++	− − +−	− + −+	− − −+
							−00+	− − ++	+ − ++	+ − 00
								−00+	− − ++	0 − +0
									−00+	− − ++
										−00+

5.1.3 Properties of double-cross matrices of completely snapped polylines

Now, we prove a property of the double-cross matrix of a completely snapped polyline, that does not hold for arbitrary polylines on a grid.

Proposition 5.15. Given the 1-partial double-cross matrix $M = \mathsf{DCM}^1(P)$ of a completely snapped polyline P, we can reconstruct $\mathsf{DCM}(P)$.

Proof. Suppose P has N vertices. Then $M = \mathsf{DCM}^1(P)$ has $N-2$ elements. Let $[-N, N]$ denote the set of integers $\{-N, -N+1, ..., 0, 1, ..., N\}$. Take $G = ([-N, N] \times \mathbf{R}) \cup (\mathbf{R} \times [-N, N])$ as a grid. We now show how we can reconstruct a polyline Q that has M as its double-cross matrix.

We start with setting $Q = \langle (0,0), (0,1) \rangle$. Then Q is a completely snapped polyline and G is big enough to reconstruct any polyline with N line segments that has as start vertex $(0,0)$. If we want to add a vertex $(\mathsf{x}_2, \mathsf{y}_2) \in G$ to Q such that Q is still a completely snapped polyline and $M[1,2] = DC(\overrightarrow{\ell_1(Q)}, \overrightarrow{\ell_2(Q)})$, we only have one possibility for the position of $(\mathsf{x}_2, \mathsf{y}_2)$. Based on the fact that Q has to a be completely snapped polyline and $(\mathsf{x}_2, \mathsf{y}_2) \in G$ there are only four possible coordinates for $(\mathsf{x}_2, \mathsf{y}_2)$. The condition $M[0,1]$ will only be true on $Q$for one of these four points and so we have a unique solution. For the same reasons we can see that if we want to extend Q to N line segments such that $M[i, i+1]$holds on Q for all $0 \leq i < N-1$, all $(\mathsf{x}_j, \mathsf{y}_j)$, for $2 \leq j \leq N$ are uniquely defined by the choice of the initial segment from $(0,0)$ to $(0,1)$; by G; and by M. Once we have obtained $Q = \langle (\mathsf{x}_0, \mathsf{y}_0), \ldots, (\mathsf{x}_N, \mathsf{y}_N) \rangle$ this way, we can calculate $\mathsf{DCM}(Q)$. From Property 5.12, we know that the choice of G has no influence on $\mathsf{DCM}(Q)$. Because a double-cross matrix is invariant under a composition of a rotation, translation and point-scaling, taking another initial line segment for Q would also not change $\mathsf{DCM}(Q)$ therefore for every Q constructed as described above $\mathsf{DCM}(Q) = \mathsf{DCM}(P)$. □

The last proof actually describes an algorithm that given a (partial) double-cross matrix M of a completely snapped polyline, generates a completely snapped polyline P with $\mathsf{DCM}(P) = M$. The following corollary tells something about the time complexity.

Corollary 5.16. Given a double-cross matrix M of size $N \times N$ of a completely snapped polyline, a completely snapped polyline P of size N with $M = \mathsf{DCM}(P)$ can be generated in $O(N)$ time. Checking whether a double-cross matrix of size N by N, can be of a completely snapped polyline takes $O(N^2)$ time. □

6

On the realizability of double-cross matrices

By Properties 3.2 and 3.3, we know that we only need to know the elements of a double-cross matrix above the diagonal. On the diagonal we find (0 0 0 0) and the elements below the diagonal can be reconstructed from those above the diagonal. For polylines of length N, this means that we can have $3^{2N(N-1)}$ possible matrices of 4-tuples $(C_1\ C_2\ C_3\ C_4) \in \{-, 0, +\}^4$. In this chapter, we address the question, which of those matrices are actually the double-cross matrix of a polyline in $\mathbf{R}^2$.

More specifically, we address the following two problems.

Problem 6.1 (Realizability). Given is an $N \times N$ matrix $M$of 4-tuples $(C_1\ C_2\ C_3\ C_4) \in \{-, 0, +\}^4$.

(a) Decide whether M is the double-cross matrix of some polyline in $\mathbf{R}^2$ of size N; and

(b) If the answer to question (a) is *yes*, then produce an example (or many examples) of a polyline P with $\mathsf{DCM}(P) = M$. □

A related problem, though of less interest to us, is the following.

Problem 6.2. Given is a length N. Which and how many of the $3^{2N(N-1)}$ possible matrices of 4-tuples $(C_1\ C_2\ C_3\ C_4) \in \{-, 0, +\}^4$ are the double-cross matrix of a polyline of size N in $\mathbf{R}^2$? □

First, we describe a theoretical, but impractical solution to this problem. If we restrict our attention to several special cases, like 90°- and 45°-polylines, we present more efficient answers. Hereto, we introduce the polar coordinate representation of polylines.

6.1 A theoretical solution

Let us first address Problem 6.1 (a). If a given $N \times N$ matrix M of 4-tuples $(C_1\ C_2\ C_3\ C_4) \in \{0,+,-\}^4$ is the double-cross matrix of a polyline $P = \langle (x_0,y_0),(x_1,y_1),,\ldots,(x_N,y_N)\rangle$, then the coordinates $x_0, y_0, x_1, y_1, ..., x_N, y_N$ should satisfy the conditions expressed by the entries of the matrix M. By Theorem 3.2, these conditions can be translated into quadratic polynomial equalities and inequalities. These are of the form

$$\left\{\begin{array}{lll} (x_j - x_i)\cdot(x_{i+1} - x_i) + (y_j - y_i)\cdot(y_{i+1} - y_i) & \alpha_{ij} & 0 \\ (x_j - x_i)\cdot(x_{j+1} - x_j) + (y_j - y_i)\cdot(y_{j+1} - y_j) & \beta_{ij} & 0 \\ (x_j - x_i)\cdot(y_{i+1} - y_i) - (y_j - y_i)\cdot(x_{i+1} - x_i) & \gamma_{ij} & 0 \\ (x_j - x_i)\cdot(y_{j+1} - y_j) - (y_j - y_i)\cdot(x_{j+1} - x_j) & \delta_{ij} & 0 \end{array}\right. \qquad (\dagger)$$

for $0 \le i < j \le N$ and $\alpha_{ij}, \beta_{ij}, \gamma_{ij}, \delta_{ij} \in \{=, <, >\}$.

The values $\alpha_{ij}, \beta_{ij}, \gamma_{ij}, \delta_{ij}$ are determined by the entries of the matrix M as indicated in Theorem 3.2 (the minus signs before the equations for C_1 and C_3 are assumed to be incorporated in the α_{ij} and γ_{ij}).

We make the following observations about this system:

- there are $4\frac{N(N-1)}{2} = 2N(N-1)$ (in)equalities in the $2(N+1)$ variables $x_0, y_0, x_1, y_1, ..., x_N, y_N$;
- each polynomial uses 6 variables from $x_0, y_0, x_1, y_1, ..., x_N, y_N$ and has at most 8 monomial terms;
- each of the polynomials is homogeneous of degree 2;
- all the coefficients of the polynomials are 0, 1 or -1.

These $4\frac{N(N-1)}{2}$ equalities and inequalities (†) describe a semi-algebraic subsets of $\mathbf{R}^{2(N+1)}$ (see [JBR98]) that is given by a system (†) of (in)equalities in the $2(N+1)$ variables $x_0, y_0, x_1, y_1, ..., x_N, y_N$.

To answer Problem 6.1 (a), we have to verify whether the system of (in)equalities (†) has a solution in $\mathbf{R}^{2(N+1)}$. In other words, we want to decide whether this semi-algebraic subset of $\mathbf{R}^{2(N+1)}$ is empty or not.

So, Problem 6.1 (a) adds up to deciding whether the first-order sentence

$$\exists x_0 \exists y_0 \exists x_1 \exists y_1 \cdots \exists x_N \exists y_N \bigwedge_{0 \le i < j \le N} \begin{cases} (x_j - x_i) \cdot (x_{i+1} - x_i) + (y_j - y_i) \cdot (y_{i+1} - y_i) & \alpha_{ij} & 0 \\ (x_j - x_i) \cdot (x_{j+1} - x_j) + (y_j - y_i) \cdot (y_{j+1} - y_j) & \beta_{ij} & 0 \\ (x_j - x_i) \cdot (y_{i+1} - y_i) - (y_j - y_i) \cdot (x_{i+1} - x_i) & \gamma_{ij} & 0 \\ (x_j - x_i) \cdot (y_{j+1} - y_j) - (y_j - y_i) \cdot (x_{j+1} - x_j) & \delta_{ij} & 0 \end{cases}$$

is true in the structure of the real ordered field. Obviously, the input for this decision problem consists of the $4\frac{N(N-1)}{2}$ conditions $\alpha_{ij}, \beta_{ij}, \gamma_{ij}, \delta_{ij}$, $0 \le i < j \le N$, that can be read from the input matrix M.

The first-order theory of the real ordered field is decidable ([Tar51]) and various implementations of decision procedures, that are based on Cylindrical Algebraic Decomposition ([Col75]) or other techniques, for this theory exist. We refer to QEPCAD ([Hon]), REDLOG ([DS97]) and MATHEMATICA ([Wol]) as a few examples. This type of software could be used, in theory, to answer Problem 6.1 (a) in practice. If there is a solution, these implementations also provide, as a byproduct of the above decision problem, sample points, thus, also, effectively answering question Problem 6.1 (b). But it is also known that the above mentioned implementations are slow and fail in practice to produce answers as soon as the number of variables increases.

Experiments with MATHEMATICA show that only polylines of up to 5 segments can be reconstructed this way (depending on the instance). This obviously does not provide a practical solution to questions Problem 6.1 (a) and (b).

We include a small example, with $N = 4$, of a trajectory found by MATHEMATICA. If we give the matrix

$$\begin{pmatrix} (0\,0\,0\,0) & (-+0+) & (--++) & (--++) \\ & (0\,0\,0\,0) & (--0+) & (--++) \\ & & (0\,0\,0\,0) & (-+0+) \\ & & & (0\,0\,0\,0) \end{pmatrix}$$

as input, and ask for a polyline starting with vertices $\mathsf{p}_0 = (0,0), \mathsf{p}_1 = (5,0)$, continuing with vertices $p_2 = (x_2, y_2), p_3 = (x_3, y_3)$ and $p_4 = (x_4, y_4)$ and ignoring the first and third entry of two successive line segments (which is always $-$ and 0), then the MATHEMATICA input looks like:

```
FindInstance[{(5 - 0)*(x2 - 5) + (0 - 0)*(y2-0)>0,
(5 - 0)*(y2 - 0) - (0 - 0)*(x2 - 5) > 0,
(x2 - 0)*(5 - 0) + (y2 - 0)*(0 - 0) > 0,
(x2 - 0)*(x3 - x2) + (y2 - 0)*(y3 - y2) < 0,
```

```
(x2 - 0)*(0 - 0) - (y2 - 0)*(5 - 0) < 0,
(x2 - 0)*(y3 - y2) - (y2 - 0)*(x3 - x2) > 0,
(x3 - 0)*(5 - 0) + (y3 - 0)*(0 - 0) > 0,
(x3 - 0)*(x4 - x3) + (y3 - 0)*(y4 - y3) < 0,
(x3 - 0)*(0 - 0) - (y3 - 0)*(5 - 0) < 0,
(x3 - 0)*(y4 - y3) - (y3 - 0)*(x4 - x3) > 0,
(x2 - 5)*(x3 - x2) + (y2 - 0)*(y3 - y2) < 0,
(x2 - 5)*(y3 - y2) - (y2 - 0)*(x3 - x2) > 0,
(x3 - 5)*(x2 - 5) + (y3 - 0)*(y2 - 0) > 0,
(x3 - 5)*(x4 - x3) + (y3 - 0)*(y4 - y3) < 0,
(x3 - 5)*(y2 - 0) - (y3 - 0)*(x2 - 5) < 0,
(x3 - 5)*(y4 - y3) - (y3 - 0)*(x4 - x3) > 0,
(x3 - x2)*(x4 - x3) + (y3 - y2)*(y4 - y3) > 0,
(x3 - x2)*(y4 - y3) - (y3 - y2)*(x4 - x3) > 0},
{x2, y2, x3, y3, x4, y4}]
```

and MATHEMATICA answers after about 10 seconds with

$x_2 = \frac{343597733205}{68719476736}$,

$y_2 = \frac{1067}{10485760}$,

$x_3 = \frac{1}{131072}$,

$y_3 = \frac{1}{4}$,

$x_4 = -(\frac{1637}{131072})$ and

$y_4 = 0$.

This is due to the intrinsic high time complexity of quantifier elimination in the ordered field of the reals [HKp04]. The theory of computational algebraic geometry gives an upper complexity bound. In particular, Theorem 13.13 in [BPR06] gives an upper time complexity bound on determining realizable sign conditions of a collection of polynomials. When applied to our decision problem, we get the following result.

Property 6.1. There exists an algorithm to compute the set of *all* realizable sign conditions of the system (†) with complexity

$$(2N(N-1))^{2N+3} \cdot 2^{O(N)}.$$

The complexity of deciding the (non-)emptiness of (†) is the same, as well as that of generating a sample point in case of non-emptiness. □

The complexity of deciding the satisfiability of the system is the same, as well as that of generating a sample point in case of non-emptiness. The use of alternative data structures to codify the polynomials can improve the time complexity, but not below exponential time ([GH01]). For a more recent discussion on lower bounds of the complexity, we refer to ([HKP13]).

The general problem of deciding an existential sentence in the first-order theory of the reals is known to be NP-hard (and to be in PSPACE) [Can88]. An exponential lower bound is not known [HKp04, HKP13].

However, we have the following, negative result from [SN01] (see also [RN07]):

Property 6.2. Problem 6.1 (a) is NP-hard. □

Whether or not this problem is in NP is less obvious. It is known that if there is a solution to the above system of polynomial (in)equalities, there is also an solution with algebraic coordinates ([BPR06]). We could, for instance, try to guess the coordinates of the vertices of a polyline and then verify whether it satisfies the above system. Guessing algebraic coordinates could be implemented by guessing a polynomial and a root of this polynomial. However, an apriori polynomial bound on the complexity of sample points (to be guessed) is not obvious ([BPR06]). Above, we have observed that each polynomial uses at most 6 variables from $x_0, y_0, x_1, y_1, ..., x_N, y_N$ and has at most 8 monomial terms. This implies our problem is part of the field of "fewnomials" ([Kho91]), where problems are notoriously difficult. And our problem and the production of sample points, is not covered by the known solutions there.

On the positive side, we have remarked in Chapter 3 that it is clear that translations, rotations and scalings of a polyline do not change its double-cross matrix. Double-cross matrices are, in fact, invariant under *similarities* of $\mathbf{R}^2$. Thus, we can conclude, that if Problem 6.1 (a) has a positive answer, we can always find a polyline, to witness this fact, that starts of with the vertices $(\mathsf{x}_0, \mathsf{y}_0) = (0, 0)$ and $(\mathsf{x}_1, \mathsf{y}_1) = (1, 0)$ and in which the other vertices have coordinates that are algebraic numbers.

We can conclude that, whereas Problem 6.1 (a) and (b) are solved in theory, a practical solution remains open.

6.2 Generating double-cross similar polylines with equal length line segments for a given polyline

In this section, we look at the problem of generating double-cross similar polylines with equal length line segments for a given polyline. The reconstruction task is considerably simplified by looking for polylines with segments of *equal length*.

For a given polyline P of size N, we compute the double-cross matrix M of size N by N; we set the desired lengths of the line segments L; and the granularity (the number of candidates to be generated) s. The reconstruction algorithm works as follows. In the first step, the algorithm creates a line of length L with as start vertex $(0, 0)$ and end vertex $(L, 0)$.

In the second step, the algorithm creates a set of candidate second line segments. By looking at $M[1, 2]$, we know what the minimum and maximum angle is between the first and the second line segment. There are two possibilities: the minimum angle is equal to maximum angle; or it is not. In the first case, there is a unique solution to position the second line segment. In the other case, we create s candidate solutions, equally distributed between the minimum and maximum angle.

Table 6.1: The double-cross matrix of the polyline in Figure 6.1.

$-+0+$	$-+++$	$-++-$	$--++$	$--++$	$--+-$	$--++$	$+++$
	$-+0+$	$-++-$	$--++$	$--++$	$--++$	$-+++$	$+++-$
		$--0-$	$---+$	$-+-+$	$--++$	$-+++$	$-++-$
			$--0+$	$--++$	$+-++$	$++--$	$++--$
				$-+0+$	$--++$	$-++-$	$-++-$
					$--0+$	$-++-$	$-++-$
						$--0-$	$----$
							$-+0-$

Given the double-cross matrix in Table 6.1, which is the double-cross matrix of the polyline in Figure 6.1, the output of the algorithm with $L = 3$ and $s = 10$, in the second step looks like the polyline in Figure 6.2.

In the i-th step of the algorithm ($i \leq N$), we take one by one the polylines created in the $(i-1)$-th step, and create new polylines by adding s possible last line segments (as we did in the second step). The only difference now is that when we have a candidate polyline P_{new}, we first check whether $\mathsf{DCM}(P_{new})$ corresponds to the given matrix M. If this is not the case, this candidate is pruned. Only when this is the case, P_{new} is further considered as possible realisation of M.

Given the double-cross matrix in Table 6.1, the output of the algorithm after the fourth step of the algorithm looks like the polyline in Figure 6.3. Figure 6.4 gives polylines that all have the double-cross matrix of Table 6.1. We remark that Figure 6.4 contains not all polylines that were considered as solution in Figure 6.3, especially the polylines of which the end vertex was below the x-axis. In this experiment, 632 polylines satisfying the double-cross matrix of Table 6.1 were created in approximately 3 seconds on a Apple

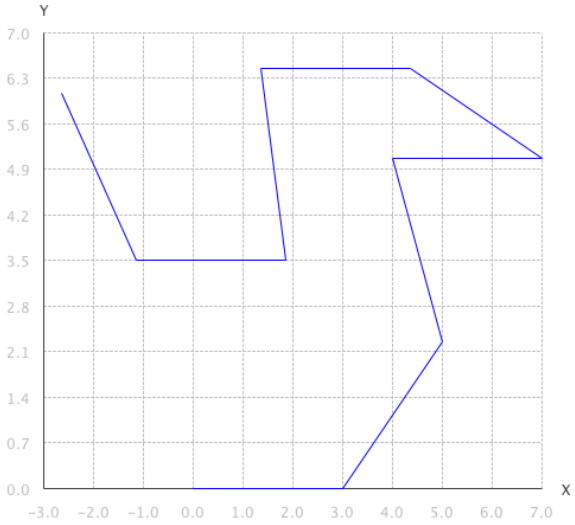


Figure 6.1: The original polyline.

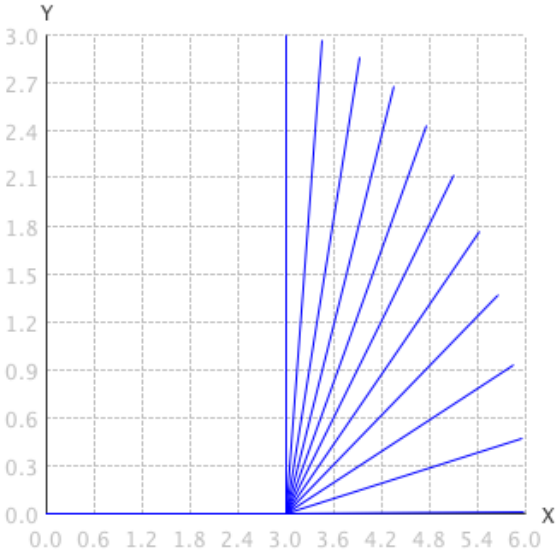


Figure 6.2: The output after the second step of the algorithm using the double-cross matrix in Table 6.1.

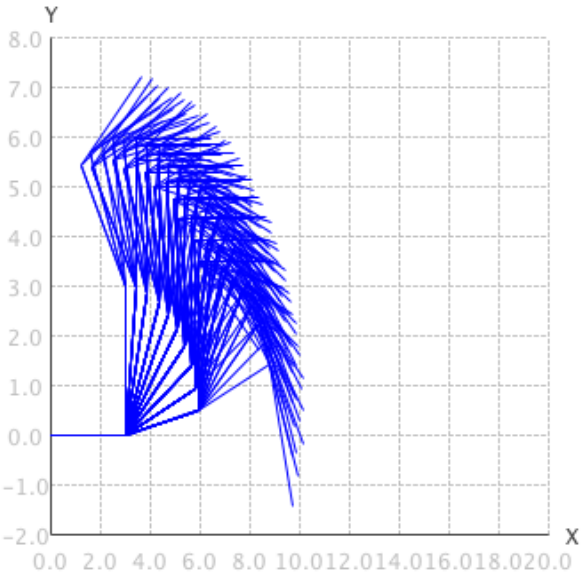


Figure 6.3: The considered solutions in step 4 of the algorithm.

Macbook with 2.16 GHz Intel Core 2 Duo processor and 1 GB RAM.

6.3 A realizability test for 90°-polylines

In this section, we give an efficient solution for a special case of Problem 6.1, that we first state.

Problem 6.3 (90°**-realizability).** Given is an $N \times N$ matrix $M$of 4-tuples $(C_1\ C_2\ C_3\ C_4) \in \{-, 0, +\}^4$.

(a) Decide whether M is the double-cross matrix of some 90°-polyline in $\mathbf{R}^2$ of size N; and

(b) If the answer to question (a) is *yes*, then produce an example (or many examples) of a 90°-polyline P with $\mathsf{DCM}(P) = M$. □

Obviously, this problem is related to the topic of polylines on a grid, that we discussed in Chapter 5, but we will treat it here in a purely algebraic way, rather than in a geometric way.

For the problem of realizability, we may assume, without loss of any generality, that the polyline that realizes a matrix M, if it exists, starts with the unit interval on the x-axis, that is, $\mathsf{p}_0 = (\mathsf{x}_0, \mathsf{y}_0) = (0, 0)$ and $\mathsf{p}_1 = (\mathsf{x}_1, \mathsf{y}_1) = (1, 0)$. Indeed, from Lemma 3.14, we know that similarities preserve the double-cross matrix of a polyline.

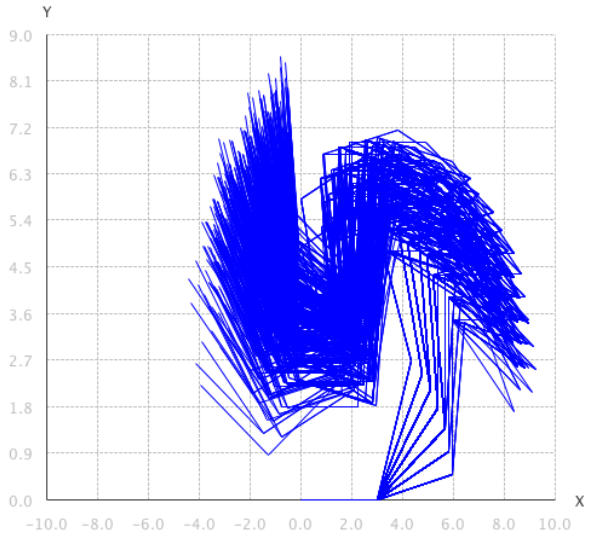


Figure 6.4: A set of complete solutions.

The following property gives a first necessary condition for the input to our decision problem, the matrix M.

Property 6.3. Let $P = \langle \mathsf{p}_0, \mathsf{p}_1, \mathsf{p}_2, ..., \mathsf{p}_N \rangle$ be a polyline. A necessary and sufficient condition for P to be a 90°-polyline is: for all i, $0 \leq i < N - 1$, $\mathsf{DC}(\overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}}, \overrightarrow{\mathsf{p}_{i+1}\mathsf{p}_{i+2}}) =$

- $(-\ -\ 0\ 0)$ (reverse);
- $(-\ 0\ 0\ -)$ (right turn);
- $(-\ +\ 0\ 0)$ (straight); or
- $(-\ 0\ 0\ +)$ (left turn). □

Since we have taken $\mathsf{p}_0 = (\mathsf{x}_0, \mathsf{y}_0) = (0, 0)$ and $\mathsf{p}_1 = (\mathsf{x}_1, \mathsf{y}_1) = (1, 0)$, all line segments of the polyline, realizing M, should be either horizontal or vertical. In fact, we have for each i, $0 \leq i < N$ that

$$\mathsf{x}_i = \mathsf{x}_{i+1} \wedge (\mathsf{y}_i < \mathsf{y}_{i+1} \vee \mathsf{y}_i > \mathsf{y}_{i+1})$$

or

$$\mathsf{y}_i = \mathsf{y}_{i+1} \wedge (\mathsf{x}_i < \mathsf{x}_{i+1} \vee \mathsf{x}_i > \mathsf{x}_{i+1}).$$

Since we take $\mathsf{p}_0 = (\mathsf{x}_0, \mathsf{y}_0) = (0, 0)$ and $\mathsf{p}_1 = (\mathsf{x}_1, \mathsf{y}_1) = (1, 0)$, all line segments of the polyline, realising M, should be or horizontal or vertical in

$\mathbf{R}^2$ with respect to the standard coordinate axes. In fact, we have for each i, $0 \leq i < N$ that $\mathsf{x}_i = \mathsf{x}_{i+1} \wedge (\mathsf{y}_i < \mathsf{y}_{i+1} \vee \mathsf{y}_i > \mathsf{y}_{i+1})$ or $\mathsf{y}_i = \mathsf{y}_{i+1} \wedge (\mathsf{x}_i < \mathsf{x}_{i+1} \vee \mathsf{x}_i > \mathsf{x}_{i+1})$. Here, for $0 \leq i < N$, we are in exactly one of the following four situations (always with $\ell_{i+1} > 0$):

$$\begin{cases} x_{i+1} = x_i + \ell_{i+1} \\ y_{i+1} = y_i; \end{cases} \quad \begin{cases} x_{i+1} = x_i - \ell_{i+1} \\ y_{i+1} = y_i; \end{cases} \quad \begin{cases} x_{i+1} = x_i \\ y_{i+1} = y_i + \ell_{i+1}; \end{cases} \quad \begin{cases} x_{i+1} = x_i \\ y_{i+1} = y_i - \ell_{i+1}. \end{cases}$$

Before we give an efficient solution to Problem 6.3, we prove a lemma that explains in which quadrant, determined by a horizontal or vertical line segment of a polyline, a vertex of a polyline is located. For clarity, we state and prove the lemma for horizontal and vertical segments of a polyline that coincide with the unit interval on the x- and y-axis (or their negatives), but the lemma can be easily extended and applied to any horizontal and vertical polyline segments after applying a scaling and translation of $\mathbf{R}^2$. This lemma is a variant of Property 3.9.

Lemma 6.4. Let $P = \langle \mathsf{p}_0, \mathsf{p}_1, , \ldots, \mathsf{p}_N \rangle$ be a polyline and assume that $\mathsf{p}_i = (0,0)$ and $\mathsf{p}_{i+1} = (\pm 1, 0)$ or $\mathsf{p}_i = (0,0)$ and $\mathsf{p}_{i+1} = (0, \pm 1)$. Let $\mathsf{p}_j = (\mathsf{x}_j, \mathsf{y}_j)$, for $0 \leq i \leq N$ and $i + 1 < j$. From the first and third component of $\mathsf{DC}(\overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}}, \overrightarrow{\mathsf{p}_j\mathsf{p}_{j+1}})$, we can determine $\mathrm{sign}(\mathsf{x}_j)$ and $\mathrm{sign}(\mathsf{y}_j)$.

Proof. First, let $\mathsf{p}_i = (0,0)$ and $\mathsf{p}_{i+1} = (\pm 1, 0)$. From Theorem 3.2 it is clear that $C_1 = -\mathrm{sign}((\mathsf{x}_j - 0) \cdot \pm 1 + \mathsf{y}_j \cdot 0) = -\mathrm{sign}(\pm \mathsf{x}_j)$ and that $C_3 = -\mathrm{sign}(\mathsf{x}_j \cdot 0 - \mathsf{y}_j \cdot \pm 1) = \mathrm{sign}(\pm \mathsf{y}_j)$.

Secondly, let $\mathsf{p}_i = (0,0)$ and $\mathsf{p}_{i+1} = (0, \pm 1)$. Similarly, Theorem 3.2 implies $C_1 = -\mathrm{sign}((\mathsf{x}_j - 0) \cdot 0 + \mathsf{y}_j \cdot \pm 1) = -\mathrm{sign}(\pm \mathsf{y}_j)$ and that $C_3 = -\mathrm{sign}(\mathsf{x}_j \cdot \pm 1 - \mathsf{y}_j \cdot 0) = -\mathrm{sign}(\pm \mathsf{x}_j)$. □ □

Now, we give an efficient solution to Problem 6.3.

Theorem 6.5. *It can be decided in time $O(N^2)$ whether a $N \times N$ matrix M of 4-tuples $(C_1\ C_2\ C_3\ C_4) \in \{-, 0, +\}^4$ is the double-cross matrix of some $90°$-polyline in $\mathbf{R}^2$ of size N. If M is $90°$-realizable, also witnesses to this can be produced in time $O(N^2)$.*

Proof. We now describe a decision procedure for Problem 6.3 : in a first step, we determine the relationship $(<, =, >)$ between coordinates of consecutive vertices. In a second step, we do it for all remaining vertices

Let M be a $N \times N$ matrix of 4-tuples $(C_1\ C_2\ C_3\ C_4) \in \{-, 0, +\}^4$. As an apriori step, we check whether M doesn't have (0 0 0 0) entries on its diagonal or doesn't have the "symmetry" properties, discussed in Section 3.2.1. If M fails this symmetry-test, we can already answer *no*, else we proceed.

Step 1. First, we inspect all entries $M[i, i+1]$, $0 \leq i < N$ of M. They should all be of the form

- $(-\ -\ 0\ 0)$ (reverse turn);
- $(-\ 0\ 0\ -)$ (right turn);
- $(-\ +\ 0\ 0)$ (straight); or
- $(-\ 0\ 0\ +)$ (left turn).

If this is not the case, we can already answer *no*. In the other case, we deduce the arrangement[1] of x_i and x_{i+1} of the coordinates of candidate vertices of a polyline. We do the same for y_i and y_{i+1} and determine whether $\mathsf{y}_i < \mathsf{y}_{i+1}$, $\mathsf{y}_i = \mathsf{y}_{i+1}$ or $\mathsf{y}_i > \mathsf{y}_{i+1}$. Then we proceed to Step 2.

Step 2. Now, we inspect all entries $M[i,j]$, $1 \leq i+1 < j < N$ of M. Now, per entry, two cases have to be considered.
Case 1 ($\mathsf{x}_i = \mathsf{x}_{i+1}$): Taking into account, $\mathsf{y}_i < \mathsf{y}_{i+1}$ or $\mathsf{y}_i > \mathsf{y}_{i+1}$, and $\mathsf{y}_j < \mathsf{y}_{j+1}$ or $\mathsf{y}_j > \mathsf{y}_{j+1}$, we can use the vertical version of Lemma 6.4, to determine the quadrant in which $(\mathsf{x}_j, \mathsf{y}_j)$ is located compared to the vertical line segment that connects p_i and p_{i+1}. This gives us the arrangement of x_j and x_i on the one hand and y_j and y_i on the other hand.
Case 2 ($\mathsf{y}_i = \mathsf{y}_{i+1}$): This case is analogous to the previous one. We can get the arrangement of x_j and x_i on the one hand and y_j and y_i on the other hand, but now using the horizontal version of version of Lemma 6.4.

Now, we have now complete information on how the x-coordinate values $\mathsf{x}_0, \mathsf{x}_1, ..., \mathsf{x}_N$ are arranged (or ordered) and how the y-coordinate values $\mathsf{y}_0, \mathsf{y}_1, ..., \mathsf{y}_N$ are arranged (since, form the definition of the double-cross matrix, the length of the last line segment is irrelevant, one of x_N and y_N may be undetermined, but we know the direction of the final line segment). We can store this arrangement information in two matrices (similarly to the double-cross matrix). The first matrix can be used to verify whether an ordering of $\mathsf{x}_0, \mathsf{x}_1, ..., \mathsf{x}_N$ is possible. To this purpose, we scan the first matrix column per column. The first column will allow us to place x_0 and x_1 on the real line (according to their arrangement). This results in at most five locations to place x_2 (before; between; after; or on x_0 and x_1). The second column of the matrix tells us where. We repeat this process until all the candidate values $\mathsf{x}_0, \mathsf{x}_1, ..., \mathsf{x}_N$ are placed on the real line. Then, we use the second matrix to place the y-coordinate values $\mathsf{y}_0, \mathsf{y}_1, ..., \mathsf{y}_N$ on the y-axis. If, in this process, we find it impossible to find a location to place one of the x_i or y_i (due to a contradiction), we answer *no*. If we have never found a contradiction at all x- and y-values can be ordered, we are ready to answer *yes*. This ordering process takes $O(N^2)$ time.

If we have found k_x different values $\mathsf{x}_0, \mathsf{x}_1, ..., \mathsf{x}_N$ and k_y different values $\mathsf{y}_0, \mathsf{y}_1, ..., \mathsf{y}_N$, we can draw an example of a polyline that realises M on the

[1] By *arrangement*, we mean which of the cases $\mathsf{x}_i < \mathsf{x}_{i+1}$, $\mathsf{x}_i = \mathsf{x}_{i+1}$ and $\mathsf{x}_i > \mathsf{x}_{i+1}$ holds.

grid $\{0,1,...,k_x-1\} \times \mathbf{R} \cup \mathbf{R} \times \{0,1,...,k_x-1\}$, with vertices belonging to $\{0,1,...,k_x-1\} \times \{0,1,...,k_x-1\}$. This drawing serves as a sample polyline and answers Problem 6.3 (b).

It is clear that the above inspection of the matrix M takes $O(N^2)$ time. The reconstruction of a polyline can be done in the same amount of time. This completes the proof. □

6.4 The polar coordinate representation of a polyline

In this section, we define the polar coordinate representation of a polyline and we describe how to go from the Cartesian coordinate representation to the polar coordinate representation and vice versa. We also express the conditions in the double-cross matrix of a polyline (as given in Thoerem 3.2 for Cartesian coordinates) in the polar coordinate representation. The polar coordinate representation of a polyline and its double-cross matrix are used as a technical tool in Section 6.5.

Definition 6.6. Let $P = \langle(\mathsf{x}_0,\mathsf{y}_0),(\mathsf{x}_1,\mathsf{y}_1),,\ldots,(\mathsf{x}_N,\mathsf{y}_N)\rangle$ be a polyline (in Cartesian coordinate representation) and let $\mathsf{p}_i = (\mathsf{x}_i,\mathsf{y}_i)$, $0 \le i \le N$. The *polar coordinate representation* of the polyline P is the list

$$\langle \ell_1, \theta_1, \ell_2, \theta_2, ..., \ell_{N-1}, \theta_{N-1}, \ell_N \rangle,$$

where ℓ_i is the length of the line segment $\mathsf{p}_{i-1}\mathsf{p}_i$ and θ_i is the counter-clockwise angel at p_i between the line connecting p_i and p_{i-1} and the line connecting p_i and p_{i+1}. □

If at p_i, the polyline turns to the left or goes straight, $\theta_i = 180° - \angle(\overrightarrow{\mathsf{p}_i\mathsf{p}_{i-1}}, \overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}})$ and if at p_i, the polyline turns to the right or returns, $\theta_i = 180° + \angle(\overrightarrow{\mathsf{p}_i\mathsf{p}_{i-1}}, \overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}})$.

So, θ_i captures the (counter-clockwise) change in direction when going from the line segment $\mathsf{p}_{i-1}\mathsf{p}_i$ to the line segment $\mathsf{p}_i\mathsf{p}_{i+1}$. This is illustrated in Figure 6.5.

6.4.1 From the Cartesian coordinate to the polar coordinate representation

To convert a polyline $P = \langle(\mathsf{x}_0,\mathsf{y}_0),(\mathsf{x}_1,\mathsf{y}_1),,\ldots,(\mathsf{x}_N,\mathsf{y}_N)\rangle$ given by the Cartesian coordinates of its vertices to polar coordinate representation is easy. For ℓ_i, we take the length of the line segment $\mathsf{p}_{i-1}\mathsf{p}_i$, that is $|\mathsf{p}_{i-1}\mathsf{p}_i|$.

By definition $\theta_i = 180° - \angle(\overrightarrow{\mathsf{p}_i\mathsf{p}_{i-1}}, \overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}})$ if the polyline turns to the left or goes straight and $\theta_i = 180° + \angle(\overrightarrow{\mathsf{p}_i\mathsf{p}_{i-1}}, \overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}})$ c. Therefore, the angle θ_i is given by the formula

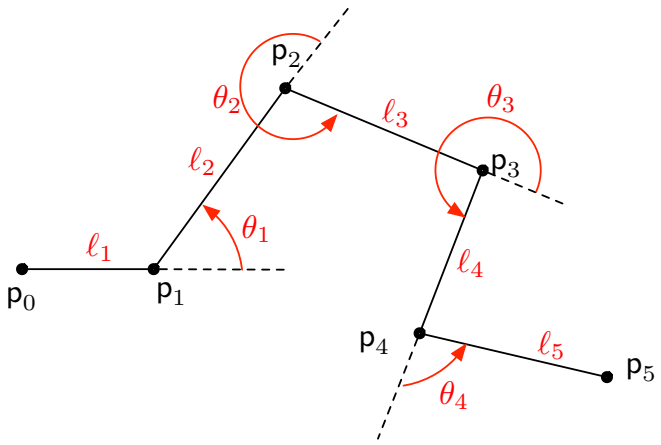


Figure 6.5: The polar coordinates $\langle \ell_1, \theta_1, \ell_2, \theta_2, \ell_3, \theta_3, \ell_4, \theta_4, \ell_5 \rangle$ (in red) of the polyline $\langle \mathsf{p}_0, \mathsf{p}_1, \mathsf{p}_2, \mathsf{p}_3, \mathsf{p}_4, \mathsf{p}_5 \rangle$ (in black).

$$\pi - \arccos\left(\frac{(\mathsf{x}_{i-1} - \mathsf{x}_i, \mathsf{y}_{i-1} - \mathsf{y}_i) \cdot (\mathsf{x}_{i+1} - \mathsf{x}_i, \mathsf{y}_{i+1} - \mathsf{y}_i)}{|(\mathsf{x}_{i-1} - \mathsf{x}_i, \mathsf{y}_{i-1} - \mathsf{y}_i)| \cdot |(\mathsf{x}_{i+1} - \mathsf{x}_i, \mathsf{y}_{i+1} - \mathsf{y}_i)|}\right)$$

if the polyline turns to the left or goes straight, and by the formula

$$\pi + \arccos\left(\frac{(\mathsf{x}_{i-1} - \mathsf{x}_i, \mathsf{y}_{i-1} - \mathsf{y}_i) \cdot (\mathsf{x}_{i+1} - \mathsf{x}_i, \mathsf{y}_{i+1} - \mathsf{y}_i)}{|(\mathsf{x}_{i-1} - \mathsf{x}_i, \mathsf{y}_{i-1} - \mathsf{y}_i)| \cdot |(\mathsf{x}_{i+1} - \mathsf{x}_i, \mathsf{y}_{i+1} - \mathsf{y}_i)|}\right)$$

if the polyline turns to the right or returns.[2]

6.4.2 From the polar coordinate to the Cartesian coordinate representation

Now, we turn to transforming the polar coordinate representation into the classical Cartesian coordinate representation, which is more laborious. Here, we can use some techniques that are also known in the description of robot arms with multiple joints (see, for instance, Chapter 6 of ([CLO97])).

Hereto, we first need some technical results. Let $P = \langle (\mathsf{x}_0, \mathsf{y}_0), (\mathsf{x}_1, \mathsf{y}_1), , \ldots, (\mathsf{x}_N, \mathsf{y}_N) \rangle$ be a polyline and let $\mathsf{p}_i = (\mathsf{x}_i, \mathsf{y}_i)$, $0 \leq i \leq N$. In each vertex $(\mathsf{x}_i, \mathsf{y}_i)$, we create a local coordinate system (X_i, Y_i). The origin of this coordinate system is $(\mathsf{x}_i, \mathsf{y}_i)$ and the positive X_i-axis is points from $(\mathsf{x}_i, \mathsf{y}_i)$ to $(\mathsf{x}_{i+1}, \mathsf{y}_{i+1})$. The Y_i-axis is perpendicular to the X_i-axis in $(\mathsf{x}_i, \mathsf{y}_i)$ in the usual way. This is illustrated in Figure 6.6.

[2]Here, the $\cdot$ in the numerator denotes the *inner product* of two vectors and the $\cdot$ in the denominator is the product of norms.

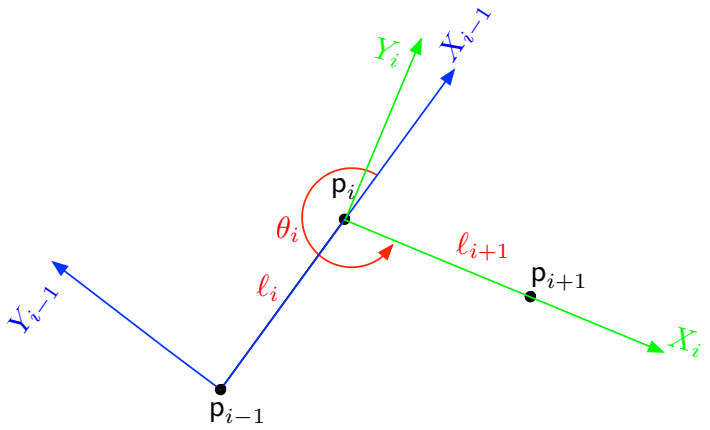


Figure 6.6: The local coordinate systems (X_{i-1}, Y_{i-1}) (in blue) and (X_i, Y_i) (in green) on the vertices p_{i-1} and p_i of a polyline.

The following property is well known from linear algebra and also from the field of multiple joint robot arms (see, Chapter 6, page 262, in [CLO97]).

Property 6.4. Let p_{i-1}, p_i and p_{i+1} be three consecutive vertices on a polyline $P = \langle(\mathsf{x}_0, \mathsf{y}_0), (\mathsf{x}_1, \mathsf{y}_1), , \ldots, (\mathsf{x}_N, \mathsf{y}_N)\rangle$ with $\mathsf{p}_i = (\mathsf{x}_i, \mathsf{y}_i)$, $0 \leq i \leq N$. If a point q in $\mathbf{R}^2$ has coordinates (a_{i-1}, b_{i-1}) and (a_i, b_i), respectively, in the local coordinate systems (X_{i-1}, Y_{i-1}) and (X_i, Y_i), respectively, then

$$\begin{pmatrix} a_{i-1} \\ b_{i-1} \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta_i & -\sin\theta_i & \ell_i \\ \sin\theta_i & \cos\theta_i & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} a_i \\ b_i \\ 1 \end{pmatrix}.$$

□

For a polyline P, given by its polar coordinate representation $\langle \ell_1, \theta_1, \ell_2, \theta_2, \ldots, \ell_{N-1}, \theta_{N-1}, \ell_N \rangle$, we set

$$P_i = \begin{pmatrix} \cos\theta_i & -\sin\theta_i & \ell_i \\ \sin\theta_i & \cos\theta_i & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

From now on, we only consider polylines with $(\mathsf{x}_0, \mathsf{y}_0) = (0, 0)$ and $(\mathsf{x}_1, \mathsf{y}_1) = (1, 0)$, such that (X_0, Y_0) is the standard coordinate system.

The following property, based on the previous property, has a straightforward induction proof.

Property 6.5. Let $P = \langle(\mathsf{x}_0, \mathsf{y}_0), (\mathsf{x}_1, \mathsf{y}_1), , \ldots, (\mathsf{x}_N, \mathsf{y}_N)\rangle$ be a polyline. If a point q in $\mathbf{R}^2$ has coordinates (a_i, b_i) in the local coordinate system (X_i, Y_i), then it has absolute Cartesian coordinates (a_0, b_0), with

$$\begin{pmatrix} a_0 \\ b_0 \\ 1 \end{pmatrix} = P_1 \cdot P_2 \cdots P_i \cdot \begin{pmatrix} a_i \\ b_i \\ 1 \end{pmatrix}.$$

□

The following property tells us what the matrix product $P_1 \cdot P_2 \cdots P_i$ looks like.

Property 6.6. For $1 \leq i < N$, we have

$$P_1 \cdot P_2 \cdots P_i = \begin{pmatrix} \cos\Theta_1^i & -\sin\Theta_1^i & \sum_{j=1}^{i} \ell_j \cos\Theta_1^{j-1} \\ \sin\Theta_1^i & \cos\Theta_1^i & \sum_{j=1}^{i} \ell_j \sin\Theta_1^{j-1} \\ 0 & 0 & 1 \end{pmatrix},$$

where Θ_i^j abbreviates $\theta_i + \theta_{i+1} + \cdots + \theta_j$, for $i \leq j$.

Proof. We proceed by induction on i. For $i = 1$, we have $\ell_1 \cos 0 = \ell_1$ and $\ell_1 \sin 0 = 0$, which clearly gives P_1.

Now, we proceed from i to $i+1$. By the induction hypothesis, $P_1 \cdot P_2 \cdots P_i \cdot P_{i+1}$ equals

$$\begin{pmatrix} \cos\Theta_1^i & -\sin\Theta_1^i & \sum_{j=1}^{i} \ell_j \cos\Theta_1^{j-1} \\ \sin\Theta_1^i & \cos\Theta_1^i & \sum_{j=1}^{i} \ell_j \sin\Theta_1^{j-1} \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos\theta_{i+1} & -\sin\theta_{i+1} & \ell_{i+1} \\ \sin\theta_{i+1} & \cos\theta_{i+1} & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

which is

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$$

with

- $a_{11} = \cos\Theta_1^i \cdot \cos\theta_{i+1} - \sin\Theta_1^i \cdot \sin\theta_{i+1} = \cos(\Theta_1^i + \theta_{i+1}) = \cos(\Theta_1^{i+1})$;
- $a_{12} = -\cos\Theta_1^i \cdot \sin\theta_{i+1} - \sin\Theta_1^i \cdot \cos\theta_{i+1} = -\sin(\Theta_1^i + \theta_{i+1}) = -\sin(\Theta_1^{i+1})$;
- $a_{13} = \ell_{i+1}\cos\Theta_1^i + \sum_{j=1}^{i} \ell_j \cos\Theta_1^{j-1} = \sum_{j=1}^{i+1} \ell_j \cos\Theta_1^{j-1}$;
- $a_{21} = \sin\Theta_1^i \cdot \cos\theta_{i+1} + \cos\Theta_1^i \cdot \sin\theta_{i+1} = \sin(\Theta_1^i + \theta_{i+1}) = \sin(\Theta_1^{i+1})$;
- $a_{22} = -\sin\Theta_1^i \cdot \sin\theta_{i+1} + \cos\Theta_1^i \cdot \cos\theta_{i+1} = \cos(\Theta_1^i + \theta_{i+1}) = \cos(\Theta_1^{i+1})$;

- $a_{23} = \ell_{i+1} \sin \Theta_1^i + \sum_{j=1}^{i} \ell_j \sin \Theta_1^{j-1} = \sum_{j=1}^{i+1} \ell_j \sin \Theta_1^{j-1}$;
- $a_{31} = 0 + 0 + 0 = 0$;
- $a_{32} = 0 + 0 + 0 = 0$; and
- $a_{33} = 0 + 0 + 1 = 1$;

where we have used the well-known formulas for cosinus and sinus of the sum of angles. This gives the desired matrix and concludes the proof. □

The following theorem tells us how to translate from polar coordinates to Cartesian coordinates.

Theorem 6.7. *Let* $P = \langle (\mathsf{x}_0, \mathsf{y}_0), (\mathsf{x}_1, \mathsf{y}_1), , \ldots, (\mathsf{x}_N, \mathsf{y}_N) \rangle$ *be a polyline that is given by its polar coordinate representation* $\langle \ell_1, \theta_1, \ell_2, \theta_2, ..., \ell_{N-1}, \theta_{N-1}, \ell_N \rangle$. *If we assume that* $(\mathsf{x}_0, \mathsf{y}_0) = (0, 0)$ *and* $(\mathsf{x}_1, \mathsf{y}_1) = (1, 0)$, *then*

$$\begin{cases} x_i &= \sum_{j=1}^{i} \ell_j \cos(\theta_1 + \cdots + \theta_{j-1}) \\ y_i &= \sum_{j=1}^{i} \ell_j \sin(\theta_1 + \cdots + \theta_{j-1}) \end{cases}$$

for $2 \leq i \leq N$.

We remark that we could also have written

$$\begin{cases} x_i &= 1 + \sum_{j=2}^{i} \ell_j \cos(\theta_1 + \cdots + \theta_{j-1}) \\ y_i &= \sum_{j=2}^{i} \ell_j \sin(\theta_1 + \cdots + \theta_{j-1}) \end{cases}$$

in the statement of this theorem, since $\ell_1 = 1$, $\cos 0 = 1$ and $\sin 0 = 0$. For esthetic reasons, we will stick to the earlier expressions.

Proof. In the local coordinate system (X_{i-1}, Y_{i-1}), the coordinates op $\mathsf{p}_i = (\mathsf{x}_i, \mathsf{y}_i)$ are $(\ell_i, 0)$. By Property 6.5, the coordinates of p_i in the standard coordinate system (X_0, Y_0) are given by

$$\begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix} = P_1 \cdot P_2 \cdots P_{i-1} \cdot \begin{pmatrix} \ell_i \\ 0 \\ 1 \end{pmatrix}.$$

By Property 6.6, this means

$$\begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix} = \begin{pmatrix} \cos \Theta_1^{i-1} & -\sin \Theta_1^{i-1} & \sum_{j=1}^{i-1} \ell_j \cos \Theta_1^{j-1} \\ \sin \Theta_1^{i-1} & \cos \Theta_1^{i-1} & \sum_{j=1}^{i-1} \ell_j \sin \Theta_1^{j-1} \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \ell_i \\ 0 \\ 1 \end{pmatrix}$$

or

$$\left\{\begin{array}{rcl} x_i &=& \ell_i \cos\Theta_1^{i-1} + \sum_{j=1}^{i-1} \ell_j \cos\Theta_1^{j-1} = \sum_{j=1}^{i} \ell_j \cos\Theta_1^{j-1} \\ y_i &=& \ell_i \sin\Theta_1^{i-1} + \sum_{j=1}^{i-1} \ell_j \sin\Theta_1^{j-1} = \sum_{j=1}^{i} \ell_j \sin\Theta_1^{j-1} \end{array}\right.$$

which concludes the proof. □

6.4.3 The double-cross conditions for polar coordinates

From Theorem 3.2, we know that for a polyline $P = \langle(\mathsf{x}_0, \mathsf{y}_0), (\mathsf{x}_1, \mathsf{y}_1), , \ldots, (\mathsf{x}_N, \mathsf{y}_N)\rangle$ with $\mathsf{p}_i = (\mathsf{x}_i, \mathsf{y}_i)$, $0 \le i \le N$, we have $\mathsf{DC}(\overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}}, \overrightarrow{\mathsf{p}_j\mathsf{p}_{j+1}}) = (C_1\ C_2\ C_3\ C_4)$ with

$$\begin{array}{rcll} C_1 &=& - & \mathrm{sign}((\mathsf{x}_j - \mathsf{x}_i)\cdot(\mathsf{x}_{i+1} - \mathsf{x}_i) + (\mathsf{y}_j - \mathsf{y}_i)\cdot(\mathsf{y}_{i+1} - \mathsf{y}_i)); \\ C_2 &=& & \mathrm{sign}((\mathsf{x}_j - \mathsf{x}_i)\cdot(\mathsf{x}_{j+1} - \mathsf{x}_j) + (\mathsf{y}_j - \mathsf{y}_i)\cdot(\mathsf{y}_{j+1} - \mathsf{y}_j)); \\ C_3 &=& - & \mathrm{sign}((\mathsf{x}_j - \mathsf{x}_i)\cdot(\mathsf{y}_{i+1} - \mathsf{y}_i) - (\mathsf{y}_j - \mathsf{y}_i)\cdot(\mathsf{x}_{i+1} - \mathsf{x}_i)); \text{ and} \\ C_4 &=& & \mathrm{sign}((\mathsf{x}_j - \mathsf{x}_i)\cdot(\mathsf{y}_{j+1} - \mathsf{y}_j) - (\mathsf{y}_j - \mathsf{y}_i)\cdot(\mathsf{x}_{j+1} - \mathsf{x}_j)). \end{array}$$

If now the polyline $P = \langle(\mathsf{x}_0, \mathsf{y}_0), (\mathsf{x}_1, \mathsf{y}_1), , \ldots, (\mathsf{x}_N, \mathsf{y}_N)\rangle$ is given by its polar coordinate representation $\langle\ell_1, \theta_1, \ell_2, \theta_2, ..., \ell_{N-1}, \theta_{N-1}, \ell_N\rangle$, Theorem 3.2 allows us to translate these conditions into polar coordinates.

Again, we assume that $(\mathsf{x}_0, \mathsf{y}_0) = (0, 0)$ and $(\mathsf{x}_1, \mathsf{y}_1) = (1, 0)$. Where needed, we use the abbreviations

$$\left\{\begin{array}{rcl} \mathsf{c}_i &=& \cos\Theta_1^i = \cos(\theta_1 + \cdots + \theta_i); \\ \mathsf{s}_i &=& \sin\Theta_1^i = \sin(\theta_1 + \cdots + \theta_i). \end{array}\right.$$

to control the length of the expressions.

The following theorem gives the double-cross conditions in polar form.

Theorem 6.8. *If now the polyline $P = \langle\mathsf{p}_0, \mathsf{p}_1, \ldots, \mathsf{p}_N\rangle$ is given by its polar coordinate representation $\langle\ell_1, \theta_1, \ell_2, \theta_2, ..., \ell_{N-1}, \theta_{N-1}, \ell_N\rangle$, and if we assume that $\mathsf{p}_0 = (0, 0)$ and $\mathsf{p}_1 = (1, 0)$, then $\mathsf{DC}(\overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}}, \overrightarrow{\mathsf{p}_j\mathsf{p}_{j+1}}) = (C_1\ C_2\ C_3\ C_4)$, for $0 \le i < j < N$, are expressed in polar coordinates as follows:*

$$\begin{array}{rcll} C_1 &=& - & \mathrm{sign}(\sum_{k=i+1}^{j} \ell_k \cos(\theta_{i+1} + \cdots + \theta_{k-1})); \\ C_2 &=& & \mathrm{sign}(\sum_{k=i+1}^{j} \ell_k \cos(\theta_k + \cdots + \theta_j)); \\ C_3 &=& - & \mathrm{sign}(\sum_{k=i+1}^{j} \ell_k \sin(\theta_{i+1} + \cdots + \theta_{k-1})); \text{ and} \\ C_4 &=& & \mathrm{sign}(\sum_{k=i+1}^{j} \ell_k \sin(\theta_k + \cdots + \theta_j)), \end{array}$$

where we agree that the empty sum of angles equals 0.

Proof. From

$$\begin{cases} x_i &= \sum_{k=1}^{i} \ell_k \cos(\theta_1 + \cdots + \theta_{k-1}) \\ y_i &= \sum_{k=1}^{i} \ell_k \sin(\theta_1 + \cdots + \theta_{k-1}), \end{cases}$$

for $0 \leq i \leq N$, we get, for $0 \leq i < j < N$,

$$\begin{cases} x_j - x_i &= \sum_{k=1}^{j} \ell_k \cos(\theta_1 + \cdots + \theta_{k-1}) - \sum_{k=1}^{i} \ell_k \cos(\theta_1 + \cdots + \theta_{k-1}) \\ &= \sum_{k=i+1}^{j} \ell_k \cos(\theta_1 + \cdots + \theta_{k-1}) \\ y_j - y_i &= \sum_{k=1}^{j} \ell_k \sin(\theta_1 + \cdots + \theta_{k-1}) - \sum_{k=1}^{i} \ell_k \sin(\theta_1 + \cdots + \theta_{k-1}) \\ &= \sum_{k=i+1}^{j} \ell_k \sin(\theta_1 + \cdots + \theta_{k-1}) \\ x_{i+1} - x_i &= \sum_{k=1}^{i+1} \ell_k \cos(\theta_1 + \cdots + \theta_{k-1}) - \sum_{k=1}^{i} \ell_k \cos(\theta_1 + \cdots + \theta_{k-1}) \\ &= \ell_{i+1} \cos(\theta_1 + \cdots + \theta_i) \\ y_{i+1} - y_i &= \sum_{k=1}^{i+1} \ell_k \sin(\theta_1 + \cdots + \theta_{k-1}) - \sum_{k=1}^{i} \ell_k \sin(\theta_1 + \cdots + \theta_{k-1}) \\ &= \ell_{i+1} \sin(\theta_1 + \cdots + \theta_i) \\ x_{j+1} - x_j &= \sum_{k=1}^{j+1} \ell_k \cos(\theta_1 + \cdots + \theta_{k-1}) - \sum_{k=1}^{j} \ell_k \cos(\theta_1 + \cdots + \theta_{k-1}) \\ &= \ell_{j+1} \cos(\theta_1 + \cdots + \theta_j) \\ y_{j+1} - y_j &= \sum_{k=1}^{j+1} \ell_k \sin(\theta_1 + \cdots + \theta_{k-1}) - \sum_{k=1}^{j} \ell_k \sin(\theta_1 + \cdots + \theta_{k-1}) \\ &= \ell_{j+1} \sin(\theta_1 + \cdots + \theta_j). \end{cases}$$

If we plug these identities in the equations of Theorem 3.2, we get

$$\begin{array}{rcll} C_1 &=& - & \operatorname{sign}(\sum_{k=i+1}^{j} \ell_k(\mathsf{c}_i\mathsf{c}_{k-1} + \mathsf{s}_i\mathsf{s}_{k-1}) \\ &=& - & \operatorname{sign}(\sum_{k=i+1}^{j} \ell_k \cos(\theta_{i+1} + \cdots + \theta_{k-1})); \\ C_2 &=& & \operatorname{sign}(\sum_{k=i+1}^{j} \ell_k(\mathsf{c}_j\mathsf{c}_{k-1} + \mathsf{s}_j\mathsf{s}_{k-1}) \\ &=& & \operatorname{sign}(\sum_{k=i+1}^{j} \ell_k \cos(\theta_k + \cdots + \theta_j)); \\ C_3 &=& - & \operatorname{sign}(\sum_{k=i+1}^{j} \ell_k(\mathsf{s}_i\mathsf{c}_{k-1} - \mathsf{c}_i\mathsf{s}_{k-1}) \\ &=& - & \operatorname{sign}(\sum_{k=i+1}^{j} \ell_k \sin(\theta_{i+1} + \cdots + \theta_{k-1})); \text{ and} \\ C_4 &=& & \operatorname{sign}(\sum_{k=i+1}^{j} \ell_k(\mathsf{s}_j\mathsf{c}_{k-1} - \mathsf{c}_j\mathsf{s}_{k-1}) \\ &=& & \operatorname{sign}(\sum_{k=i+1}^{j} \ell_k \sin(\theta_k + \cdots + \theta_j)). \end{array}$$

In the last equalities we used the well-known formulas $\sin(\alpha \pm \beta) = \sin\alpha\cos\beta \pm \cos\alpha\sin\beta$ and $\cos(\alpha \pm \beta) = \cos\alpha\cos\beta \mp \sin\alpha\sin\beta$.

□

We remark that all the double-cross conditions in the above theorem are *linear* expressions in the lengths $\ell_1, ..., \ell_{N-1}$. We also remark that an alternative way to write these conditions is

$$\begin{array}{rcll} C_1 &=& - & \operatorname{sign}(\ell_{i+1} + \sum_{k=i+2}^{j} \ell_k \cos(\theta_{i+1} + \cdots + \theta_{k-1})); \\ C_2 &=& & \operatorname{sign}(\sum_{k=i+1}^{j} \ell_k \cos(\theta_k + \cdots + \theta_j)); \\ C_3 &=& - & \operatorname{sign}(\sum_{k=i+2}^{j} \ell_k \sin(\theta_{i+1} + \cdots + \theta_{k-1})); \text{ and} \\ C_4 &=& & \operatorname{sign}(\sum_{k=i+1}^{j} \ell_k \sin(\theta_k + \cdots + \theta_j)). \end{array}$$

Because of the special location of $(\mathsf{x}_0, \mathsf{y}_0) = (0,0)$ and $(\mathsf{x}_1, \mathsf{y}_1) = (1,0)$, we look at a special case of this theorem, namely $i = 0$ and $j = 1$. Here, we have $\mathsf{DC}(\overrightarrow{\mathsf{p}_0\mathsf{p}_1}, \overrightarrow{\mathsf{p}_1\mathsf{p}_2}) = (C_1\ C_2\ C_3\ C_4)$, with

$$\begin{array}{lcll} C_1 & = & - & \text{sign}(1) = -; \\ C_2 & = & & \text{sign}(\ell_1 + \ell_2\mathsf{c}_1 - 1) = \text{sign}(\ell_2\mathsf{c}_1); \\ C_3 & = & - & \text{sign}(0) = 0; \quad \text{and} \\ C_4 & = & & \text{sign}(\ell_2\mathsf{s}_1); \end{array}$$

Because, by Assumption 1, we have $\ell_2 > 0$, we can simplify conditions C_2 and C_4 and we get

$$\begin{array}{lcll} C_1 & = & - & \\ C_2 & = & \text{sign}(\cos\theta_1) & \\ C_3 & = & 0 & \text{and} \\ C_4 & = & \text{sign}(\sin\theta_1). & \end{array}$$

More generally, we look at the following special case of consecutive line segments of a polyline.

Corollary 6.9. If now the polyline $P = \langle \mathsf{p}_0, \mathsf{p}_1, \ldots, \mathsf{p}_N \rangle$ is given by its polar coordinate representation $\langle \ell_1, \theta_1, \ell_2, \theta_2, ..., \ell_{N-1}, \theta_{N-1}, \ell_N \rangle$, and if we assume that $\mathsf{p}_0 = (0,0)$ and $\mathsf{p}_1 = (1,0)$, then $\mathsf{DC}(\overrightarrow{\mathsf{p}_i\mathsf{p}_{i+1}}, \overrightarrow{\mathsf{p}_{i+1}\mathsf{p}_{i+2}}) = (C_1\ C_2\ C_3\ C_4)$, for $0 \leq i < N-1$, are expressed in polar coordinates as follows:

$$\begin{array}{lcl} C_1 & = & -; \\ C_2 & = & \text{sign}(\cos\theta_{i+1}); \\ C_3 & = & 0; \\ C_4 & = & \text{sign}(\sin\theta_{i+1}). \end{array}$$

□

6.5 A realizability test for 45°-polylines

In this section, we describe how it can be decided whether a given $N \times N$ matrix is realizable in the plane by a 45°-polyline. First, we state the problem formally.

Problem 6.10 (**45°-realizability**). Given is an $N \times N$ matrix M of 4-tuples $(C_1\ C_2\ C_3\ C_4) \in \{-, 0, +\}^4$.

(a) Decide whether M is the double-cross matrix of some 45°-polyline in $\mathbf{R}^2$ of size N; and

(b) If the answer to question (a) is *yes*, then produce an example (or many examples) of a 45°-polyline P with $\mathsf{DCM}(P) = M$. □

For the problem of realizability, here again, we may assume, without loss of any generality, that the polyline that realizes a matrix M, if it exists, starts with the unit interval on the x-axis, that is, $\mathsf{p}_0 = (\mathsf{x}_0, \mathsf{y}_0) = (0, 0)$ and $\mathsf{p}_1 = (\mathsf{x}_1, \mathsf{y}_1) = (1, 0)$. Indeed, from Lemma 3.14, we know that similarities preserve the double-cross matrix of a polyline. This also permits us, to use the results on the polar representation from the previous section.

Theorem 6.11. *It can be decided in polynomial time whether an $N \times N$ matrix M of 4-tuples $(C_1\ C_2\ C_3\ C_4) \in \{-, 0, +\}^4$ is the double-cross matrix of some 45°-polyline of size N in $\mathbf{R}^2$. If M is 45°-realizable, also witnesses to this can be produced in polynomial time.*

Proof. We now describe a decision procedure that solves Problem 6.10. Let M be a $N \times N$ input matrix of 4-tuples $(C_1\ C_2\ C_3\ C_4) \in \{-, 0, +\}^4$. In a first step, we determine the polar angles of the polyline, we attempt to construct. In a second step, we see if appropriate lengths of line segments can be found. As an apriori step, we check whether M doesn't have (0 0 0 0) entries on its diagonal or doesn't have the "symmetry" properties. If M fails this symmetry-test, we can already answer *no*, else we proceed.

Step 1 (Determining the angles $\theta_1, \theta_2, ..., \theta_{N-1}$). First, we inspect the entries $M[i, i+1]$, $0 \leq i < N$ of M. Hereto, we use Corollary 6.9. So, C_1 should be $-$ and C_3 should be 0. If this is not the case, we can already answer *no*. From C_2 and C_4 in all entries $M[i, i+1]$, we can determine the angles θ_i as is shown in the following table.

C_2	C_4	θ_i
0	0	answer *no*
0	+	270°
0	−	90°
+	0	180°
+	+	225°
+	−	135°
−	0	0°
−	+	315°
−	−	45°

Obviously, if both $C_2 = \text{sign}(\cos\theta_i)$ and $C_4 = \text{sign}(\sin\theta_i)$ (see Corollary 6.9) are 0, we have an impossible situation. So, at this point, or we have answered *no*, or we know all the angles $\theta_1, \theta_2, ..., \theta_{N-1}$ of a possible realisation of M. In the latter case, we proceed to Step 2.

Step 2 (Determining $\ell_1, \ell_2, ..., \ell_N$). Once, we have determined the angles $\theta_1, \theta_2, ..., \theta_{N-1}$, we can compute all the values $\cos(\theta_{i+1} + \cdots + \theta_{k-1})$, $\cos(\theta_k + \cdots + \theta_j)$, $\sin(\theta_{i+1} + \cdots + \theta_{k-1})$ and $\sin(\theta_k + \cdots + \theta_j)$ that appear in the expressions given in Theorem 6.8. Since all these sums of angles are multiples of $45°$, these cosines and sines will take values as shown in the following table.

α	$\cos\alpha$	$\sin\alpha$
$0°$	1	0
$45°$	$\frac{\sqrt{2}}{2}$	$\frac{\sqrt{2}}{2}$
$90°$	0	1
$135°$	$-\frac{\sqrt{2}}{2}$	$\frac{\sqrt{2}}{2}$
$180°$	-1	0
$225°$	$-\frac{\sqrt{2}}{2}$	$-\frac{\sqrt{2}}{2}$
$270°$	0	-1
$315°$	$\frac{\sqrt{2}}{2}$	$-\frac{\sqrt{2}}{2}$

This means that the double-cross conditions given by Theorem 6.8, together with the constraints that the ℓ_i are strictly positive lengths, can be seen as linear constraint conditions in $\ell_1, \ell_2, ..., \ell_N$ of the form

$$\left\{\begin{array}{lllll} - & \sum_{k=i+1}^{j} \ell_k \cos(\theta_{i+1} + \cdots + \theta_{k-1}) & \alpha_{ij} & 0 & (0 \le i < j < N) \\ & \sum_{k=i+1}^{j} \ell_k \cos(\theta_k + \cdots + \theta_j) & \beta_{ij} & 0 & (0 \le i < j < N) \\ - & \sum_{k=i+1}^{j} \ell_k \sin(\theta_{i+1} + \cdots + \theta_{k-1}) & \gamma_{ij} & 0 & (0 \le i < j < N) \\ & \sum_{k=i+1}^{j} \ell_k \sin(\theta_k + \cdots + \theta_j) & \delta_{ij} & 0 & (0 \le i < j < N) \\ & \ell_i & > & 0 & (0 < i \le N) \end{array}\right. \qquad (*)$$

with $\alpha_{ij}, \beta_{ij}, \gamma_{ij}, \delta_{ij} \in \{=, <, >\}$ determined by the entries of the matrix M. Since all cosines and sines take values in the set $\{0, 1, -1, \frac{\sqrt{2}}{2}, -\frac{\sqrt{2}}{2}\}$, all these conditions are linear in $\ell_1, \ell_2, ..., \ell_N$. Therefore, $(*)$ can be seen as a *linear programming problem*, or at least almost. Normally in a linear programming problem, linear polynomial conditions of the form

$$a_1\ell_1 + a_2\ell_2 + \cdots + a_N\ell_N \ge 0,$$

with the coefficients a_i rational numbers are expected to appear together with the additional conditions

$$\ell_i \ge 0 \quad (0 \le i \le N).$$

So, we are left with three problems:

(1) we have $\ell_i > 0$ for $0 < i \leq N$ and not the traditional $\ell_i \geq 0$;

(2) we have $\alpha_{ij}, \beta_{ij}, \gamma_{ij}, \delta_{ij} \in \{=, <, >\}$ and not the traditional $\geq$; and

(3) we have irrational coefficients $\frac{\sqrt{2}}{2}$, $-\frac{\sqrt{2}}{2}$.

The linear polynomial condition

$$a_1\ell_1 + a_2\ell_2 + \cdots + a_N\ell_N = 0$$

is obviously equivalent to

$$a_1\ell_1 + a_2\ell_2 + \cdots + a_N\ell_N \geq 0 \text{ and } a_1\ell_1 + a_2\ell_2 + \cdots + a_N\ell_N \leq 0.$$

This solves the case of equality. Obviously,

$$a_1\ell_1 + a_2\ell_2 + \cdots + a_N\ell_N < 0$$

is equivalent to

$$-a_1\ell_1 - a_2\ell_2 - \cdots - a_N\ell_N > 0.$$

So, we are left with $a_1\ell_1 + a_2\ell_2 + \cdots + a_N\ell_N > 0$. To solve the problem of the strict inequalities in (1) and (2), there is a known trick from the linear programming literature that we can use (see page 22 of [MG06]). We introduce a new variable δ, which stands for the "gap" between the left and the right side of each inequality and we try to make this gap as large as possible. Then $a_1\ell_1 + a_2\ell_2 + \cdots + a_N\ell_N > 0$ is equivalent to

$$\begin{array}{rl} \text{maximize} & \delta \\ \text{subject to} & a_1\ell_1 + a_2\ell_2 + \cdots + a_N\ell_N - \delta \geq 0 \\ \text{and} & \delta \geq 0. \end{array}$$

And this single δ can be used to deal with several strict inequalities all at once. Indeed, the linear program has now an extra variable δ and the optimal δ is strictly positive exactly when the original system with strict inequalities has a solution.

Let us write the first $2N(N-1)$ linear polynomials appearing in $(*)$ as

$$P_{ij}^{\sigma_{ij}}(\ell_1, \ell_2, ..., \ell_N)\ \sigma_{ij}\ 0,$$

with $0 \leq i < j < N$ and $\sigma_{ij} \in \{\alpha_{ij}, \beta_{ij}, \gamma_{ij}, \delta_{ij}\}$.

We define the sets

$$\begin{array}{rcl} S_{=} & := & \{(i,j,\sigma_{ij} \mid 0 \le i < j < N, \sigma_{ij} \in \{\alpha_{ij}, \beta_{ij}, \gamma_{ij}, \delta_{ij}\} \text{ and } \sigma_{ij} = =\}; \\ S_{>} & := & \{(i,j,\sigma_{ij} \mid 0 \le i < j < N, \sigma_{ij} \in \{\alpha_{ij}, \beta_{ij}, \gamma_{ij}, \delta_{ij}\} \text{ and } \sigma_{ij} = >\}; \text{ and} \\ S_{<} & := & \{(i,j,\sigma_{ij} \mid 0 \le i < j < N, \sigma_{ij} \in \{\alpha_{ij}, \beta_{ij}, \gamma_{ij}, \delta_{ij}\} \text{ and } \sigma_{ij} = <\}. \end{array}$$

Now, we can see that $(*)$ can be converted to the following linear programming problem:

$$\begin{array}{rrll} \text{maximize} & \delta & & \\ \text{subject to} & P_{ij}^{\sigma_{ij}}(\ell_1, \ell_2, ..., \ell_N) & \ge 0 & \text{for } (i,j,\sigma_{ij}) \in S_{=} \\ & -P_{ij}^{\sigma_{ij}}(\ell_1, \ell_2, ..., \ell_N) & \ge 0 & \text{for } (i,j,\sigma_{ij}) \in S_{=} \\ & P_{ij}^{\sigma_{ij}}(\ell_1, \ell_2, ..., \ell_N) - \delta & \ge 0 & \text{for } (i,j,\sigma_{ij}) \in S_{>} \\ & -P_{ij}^{\sigma_{ij}}(\ell_1, \ell_2, ..., \ell_N) - \delta & \ge 0 & \text{for } (i,j,\sigma_{ij}) \in S_{<} \\ & \ell_i - \delta & \ge 0 & \text{for } 0 < i \le N \\ \text{and} & \delta & \ge 0. & \end{array}$$

What remains is Problem (3), namely that we may have the irrational coefficients $\frac{\sqrt{2}}{2}$ and $-\frac{\sqrt{2}}{2}$ in our linear programming problem. However, a result by Adler and Beling ([AB94]) shows that linear programming with algebraic coefficients also has a time complexity that is a polynomial of

(i) the "rank" of the linear system of inequalities, which applied to our example is $O(N^2)$;

(ii) the degree of the extension of the rationals in which we work, which is in our case 2, since $(\mathbf{Q}(\sqrt{2}) : \mathbf{Q}) = 2$; and

(iii) a quantity related to the conjugate norm of the linear system, which in our case is $O(N^2 \log N)$.

So, we can conclude that our linear programming problem can be solved in polynomial time in N. This completes the proof. □

6.6 Convexity properties of 45°-polylines

In this section, we discuss some implications of Theorem 6.11 on the convexity of the solution set, determined by a matrix that is realisable in the plane by a 45°-polyline.

From Step 1 of the proof of Theorem 6.11, it follows that a matrix M that is realisable by a 45°-polyline determines the angles θ_i uniquely for $0 < i < N$. This proves the following corollary.

Corollary 6.12. If an $N \times N$ matrix M of 4-tuples $(C_1\ C_2\ C_3\ C_4) \in \{-, 0, +\}^4$ is realizable by two 45°-polylines P_1 and P_2 that start with the line segment connecting $(0, 0)$ and $(1, 0)$ and have polar-coordinate representations $\langle \ell_1, \theta_1, \ell_2, \theta_2, ..., \ell_{N-1}, \theta_{N-1}, \ell_N \rangle$ and $\langle \ell'_1, \theta'_1, \ell'_2, \theta'_2, ..., \ell'_{N-1}, \theta'_{N-1}, \ell'_N \rangle$, respectively, then $\theta_i = \theta'_i$ for $0 < i < N$. □

Also from the proof of the previous theorem, the following property follows.

Corollary 6.13. If an $N \times N$ matrix M of 4-tuples $(C_1\ C_2\ C_3\ C_4) \in \{-, 0, +\}^4$ is realizable by two 45°-polylines P_1 and P_2 (that start with the line segment connecting $(0, 0)$ and $(1, 0)$) and have polar-coordinate representations $\langle \ell_1, \theta_1, \ell_2, \theta_2, ..., \ell_{N-1}, \theta_{N-1}, \ell_N \rangle$ and $\langle \ell'_1, \theta'_1, \ell'_2, \theta'_2, ..., \ell'_{N-1}, \theta'_{N-1}, \ell'_N \rangle$, respectively, then for any real numbers $\alpha_1, \alpha_2 > 0$, the 45°-polyline given by the polar coordinate representation $\langle \alpha_1 \cdot \ell_1 + \alpha_2 \cdot \ell'_1, \theta_1, \alpha_1 \cdot \ell_2 + \alpha_2 \cdot \ell'_2, \theta_2, ..., \alpha_1 \cdot \ell_{N-1} + \alpha_2 \cdot \ell'_{N-1}, \theta_{N-1}, \alpha_1 \cdot \ell_N + \alpha_2 \cdot \ell'_N \rangle$ also realizes M.

Proof. Corollary 6.12 takes care of the angles. From Step 2 of the proof of the previous theorem it follows that if P_1 and P_2 are realizations of a matrix M their lengths satisfy the same set of linear conditions of the form $a_1\ell_1 + a_2\ell_2 + \cdots + a_N\ell_N\ \alpha\ 0$, with $\alpha \in \{<, =, >\}$. Suppose that we have

$$\begin{cases} a_1\ell_1 + a_2\ell_2 + \cdots + a_N\ell_N & > & 0 \text{ and} \\ a_1\ell'_1 + a_2\ell'_2 + \cdots + a_N\ell'_N & > & 0 \end{cases}$$

for P_1 and P_2, for any of these linear conditions. Since both $\alpha_1 > 0$ and $\alpha_2 > 0$, we also have

$$\begin{cases} \alpha_1 \cdot (a_1\ell_1 + a_2\ell_2 + \cdots + a_N\ell_N) & > & 0 \text{ and} \\ \alpha_2 \cdot (a_1\ell'_1 + a_2\ell'_2 + \cdots + a_N\ell'_N) & > & 0. \end{cases}$$

So, also the sum of the two left hand sides,

$$\alpha_1 \cdot (a_1\ell_1 + a_2\ell_2 + \cdots + a_N\ell_N) + \alpha_2 \cdot (a_1\ell'_1 + a_2\ell'_2 + \cdots + a_N\ell'_N)$$

will be strictly larger than 0. The same argument hold when α is $=$ or $<$. This completes the proof. □

We end this chapter with the following *convexity property* for 45°-polylines.

Corollary 6.14. If an $N \times N$ matrix M of 4-tuples $(C_1\ C_2\ C_3\ C_4) \in \{-, 0, +\}^4$ is realizable by two 45°-polylines P_1 and P_2 (that start with the line segment connecting $(0, 0)$ and $(1, 0)$) and have polar-coordinate representations $\langle \ell_1, \theta_1, \ell_2, \theta_2, ..., \ell_{N-1}, \theta_{N-1}, \ell_N \rangle$ and $\langle \ell'_1, \theta'_1, \ell'_2, \theta'_2, ..., \ell'_{N-1}, \theta'_{N-1}, \ell'_N \rangle$, respectively,

then for any λ with $0 \leq \lambda \leq 1$, the $45°$-polyline given by the polar coordinate representation

$$\begin{aligned}\langle &\lambda \cdot \ell_1 + (1-\lambda) \cdot \ell'_1, \lambda \cdot \theta_1 + (1-\lambda) \cdot \theta'_1, \\ &\lambda \cdot \ell_2 + (1-\lambda) \cdot \ell'_2, \lambda \cdot \theta_2 + (1-\lambda) \cdot \theta'_2, ..., \lambda \cdot \ell_{N-1} + (1-\lambda) \cdot \ell'_{N-1}, \\ &\lambda \cdot \theta_{N-1} + (1-\lambda) \cdot \theta'_{N-1}, \lambda \cdot \ell_N + (1-\lambda) \cdot \ell'_N \rangle\end{aligned}$$

also realizes M.

Proof. From Corollary 6.12, it is clear that $\lambda \cdot \theta_i + (1-\lambda) \cdot \theta'_i = \theta_i = \theta'_i$ for $0 < i < N$.

For λ with $0 \leq \lambda \leq 1$, we observe that if we take $\lambda = 0$, we get P_2 and if we take $\lambda = 1$, we get P_1. This leaves us with the case $0 < \lambda < 1$. But here, both λ and $1 - \lambda$ are strictly larger than 0 and Corollary 6.13 applies with $\alpha_1 = \lambda$ and $\alpha_2 = 1 - \lambda$. This completes the proof. □

Part II

Map matching techniques for trajectory data

7

Introduction to map matching

7.1 What is map matching?

Nowadays, one of the most popular technologies, used by location-aware devices, is GPS. Although most people use a GPS as a navigational tool, it can also be used for storing the position of a moving object (for instance, a car or a pedestrian) for data analysis. For instance, we can analyse the routes followed by a person and then study why that person chose one road instead of another. The main disadvantage of using GPS obtained coordinates is that they are not always very accurate and will not always match the road followed by a car or a pedestrian. Therefore, most GPS devices, used in cars, account for these errors by mapping the measured location to the street that was followed, instead of just displaying the location information received from a satellite. Many algorithms were developed for this task. We give an overview of them in Section 7.4. The general problem of matching GPS positions to a road network is called *map matching.*

7.2 Map matching: issues and problem statement

Typically, when the position of a car or a pedestrian is monitored using GPS, around ninety-five percent of the recorded time-space points fall outside the actually followed road network. Besides the measurement errors, there are also other problems with real-world data. We think of traffic jams and gaps between measured points, produced by some interference in the satellite signal (for instance, when moving in tunnels).

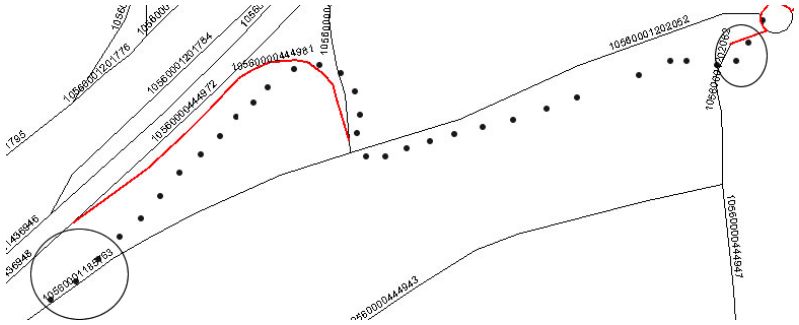


Figure 7.1: A GPS signal that moves away from the followed road before it returns to it (from [Ghy07]).

As an example, we consider Figure 7.1, where the measures time-space points start moving towards a road that is north with respect to the road that was actually followed. After some time the GPS measured points appear to return to the correct road. This is a typical situation in real data, due to uncertainty in the measurement of the object's position. A second problem, that map matching algorithms have to deal with, are large gaps in the measured data. Gaps, illustrated in Figures 7.2 and 7.3, appear due to several reasons:

(a) a faulty GPS signal;

(b) an interruption of the communication; or

(c) ambiguous data.

For (a), a faulty GPS signal may be caused by the bad reception of the coordinates. This is quite unlikely to occur, however.

For (b), an interruption of the communication between the GPS satellite and the device of the moving object, may be due, for example, to a densely forested area, a tunnel, or high buildings. If a GPS signal is blocked because of tall buildings, we speak of *urban canyons*. For example, Figure 7.2 shows a gap of 140 meters (indicated by an ellipse on the map and the satellite image), while Figure 7.3 shows a gap due to the existence of a tunnel. These two examples show that, sometimes, it is useful and necessary to combine information from different sources.

For (c), we remark, as an example, that sometimes roads may run parallel for some distance. This may cause GPS data to be ambiguous. Figure 7.4, where two roads are possibly correct, illustrates this. Sometimes this problem can be easily solved by examining the timestamp of each GPS point, and looking, for instance, at the direction of both road segments, that is, if they are one-way or not.

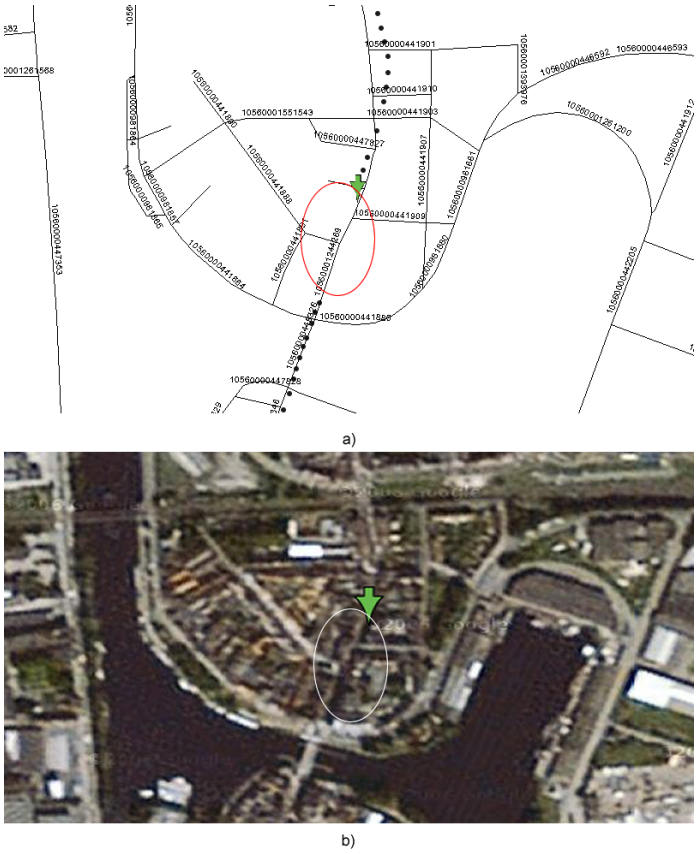


Figure 7.2: a) a large gap in a trajectory sample, with b) a map of the same area (from [Ghy07]).

We conclude that all these types of problems make mapping the measured time-space points to the road network necessary. The following classical definition or description of the *map matching problem* is given by White, Bernstein and Kornhauser [WBK00]: *An object is moving along a finite system (or set) of streets,* $\overline{N}$*. A location-aware device such as GPS provides an estimate for the vehicles location at a finite number of points in time, denoted by* $\{0, 1, \ldots, t\}$*. The vehicles actual location at time* t *is denoted by* $\overline{P}^t$ *and the estimate is denoted* P^t*.* Map matching *is the process of determining the street in* $\overline{N}$ *that contains* P^t*. That is, to determine the street that the vehicle is on at time* t*.*

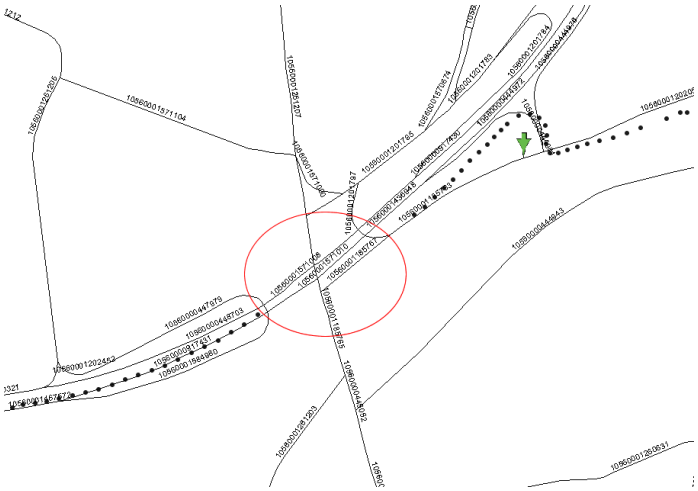


Figure 7.3: A tunnel producing a gap, indicated by an ellipse (from [Ghy07]).

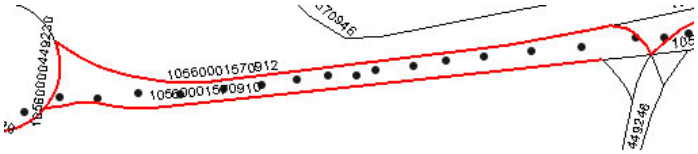


Figure 7.4: Both road segments (in red) could match the measured GPS points (from [Ghy07]).

7.3 Classifications of map-matching algorithms

There are two important ways to classify map matching algorithms:

- offline versus online; and
- low sampling rate versus high sampling rate.

We discuss these in this section.

7.3.1 Offline versus online map matching

Map matching algorithms are mainly classified in the operational way that they can be used. One type of use is *online*. This means they can be used in a real-time situation, where the object is still on the move. An example is a GPS route planner in a driving car. This type of algorithm is usually referred to as a *local map matching algorithm*. It is called local, since the full trajectory data is not available yet, but only the previous and current local data is available.

Figure 7.5: Example of a misclassified sampling rate (from [Bam12]).

The other type of map matching is *offline*. This means the complete trajectory data of a moving object are available to the map matching algorithm. Here, since more information is available, this type of algorithm generally provides better results.

Online algorithms can also be used offline: it does not matter if only a part of the trajectory data or the full trajectory data are available. However, not all offline algorithms work in online situations, since they generally need more time to calculate the path or cannot function without the full trajectory data.

7.3.2 Low sampling rate versus high sampling rate map matching

The rate at which the GPS trajectory data are sampled depends entirely on the source of the data. This can be anywhere from one sample point per second up to one sample point per fifteen minutes (or, basically, any rate the application desires). Lower sampling rates are often used when trying to save energy or disk space. This can occur when you have to process data from a lot of different sources at once.

Most of the literature mentioning a *sampling rate*, defines the threshold between a low and high sampling rate to be a fixed rate. Often (strictly) more than one sample point per two minutes is used as the limit for high sampling. However, this makes it possible for data to be misclassified [Bam12] . We consider, for example, the situation depicted in Figure 7.5. The sampling rate of the data is one sample per two minutes, which would be classified as low. But since the data points are all on or near the same road segment, a map matching algorithm, which assumes a high sampling rate, would also work here.

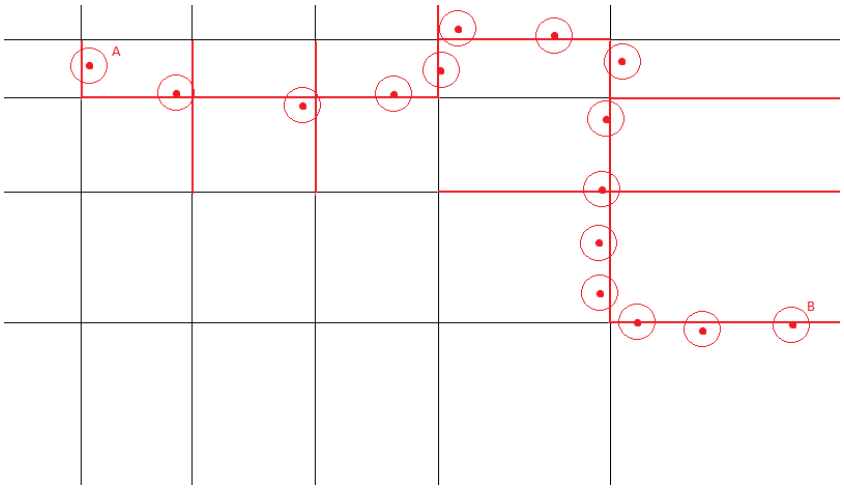


Figure 7.6: An example of a high sampling rate. The red points are the GPS points, the circles around them contain the confidence area. The road segments selected are drawn in red. Since a trajectory from starting point A to end point B exists, this is classified as high sampling rate (from [Bam12]).

Therefore, we propose the following, novel way of classifying the data: *in order to classify the sampling rate based on something else than the absolute time, we use the number of samples per road segment. If there is at least one sample per road segment traveled, we can assume the sampling rate is high enough for the data to be used with a regular algorithm.*

To achieve this, we calculate a confidence area around each GPS point and limit the road network to the road segments that fall within the confidence area. This is done by drawing a circle around the sample point, with a diameter equal to the possible measurement error. In practice, this is often in the range of twenty-five meters. If there is a trajectory possible from segment of a road, given by the start point to a segment of the same road segment given by the end point, we classify it as a *high* sampling rate. If this is not possible and there are gaps in the trajectory, it is classified as *low* sampling rate. An example of a high sampling rate is given in Figure 7.6. An example of a low sampling rate can be seen in Figure 7.7.

Apart from the type of data for which map matching algorithms are designed to work, these algorithms can be further on divided into three categories. This will be further discussed in next section, which gives an overview of existing map matching algorithms.

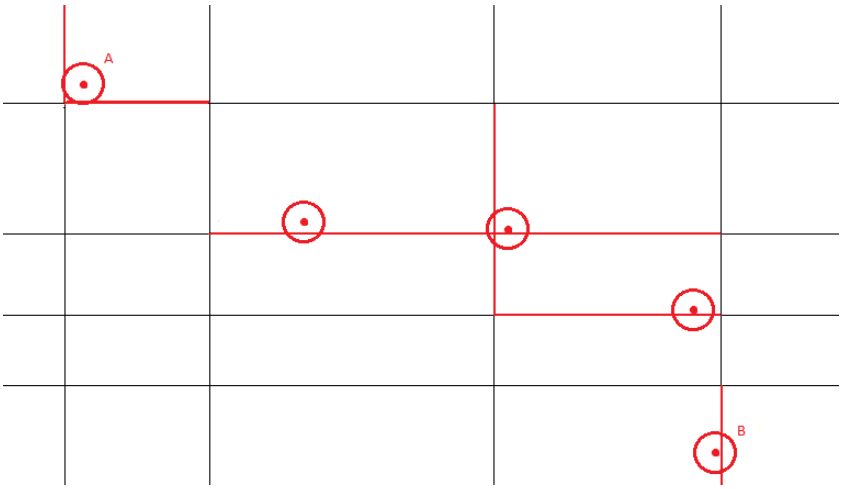


Figure 7.7: An example of a low sampling rate. The red points are the GPS points, the circles around them contain the confidence area. The road segments selected are drawn in red. Since there is no trajectory from point A to B, this is classified as low sampling rate (from [Bam12]).

7.4 Existing Map Matching algorithms

In this section, we give an overview of the existing algorithms for map matching. These algorithms can be divided in three categories based on the way they compute their solution. We will first discuss *geometric methods* (in Section 7.4.1) that additionally use geometric information of the original road network, and do not consider topological information. Next, we discuss *topological methods* (in Section 7.4.2) that make use of the geometry of the road network as well as of the connectivity and contiguity of the links in it. Finally, we consider *probabilistic methods* (in Section 7.4.3). These methods define an elliptical or rectangular confidence region. The error region is superimposed on the road network to identify a relevant road segment. If the error region contains more than one street, probabilistic algorithms perform a weighted search on the candidate streets.

Some authors proposed *fuzzy logic* approaches as a fourth category, but the most popular fuzzy methods can be categorised in one of the above three classes.

7.4.1 Geometric methods

The algorithms in this section provide solutions to the map matching problem by using the shape of the road segments. They ignore the way in which road

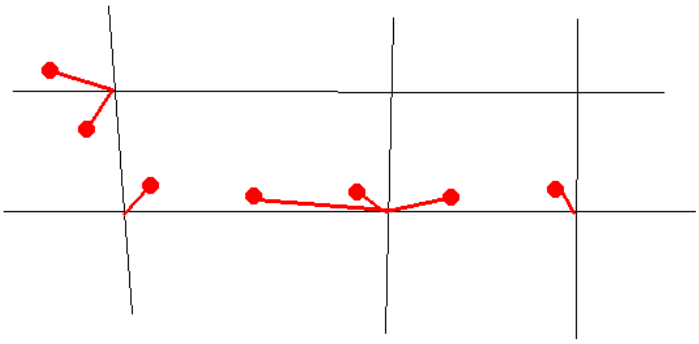


Figure 7.8: An example of point-to-point matching (from [Bam12]).

segments are connected. These algorithms are the most straightforward and can be called, in some sense, naive.

The expositions of the algorithms, that we discuss in this section, are based on [BK98] and [WBK00].

7.4.1.1 Point-to-point matching

In *point-to-point matching*, each time-space or sample point is simply mapped to the nearest node (or vertex) in the road network, as is illustrated in Figure 7.8.

7.4.1.2 Point-to-curve matching

Point-to-curve matching is similar to *point-to-point* matching, but time-space points are matched to the closest road segment, instead of the closest node in the road network. This is illustrated in Figure 7.9. Improvements of this type of algorithms exist, which use heading data (that is, the direction the object is heading). If the heading of the object is not compatible to the heading of the road segment (for instance, in a one-way road), this road segment is discarded.

There are several problems with the *point-to-point* and the *point-to-curve* matching algorithms. A first problem is that there could be sampling errors in the provided data. Noise in the sampling rate can occur, for example, when two consecutive time-space points are further apart in time than they should be (for instance, given background information such as speed limitations). This problem is illustrated in Figure 7.10.

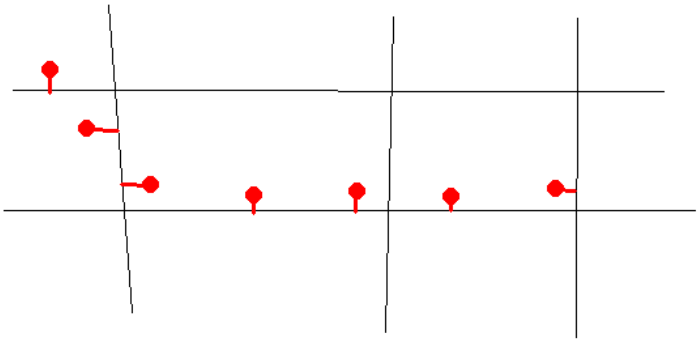


Figure 7.9: An example of point-to-curve matching (from [Bam12]).

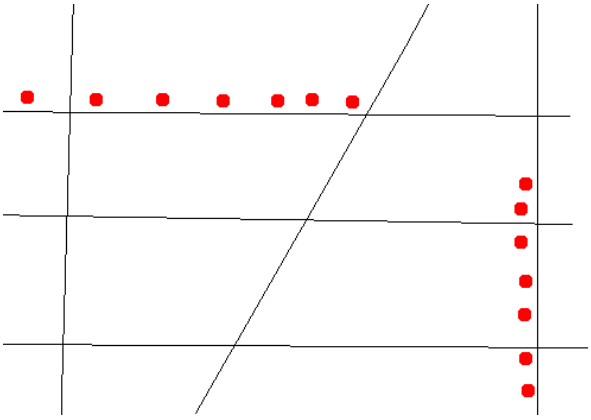


Figure 7.10: An example of a case where a naive algorithm fails due to sampling errors. In this case, there would be a gap in the calculated trajectory (from [Bam12]).

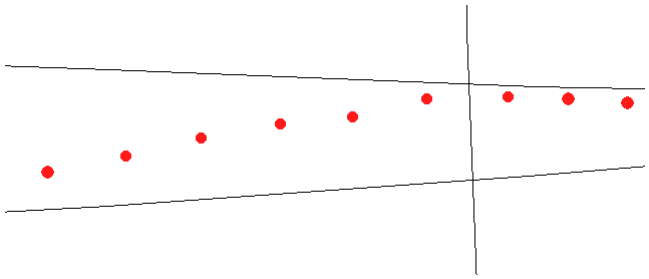


Figure 7.11: An example of a case where a naive algorithm fails due to the inaccuracy of GPS data (from [Bam12]).

Another problem is the inaccuracy of GPS data. Due to this inaccuracy, a recorded time-space point can be closer to a road that is not in the actual trajectory. This typically occurs when there are close by parallel roads or in the vicinity of crossings. This problem is illustrated in Figure 7.11.

Other disadvantages are that these algorithms only look at geometric information, and do not take into account what road segment the previous point was matched with.

7.4.1.3 Point-to-curve matching with topological information

Point-to-curve matching with topological information uses connectivity information to match points with a road segment [WBK00]. The topology of the road network is used, whenever possible, to determine candidate nodes. Only road segments that are directly reachable from the current segment are considered. But, if the algorithm has low confidence in the previous match, it will switch to the *point-to-curve matching* algorithm from the previous section. In [WBK00], confidence is reached when the error is less than 0.15 km and two times the average error. To get an intuition of how this approach works, we look at Figure 7.12. If we know the object was originally at node n, we know that P_1 can only be matched to road segments r_1, r_2, r_3 or r_4 (given a sufficiently high sampling rate). Given that the sampling rate is high, it is possible to know that also P_2 and P_3 can only be matched to these road segments.

7.4.1.4 Curve-to-curve matching

Curve-to-curve matching matches arcs, produces by a trajectory sample, with an arc on the road network [WBK00]. This is achieved by measuring the distance between the two arcs, and choosing the arc on the road network at the smallest distance. A possible problem, caused by this method, is illustrated in Figure 7.13, where a trajectory arc is matched to the wrong road segment.

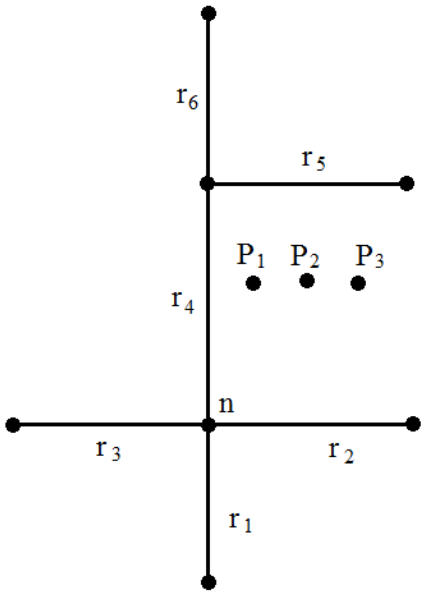


Figure 7.12: An example of point-to-curve matching with topological information (from [WBK00]).

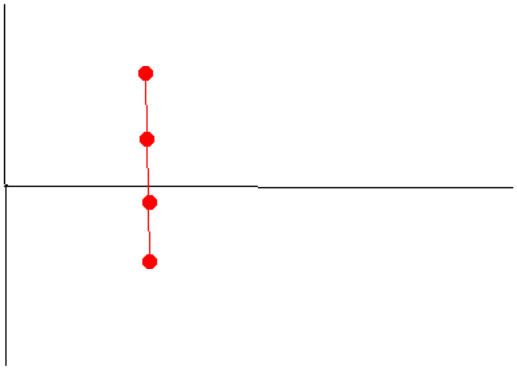


Figure 7.13: Curve-to-curve matching would match the sample points to the horizontal curve, since the distance to this curve is the smallest (from [Bam12]).

7.4.2 Topological analysis

The algorithms in this section provide solutions to the map matching problem by using the way in which nodes are connected within the road network. Generally speaking, they give a weight to each road segment, and find the best path considering these weights.

7.4.2.1 Hidden Markov model map matching

A *hidden Markov model* models a process, following a path throughout many possible states [Edd04]. Each state transition has a probability and the measurements of the states are uncertain. An example of this is given by a person that performs certain activities based on the weather conditions. In this case, based on the observations of the activities of a person on a certain day, it is possible to guess the weather conditions of that day. Let us suppose that there could be two states of the weather that work as a Markov chain: rainy and sunny. But these are not observed directly. They are hidden. We call the activities, that a person performs during some day, the observations. In a system based on a hidden Markov model, the parameters could be the weather trends and the average activities of a person.

Applied to map matching, a state could be a road segment, with each transition between road segments having some probability. The state measurements are the measured sample points.

For each time-space point, P_t, and for each road segment, r_i, there is a measurement probability $p(P_t|r_i)$. This probability expresses how likely it is that we observe the time-space measurement P_t actually on the road segment r_i. In [NK09], this is modeled using the great circle distance between the road segment and the time-space point, as depicted in Figure 7.14. Usually, only the probabilities of roads that are close to the time-space measurement are taken into consideration.

The noise on GPS localization is modeled as zero-mean Gaussian, which gives us the following formula (of a normal distribution), which uses the great circle distance:

$$p(P_t|r_\mathrm{i}) = \frac{1}{\sqrt{\pi}\sigma_P} e^{-\frac{1}{2}\left(\frac{\left\|P_t - x_{\mathrm{t,i}}\right\|_{\text{great circle}}}{\sigma_P}\right)^2}.$$

The expression σ_P is the standard deviation of time-space measurements.

Next, we need to calculate the transition probabilities. Each time-space point has a list of possible road matches. Given two consecutive time-space points, the transition probability gives the chance of a vehicle traveling between two of these possible road matches. In [NK09], the difference between

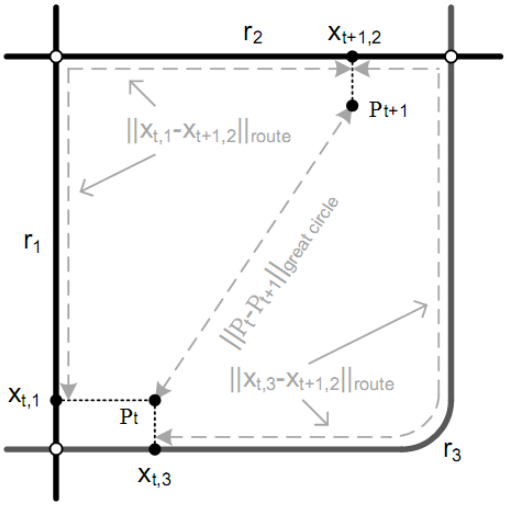


Figure 7.14: Great circle and route distance. The road segments are r_1, r_2 and r_3 and P_t and P_{t+1} are measured time-space points (from [NK09]).

the great circle distance and the route distance is used to calculate this probability.

Using the transition and measurement probabilities, the *Viterbi algorithm* [Vit67] is used to find the most likely sequence of road segments. For a transitions from time t_1 to time t_2, the Viterbi algorithm, first, selects the M most likely paths. Next, the M most likely survivors are extended to time t_3 by using the transitions from time t_2 to t_3. Once more, only the M most likely paths are selected. This process continues until we reach the end time. An example of an input network is given in Figure 7.15. Figure 7.16 illustrates the working of the Viterbi algorithm for $M = 4$ and six sample times. In this example, a shorter path has a higher likelihood.

7.4.2.2 Related work

There are other algorithms, which also use a *hidden Markov model* for map matching. In [Hum06], the GPS noise assumption also uses a Gaussian distribution. The main differences with the algorithm described above is the use of another term in the calculation of measurement probabilities. This term represents the heading mismatch between the vehicle on the road network. The heading indicates the direction the vehicle aimed at (for instance, North-West). This information is not used in the above algorithm since heading information is not always available. Heading data can also be very inaccurate when it is calculated using the GPS points. The second main difference is the

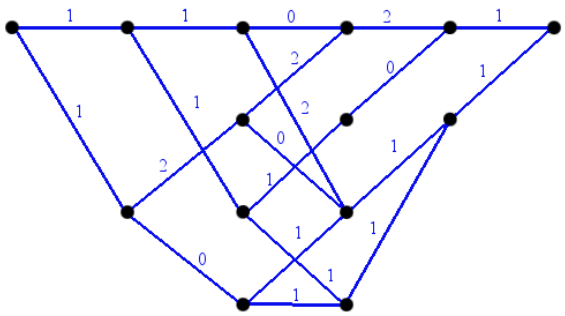


Figure 7.15: The transitions with their lengths in blue (from [Bam12]).

calculation of the transition probability. In [Hum06], only roads immediately adjacent to the current GPS point are used.

Another, similar algorithm is given in [KLH07]. The main difference with the previous approaches is the calculation of the transition probabilities. Whereas, in the above algorithm, the difference in distance is used, this algorithm uses the difference in time.

7.4.2.3 Greenfeld's algorithm

Greenfeld's algorithm [Gre02] only uses the coordinate information of the object and does not use any knowledge about the expected traveling route, the traveling speed and heading. It consists of two sub-routines: `InitialMapping` to determine an initial match and `Map` to calculate weight.

`InitialMapping` The InitialMapping sub-routine uses the point-to-point algorithm from Section 7.4.1.1 to find an initial node on the road network. Next, it determines all the road segments in the road network connected to this node. When the next time-space point becomes available, it maps the initial point to one of the previously found road segments.

This sub-routine is called when the first time-space point is being processed, when the distance between two consecutive time-space points exceeds a chosen distance tolerance, or when the `Map` sub-algorithm is unable to map the time-space point to a road segment.

`Map` To determine to what road segment a time-space point will be matched, a weight W is calculated for several candidate road segments. Tree measures

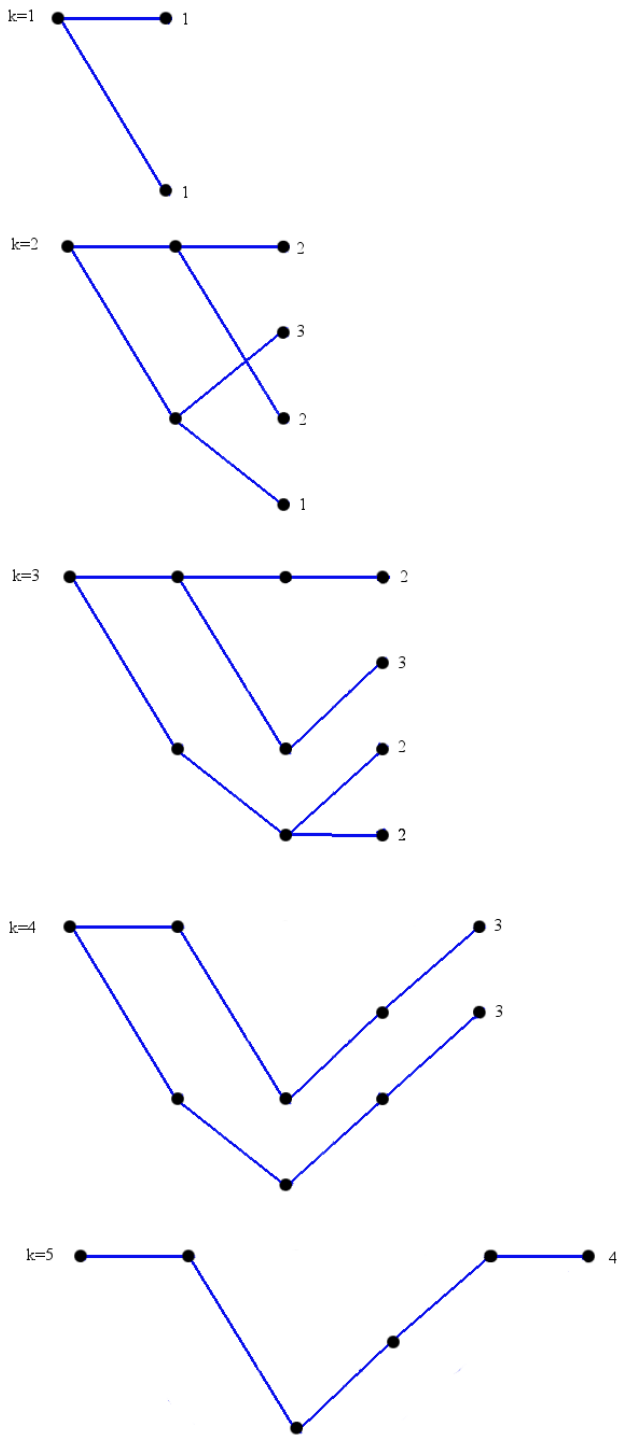


Figure 7.16: The working of the Viterbi algorithm for six sample times (from [Bam12]).

for similarity are used:

W_D is the weight that expresses the distance from the time-space point to the road segment;

W_{AZ} is the weight for direction (azimuth) and expressed how similar the line between consecutive sample points is to the direction of the road segment;

W_I is a weight that express whether or not the line between two consecutive time-space points intersect the road segment.

The total weight W is calculated as

$$W = W_D + W_{AZ} + W_I.$$

7.4.2.4 Local look-ahead

The *local look-ahead algorithm* [BPSW05] is very similar to the one described in Section 7.4.2.3. The calculation of the weight is different, however. Here, the weight for intersection is not used, but the weight is defined as

$$W = W_D + W_{AZ}.$$

To improve this weighting algorithm, for each candidate road segment r_j, the score of the best candidates among the exiting road segments $r_{j,k}$ is recursively calculated as

$$W(P_{i,}, r_j) = \sum_{k,l=0}^{depth} W(P_{i+k,}, r_{j+l}),$$

where P_i is a time-space point.

This method explores which branch would be the best match instead of simply looking for the best edge. An example of this is given in Figure 7.17.

7.4.2.5 A global algorithm that uses the Fréchet distance

The *global algorithm using Fréchet distance*, in [BPSW05], tries to find a curve (that is, a number of connected road segments) in the road network that is as close as possible to the followed trajectory (that is, the curve formed by a sequence of GPS points). As a measure of distance between two curves, the *Fréchet distance* or *weak Fréchet distance* can be used.

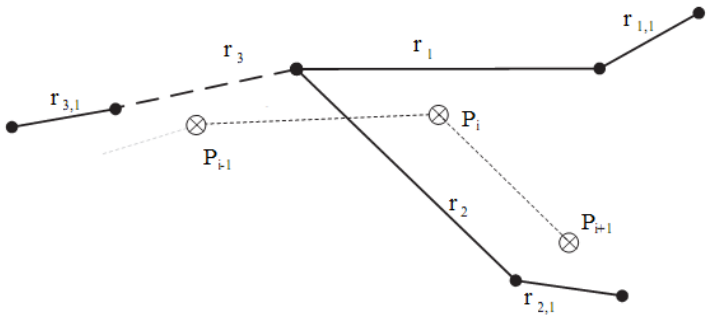


Figure 7.17: An example of the look-ahead algorithm. Point P_i will be matched to road segment r_2 rather than r_1, because of the weight of point P_{i+1} (from [Gre02]).

The Fréchet distance The Fréchet distance is usually explained by the analogy of a person walking his dog on a leash. The person is walking on one curve, while the dog is walking on another curve. Both person and dog may vary speed or stop altogether. But they are not allowed to walk back. The *Fréchet distance* is the length of the shortest leash length that makes it possible to traverse both curves in this way. The *weak Fréchet distance* allows traversing backwards on the curves.

Assume A and B are two curves, the Fréchet distance is formally defined as

$$F(A,B) = \inf_{\alpha,\beta} \max_{t\epsilon[0,1]} \{d(A(\alpha(t)), B(\beta(t)))\},$$

where reparameterizations $\alpha, \beta : [0,1] \rightarrow [0,1]$ of A and B are continuous, non-decreasing surjections. In the above formula, d is the standard Euclidean distance function. See Figure 7.18 for an example. The weak Fréchet distance removes the non-decreasing requirement.

The free-space diagram and surface algorithm To check whether there is a curve A on the road network with at Fréchet distance at most ε (for $\varepsilon > 0$) to a moving object's trajectory B, a *free-space diagram* is used. It is defined as

$$D_\varepsilon(A,B) = \left\{(\alpha,\beta)\epsilon[0,1]^2 \mid d(A(\alpha), B(\beta)) \leq \varepsilon\right\}.$$

Such a curve exists if there is a path in the free-space diagram $D_\varepsilon(A,B)$ from the lower left corner to the upper right which is monotone in both coordinates. For the weak Fréchet distance, the path does not have to be monotone. An example of a free-space diagram is given in 7.19.

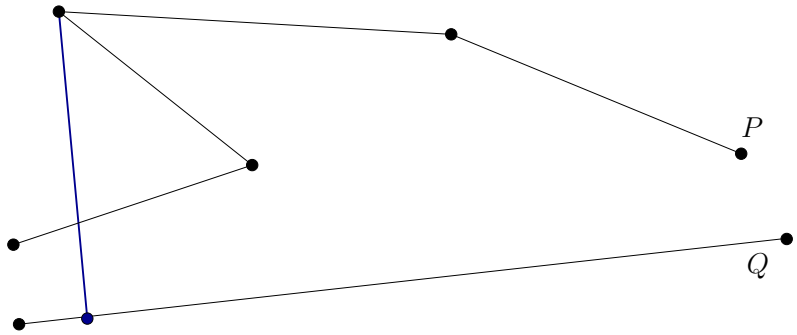


Figure 7.18: The Fréchet distance between P and Q shown in blue.

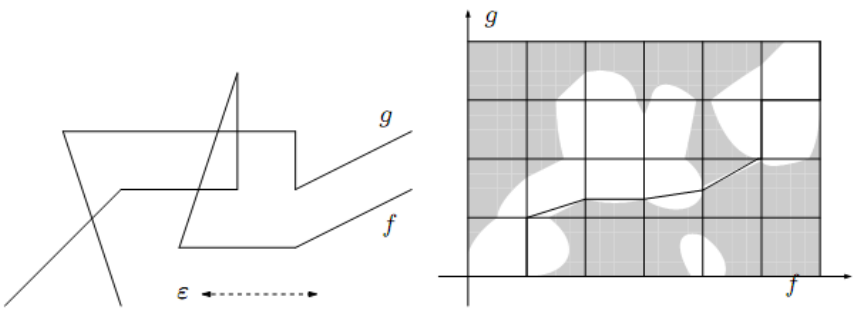


Figure 7.19: A free-space diagram. Here, f and g are two polygonal curves and ε is a distance. On the right the corresponding free-space diagram is given. The holes (white space) in the diagram are the places where the distance between f and g is below ε (from [BPSW05]).

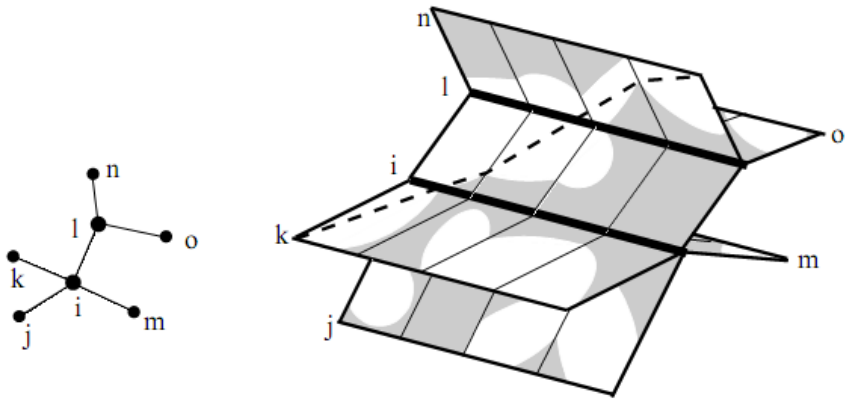


Figure 7.20: A free-space surface. The dashed curve is an example path (from [BPSW05]).

In the free-space diagram algorithm, a free space surface is used, which is the union of all free-space diagrams, as shown in 7.20.

Calculating the result As shown in the previous paragraphs, the decision problem for (weak) Fréchet distance can be solved by finding a path in the free-space from the lower left corner to the top right corner. To find such a path, the algorithm, in [AERW03], is used for the (regular) Fréchet distance. For the weak Fréchet distance, any graph traversal algorithm can be used. Solving the actual minimisation problem is done by applying a binary or parametric search.

Localizing the Fréchet distance algorithm In [BPSW05], an *adaptive clipping algorithm* is proposed, which tries to improve the running time of the above algorithm. This is achieved by using a worst-case motion estimate. To this aim, an error is used, based on the maximum speed the object can move at. It is calculated as shown in Figure 7.21. This error ellipse is a different name for a bead or space-time prism, that is discussed in Chapter 8.

Adaptive clipping The Fréchet distance algorithm could be improved by calculating the active region of the road network, and clipping it to this region. The active region $A(e)$ of an edge e, is defined as the Minkowski sum of the error ellipse with a disk radius of u. However, this would imply that the algorithm is of a global nature.

The adaptive clipping algorithm uses the weak Fréchet distance algorithm to calculate the free-space diagram. Start and end nodes are identified and

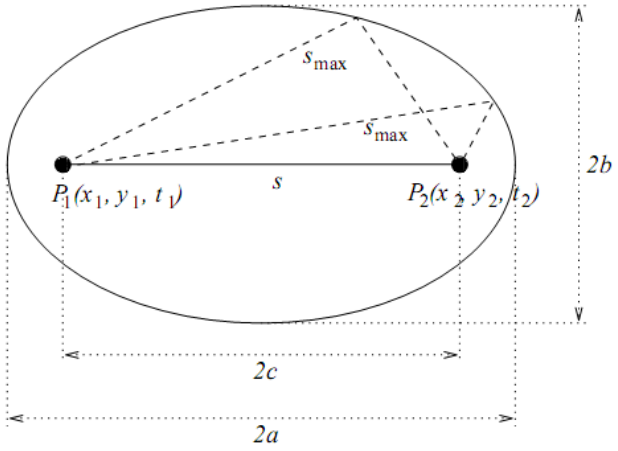


Figure 7.21: The error ellipse. The distance between the two points is $2c$ and the maximum distance the object can travel is $2a$. The value $2b$ is calculated from these two values (from [Gre02]).

then Dijkstra's shortest path algorithm is used. It runs in stages, with each stage corresponding to an edge in the GPS measured trajectory sample. At each stage, only the active region is considered.

7.4.2.6 An algorithm by Yin and Wolfson

In [YW04], Yin and Wolfson use Dijkstra's shortest path algorithm to calculate the path with the smallest weight. Consider a trajectory, which contains a number of time-space points, with a beginning and end point. Given a road map, we calculate the minimum distance between every road segment of this map and the trajectory. We consider the distance of each arc to be the weight of that arc. Now, the lowest weight path between the beginning and end point is taken to be the snapped path.

In order to actually calculate the weight, this algorithm uses a 3D-view weight algorithm. It raises each arc from a two-dimensional polyline to three dimensions by finding the two closest time-space points on the trajectory, and then using linear interpolation between the sample points (see Section 1.2.3). The weight is then calculated using the Euclidean distance between the three dimensional arc and the sub-trajectory between these two sample points.

7.4.3 Probabilistic algorithms

Probabilistic algorithms use a confidence region around each sample point to compute a match point on a road network. This confidence region is based on

the error variances that time-space points typically have.

7.4.3.1 An algorithm by Ochieng, Quddus and Noland

Like many other algorithms, the algorithm by Ochieng, Quddus and Noland [QON03] consists of a initial mapping process, which finds an initial match, and a subsequent mapping process. Like the space-time prisms algorithm discussed in Chapter 8, an area is calculated around each time-space point. The main difference is that this algorithm uses probability concepts to immediately select each road match, whereas the space-time prism algorithm attributes a weight to each segment, and then uses k-shortest paths to find matches. The k-shortest paths algorithm is further explained in Section 8.3.3.

Initial mapping process To find an initial match, a confidence region around the sample point is constructed, as illustrated in Figure 7.22, where

$$a = \sigma_0 \sqrt{\tfrac{1}{2}(\sigma_x^2 + \sigma_y^2 + \sqrt{(\sigma_x^2 - \sigma_y^2)^2 + 4\sigma_{xy}^2}},$$

$$b = \sigma_0 \sqrt{\tfrac{1}{2}(\sigma_x^2 + \sigma_y^2 - \sqrt{(\sigma_x^2 - \sigma_y^2)^2 + 4\sigma_{xy}^2}}, \quad \text{and}$$

$$\phi = \pi/2 - 1/2 \arctan(\tfrac{2\sigma_{xy}}{\sigma_x^2 - \sigma_y^2}).$$

Here, a and b are the lengths of the axes of the of the ellipse and ϕ is the orientation of the ellipse compared to the North. The number σ_0 is the expansion factor. The numbers σ_x^2 and σ_y^2 are the error variances, and σ_{xy} is the covariance. The formulas for a and b are chosen so that the error ellipse scales with the error variance. An example of such an error ellipsis can be found in Figure 7.22.

Road segments, that are in this confidence region, are considered the candidate segments. If there is only one such segment, it is selected. If not, link connectivity, heading information, closeness to the segment in question and historical info are used to select one.

Subsequent mapping process The algorithm considers the object to remain on the same road segment until a junction is reached or a turning manoeuvre is detected. Whenever this is detected, the initial matching process is called again, and a new road segment is chosen. The detection of a turning manoeuvre is based on the speed of the object, the time it takes to do the turn, and the change in heading angle. After matching a time-space point to a road segment, an estimation of where the object is located on the road segment is calculated.

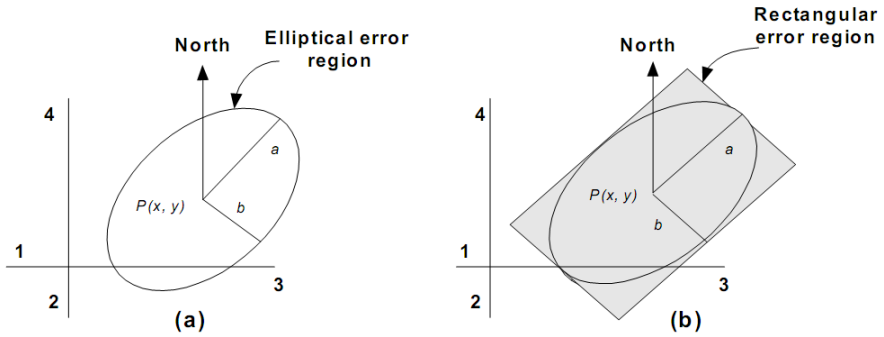


Figure 7.22: A confidence region(from [SC04])

The algorithm described above is an improvement to the algorithm used by Zhao [Zha97]. The main difference is in the subsequent mapping process. The algorithm above only calculates the confidence region when the object moves through a junction or does a manoeuvre, whereas the algorithm of Zhao calculates it when traveling along a road segment as well.

7.4.4 Combined algorithms

7.4.4.1 Fuzzy logic

Fuzzy logic [Zad65] is similar to probabilistic logic in the sense that they both use values between zero and one. Conceptually they are different: fuzzy logic models a degree of truth, while probabilistic logic models likelihood. As an example, consider "temperature." Whether a certain temperature is considered as "warm" or "cold" is subjective, but this can be modelled using fuzzy logic. Membership functions are used to determine the degree of membership of an element in a set. Using the temperature example, $f(25) = 0.8$ would mean that a temperature of 25 degrees has a 0.8 probability to be considered warm. This is illustrated in Figure 7.23.

Algorithms for fuzzy logic are very similar to the weighted topological algorithms. The main difference is that fuzzy definitions are used to calculate a membership degree of a time-space point to a road segment, instead of calculating a weight.

The first map matching algorithm using fuzzy logic was designed by Zhao [Zha97] and it only matches the time-space points to a nearby road network segment. Syed and Cannon [SC04] improve his approach by using more inputs and matching the time-space point to a location on the road segment. Quddus [Qud06] further improves this algorithm by using even more fuzzy inputs and by also using connectivity information and the historical trajectory of the

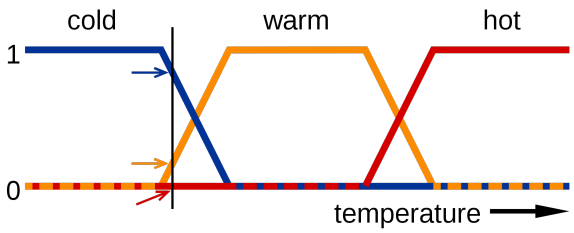


Figure 7.23: An example of fuzzy logic. The blue, orange and red graph are membership functions for "cold", "warm" and "hot", respectively. At the vertical line, the arrows indicate the probability a certain state has. In this example, "cold" has a probability of 0.8, "warm" has a probability of 0.2 "hot" has a probability of 0 (from [Wik]).

object.

This algorithm is very similar to Greenfelds (see Section 7.4.2.3). Only the way the weight is calculated has to be changed to use fuzzy logic.

7.4.4.2 Extended Kalman filter

A *Kalman filter* [Kal60] is a mathematical method with the purpose to use measurements that contain inaccuracies (for example, localisation with GPS-device), to get a result that is closer to the true value. A Kalman filter is recursive in the sense that it uses the value of the previous result in the current calculation. It works by predicting a value, and estimating the uncertainty of this predicted value. A Kalman filter uses a dynamic model of a system; inputs to that model; and known measurements to create an estimate that is better than an estimate that only uses one measurement. In every step of the filter, a weight is used to determine which values are uncertain. These weights are then used to make estimations.

In [QZON03], an *extended Kalman filter* is used to linearise a trajectory. It can be continually updated with sample points if the algorithm is used in real time. The inputs for the filter are derived from the sample points. This particular algorithm also uses "Dead Reckoning" data as input for the filter. Dead Reckoning uses the estimated speed and direction from the last point to calculate the current point. A problem with this approach is that errors are cumulative.

7.4.5 Algorithms for data with low sampling rate

The algorithms described until now require a high sampling rate to provide meaningful results. Not all provided data will have a sufficiently high sampling

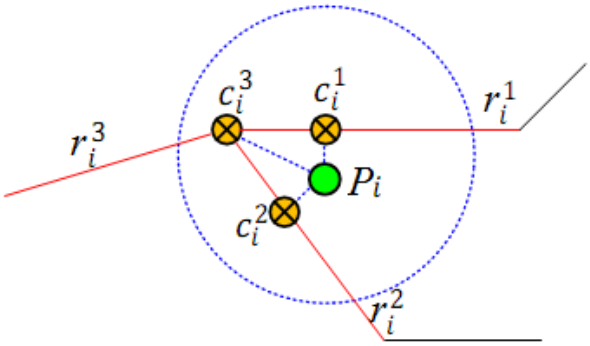


Figure 7.24: The generation of the candidate points (from [LZZ+09]).

rate, hence the need for algorithms that can handle a *low sampling rate*.

7.4.5.1 ST-matching algorithm

In *ST-matching* [LZZ+09], the first step is generating candidate points. Afterwards, the algorithm uses spatial (using distance between a time-space point and road segment) and temporal (using average speed between consecutive points) analysis to give weights to each candidate point. Using these weights, the path with the highest weight is selected.

Generating candidate points For each sample point P_i in the trajectory, candidate points on the road segments within a certain radius of the point are generated. An example of this can be seen in Figure 7.24. The candidate points are chosen so that the distance between the road segment and time-space point is minimal. The j-th road segment for time-space point P_i is denoted as r_i^j and the candidate point as c_i^j.

Spatial Analysis Measurement probability, as defined in Section 7.4.2.1, is also calculated here and instead of modeling noise with a zero-mean Gaussian distribution, a normal distribution is used. Also, the transition probability is calculated in the same as in Section 7.4.2.1.

To represent the likelihood that an object moves from c_i^j to c_{i+1}^k (this is called the *spatial analysis function*), the product of the measurement and transition probability is used. For all consecutive GPS sample points P_i and P_{i+1}, all candidate paths $c_i^j \rightarrow c_{i+1}^k$ are generated, along with their likelihoods that are derived from the spatial analysis function.

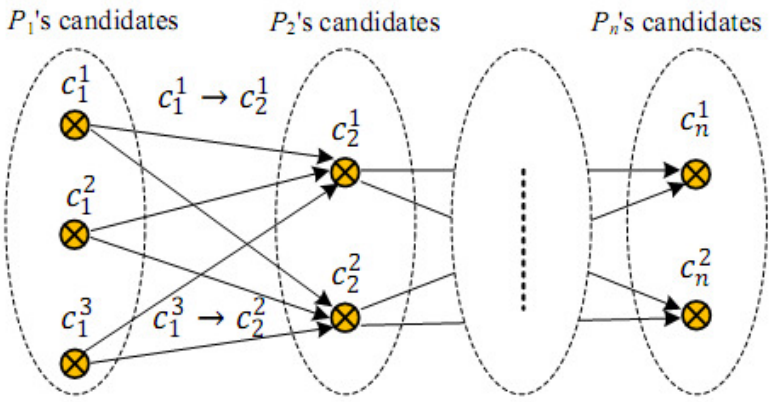


Figure 7.25: The candidate graph $G_T^{'}(V_T^{'}, E_T^{'})$ with $V_T^{'}$ the set of candidate points for each GPS point, and $E_T^{'}$ the set of edges, representing the shortest path between two consecutive candidate points (from [LZZ+09]).

Temporal Analysis Spatial analysis does not take the speed of the object into consideration. In some cases, this information is not enough, and we can use temporal analysis to determine the location of the object. This is achieved by using the average speed of the vehicle, and the maximum speed on the road segments nearby. In this algorithm, the cosine distance is used to measure the similarity between the average speed between two consecutive points, and the speed constraint on the path. We call this the *temporal analysis function.*

Result matching The best matching path S for a trajectory T is selected using the following expression:

$$S = \arg \max_{S_c} F(S_c), \forall S_c \in G_T^{'}(V_T^{'}, E_T^{'}),$$

where S_c is a candidate path sequence and $G_T^{'}(V_T^{'}, E_T^{'})$ as defined in Figure 7.25. The quantity F is the product of the temporal analysis function and the spatial analysis function.

8

An uncertainty-based map matching algorithm

The previous chapter gives a broad overview of existing map matching algorithms. In this chapter, we propose a novel map matching algorithm that exploits the uncertainty caused by a moving object's unknown location between sampled time-space points, by using background information, such as speed limitations. We study the relation between map matching and uncertainty, and propose an algorithm that combines weighted k-shortest paths with space-time prisms.

8.1 Introduction

One particular model for the management of the uncertainty of the moving object's position in between sample points is provided by the *space-time prism* [1] model. In this model, it is assumed that besides the time-stamped locations of the moving object also some background knowledge, in particular a (physically or law imposed) speed limitation v_i at location $(\mathsf{x}_i, \mathsf{y}_i)$ is known. The space-time prism between two consecutive sample points is defined as the collection of time-space points where the moving objects may have passed, given the (local) speed limitation. The chain of space-time prisms connecting consecutive trajectory sample points is called a *space-time prism chain* [Ege03]. We

[1] Sometimes, the term *bead* is used for space-time prism and *lifeline necklace* for space-time prism chain [Oth09].

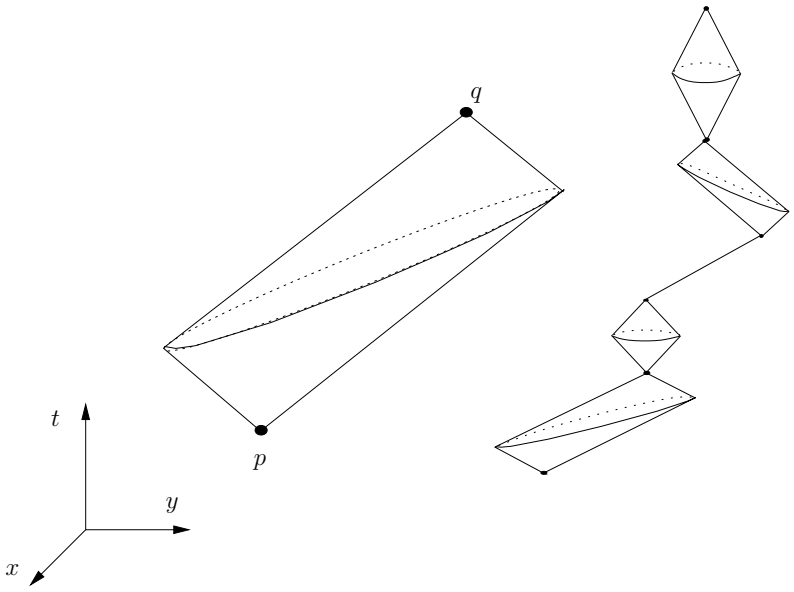


Figure 8.1: A space-time prism and a space-time prism chain (from [Oth09]).

refer to Figure 8.1 for an illustration of a space-time prism and a space-time prism chain in time-space space. Whereas space-time prisms were already conceptually known in the time geography of Hägerstrand in the 1970s [Häg70], they were introduced in the area of GIS by Pfoser [PJ99] and later studied by Egenhofer and Hornsby [HE02, Ege03], and Miller [Mil05].

The main contribution of this chapter is a novel map matching algorithm that uses a combination of techniques for handling uncertainty in trajectory databases. More precisely, we propose to use *space-time prisms* in combination with weighted k-shortest paths algorithms. In the next chapters, we discuss experimental results using this novel algorithm in comparison with existing algorithms (as discussed in the previous chapter). We use two real-world cases. A first one contains very precise information, given at small interval between measurements. Then, we apply the same methodology to a second real-world case study, corresponding to trajectories of cars in the city of Milan, recorded over the course of one week. Here, GPS coordinates are recorded at irregular and less frequent intervals. This implies that data are more imprecise. Another difference between the two cases is that in the first case, moving objects have similar characteristics, while in the second case, we have cars, trucks, buses, among the kinds of vehicles recorded. We compare not only the results over these two cases, but we also compared our algorithm against existing ones. We show that while the geometric algorithm performs rather well in the case of precise data, when data become imprecise (like in the case of

the Milan example) most of the trajectories cannot be reconstructed, while our algorithm achieves a rate of above 90% of success in map-matched trajectory reconstruction.

8.2 Modeling uncertainty with space-time prisms

Often, in practical applications, more is known about measured trajectories than merely some sample points $(\mathsf{t}_i, \mathsf{x}_i, \mathsf{y}_i)$. For instance, background knowledge, like a physically or law imposed speed limitation v_i at location $(\mathsf{x}_i, \mathsf{y}_i)$, might be available. Such a speed limitation might even depend on t_i. For instance, some streets might have different speed limits during the day and the night; or during rush hours. The speed limits, that hold between two consecutive sample points, can be used to model the uncertainty of a moving object's location between sample points. For modeling uncertainty, Pfoser et al. [PJ99], and later Egenhofer et al. [Ege03, HE02], introduced the notion of *beads* (that is, space-time prisms) in the moving object database literature. Before, Wolfson used *cylinders* to model uncertainty [Wol02, GS05]. However, cylinders give less precision (by a factor of 3, compared to space-time prisms).

Let S be a trajectory sample $\{(\mathsf{t}_0, \mathsf{x}_0, \mathsf{y}_0), (\mathsf{t}_1, \mathsf{x}_1, \mathsf{y}_1), ..., (\mathsf{t}_N, \mathsf{x}_N, \mathsf{y}_N)\}$, with $t_0 < t_1 < \cdots < t_N$. Basically, the cylinder approach to managing uncertainty, depends on an uncertainty threshold value $\varepsilon > 0$ and gives a buffer of radius ε around the linear interpolation trajectory $LIT(S)$ of S (see Section 1.2.3). In the space-time prism approach, for each pair $(\mathsf{t}_i, \mathsf{x}_i, \mathsf{y}_i), (\mathsf{t}_{i+1}, \mathsf{x}_{i+1}, \mathsf{y}_{i+1})$, with $0 \leq i < N$, in the sample S, their corresponding space-time prism does not depend on a (universal) uncertainty threshold value $\varepsilon > 0$, but rather on a maximal velocity value v_i of the moving object between those two locations.

We now formalise the concepts above (the description in this section is based on [Oth09], which contains more, detailed information on this topic). We know that, given the speed limitation v_i, at a time t, $\mathsf{t}_i \leq t \leq \mathsf{t}_{i+1}$, the object's distance to $(\mathsf{x}_i, \mathsf{y}_i)$ is at most $\mathsf{v}_i(t - \mathsf{t}_i)$ and its distance to $(\mathsf{x}_{i+1}, \mathsf{y}_{i+1})$ is at most $\mathsf{v}_i(\mathsf{t}_{i+1} - t)$. The spatial location of the object is therefore somewhere in the intersection of the disc with center $(\mathsf{x}_i, \mathsf{y}_i)$ and radius $\mathsf{v}_i(t - \mathsf{t}_i)$ and the disc with center $(\mathsf{x}_{i+1}, \mathsf{y}_{i+1})$ and radius $\mathsf{v}_i(\mathsf{t}_{i+1} - t)$. The geometric location of these points is referred to as a *space-time prism*, and defined as follows, for arbitrary points $p = (t_p, x_p, y_p)$ and $q = (t_q, x_q, y_q)$ and speed limit $v_{\max}$.

Definition 8.1. The *space-time prism* with origin $p = (t_p, x_p, y_p)$, destination $q = (t_q, x_q, y_q)$, for $t_p \leq t_q$, and maximal speed $v_{\max} \geq 0$ is the set of all points $(t, x, y) \in \mathbf{R} \times \mathbf{R}^2$ that satisfy the following constraint formula:

$$\Psi_{\mathcal{P}}(t, x, y, t_p, x_p, y_p, t_q, x_q, y_q, v_{\max}) := (x - x_p)^2 + (y - y_p)^2 \leq (t - t_p)^2 v_{\max}^2 \\ \wedge\ (x - x_q)^2 + (y - y_q)^2 \leq (t_q - t)^2 v_{\max}^2\ \wedge\ t_p \leq t \leq t_q.$$

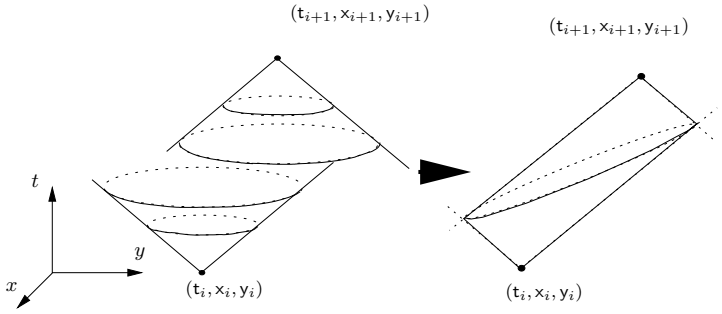


Figure 8.2: An example of a space-time prism $\mathcal{P}(\mathsf{t}_i, \mathsf{x}_i, \mathsf{y}_i, \mathsf{t}_{i+1}, \mathsf{x}_{i+1}, \mathsf{y}_{i+1}, \mathsf{v}_i)$ as the intersection of an upward and a downward cone (from [Oth09]).

We denote this space-time prism by $\mathcal{P}(p, q, v_{\max})$ or $\mathcal{P}(t_p, x_p, y_p, t_q, x_q, y_q, v_{\max})$. □

In the formula $\Psi_{\mathcal{P}}(t, x, y, t_p, x_p, y_p, t_q, x_q, y_q, v_{\max})$, we consider t_p, x_p, y_p, $t_q x_q, y_q, v_{\max}$ to be parameters, whereas t, x, y are considered (free) variables defining a subset of the time-space space $\mathbf{R} \times \mathbf{R}^2$.

Figure 8.2 illustrates the notion of a space-time prism, as the intersection of an upward and a downward cone, in time-space space. A space-time prism can be seen as an envelope which includes all time-space points that the moving object could have visited between the two sample points, given the speed limitation.

For or a trajectory sample $S = \{(\mathsf{t}_0, \mathsf{x}_0, \mathsf{y}_0), (\mathsf{t}_1, \mathsf{x}_1, \mathsf{y}_1), ..., (\mathsf{t}_N, \mathsf{x}_N, \mathsf{y}_N)\}$, the chain of space-time prisms, connecting succeeding trajectory sample points is the set $\bigcup_{i=0}^{N-1} \mathcal{P}(\mathsf{t}_i, \mathsf{x}_i, \mathsf{y}_i, \mathsf{t}_{i+1}, \mathsf{x}_{i+1}, \mathsf{y}_{i+1}, \mathsf{v}_i)$ and is called the *space-time prism chain* of S.

8.3 Using space-time prisms for map matching

In Section 1.2.3, we discussed linear interpolation as a method for reconstructing trajectories from trajectory samples. However, linear interpolation relies on the assumption that between two consecutive points, the object moves at a *constant* (minimal) speed. A more realistic assumption would be that the object moves within a bounded speed limitation, which leads to the *space-time prisms* uncertainty model (Section 8.2). For example, given the time between two consecutive recorded points, and a maximal speed, a moving object (for instance, a car) could have been in many possible locations, given by the projection of the space-time prisms over the plane. The projection of a space-time prisms on the two-dimensional spatial component of time-space space is an *ellipse* with focal points $(\mathsf{x}_i, \mathsf{y}_i)$ and $(\mathsf{x}_{i+1}, \mathsf{y}_{i+1})$ and semi-major

axis $\frac{\mathsf{v}_i(\mathsf{t}_{i+1}-\mathsf{t}_i)}{2}$. However, for simplicity, we compute a *bounding box* of this ellipse (with respect to the x- and y-axes).

The following sections describe how this bounding box of the projection of a space-time prism can be determined and how it can be used in practice, respectively.

8.3.1 Computation of the projection of a space-time prism and its bounding box

First, we define what we mean by the *bounding box* of the ellipse that is the spatial projection of a space-time prism.

Definition 8.2. We define the *bounding box* of an ellipse in the plane, to be the smallest rectangle, with sides parallel to the x- and y-axes, that encloses the ellipse. □

Alternatively, we could say that the bounding box of an ellipse is (the border of) the rectangle that is the Cartesian product of the projection of the ellipse on the x-axis with the projection of the ellipse on the y-axis.

The theorem in this section shows how the bounding box of an ellipse in the plane can be determined, given the focal points (x_1, y_1) and (x_2, y_2) of the ellipse and the semi-major axis $L > 0$ of the ellipse. We remark that the *major axis*, $2L$, expresses twice the longest distance from the centre of the ellipse to a point on the ellipse. Loosely, we will also use the terms major axis and minor axis (see below) to indicate the actual lines that carry these lengths.

In this theorem, we call the x-values of the left and right vertical sides of the bounding box X_1 and X_2 and the y-values of the lower and upper horizontal sides of the bounding box Y_1 and Y_2. So, the bounding box of the ellipse is (the border of) the set

$$[X_1, X_2] \times [Y_1, Y_2].$$

Figure 8.3 gives an illustration of an ellipse and its bounding box.

To contain the length of expressions, we introduce some abbreviations. The *centre* of the ellipse is the point

$$(x_c, y_c) = (\frac{x_1 + x_2}{2}, \frac{y_1 + y_2}{2}).$$

The distance between the foci is abbreviated by d_f, that is

$$d_f = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}.$$

The semi-minor axis of the ellipse is denoted by ℓ. Therefor, 2ℓ is the length of the minor axis of the ellipse.

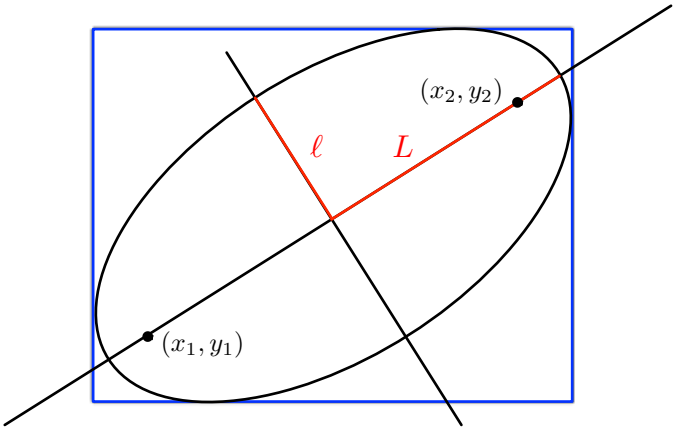


Figure 8.3: An ellipse with focal points (x_1, y_1) and (x_2, y_2), semi-major axis L and semi-minor axis ℓ (in red). Its bounding box is shown in blue.

To determine ℓ, we look at Figure 8.4, where a and b are points on the intersection of the ellipse with its major and minor axis, respectively. If we think of the rope-drawing construction of the ellipse, then we can see that

$$d_f + d((x_1, y_1), a) + d((x_2, y_2), a) = d_f + d((x_1, y_1), b) + d((x_2, y_2), b),$$

since both the left and the right side in this equality equal the length of the rope, needed to draw the ellipse. We also know that $d((x_1, y_1), a) + d((x_2, y_2), a) = 2L$, the length of the major axis.

Furthermore, we know that $d((x_1, y_1), b) = d((x_2, y_2), b) = H$, since we have an equilateral triangle, if we call H the length of the line segment connecting a focal point with b (that is, the length of the hypotenuse of the triangle formed by a focal point, the centre of the ellipse and b). So, we get $d_f + 2L = d_f + 2H$ or $L = H$. Pythagoras' theorem, applied to the triangle formed by the centre of the ellipse, the focal point (x_1, y_1) and the point b, then gives $H^2 = (\frac{d_f}{2})^2 + \ell^2$. Since $L = H$, we obtain

$$\ell = \frac{1}{2}\sqrt{4L^2 - d_f^2}.$$

Finally, if $x_1 \neq x_2$, we abbreviate the slope of the line connecting (x_1, y_1) and (x_2, y_2) by s, that is

$$s = \frac{y_1 - y_2}{x_1 - x_2}.$$

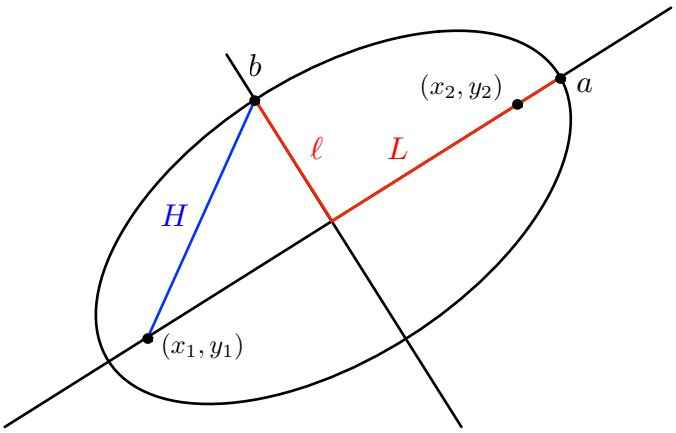


Figure 8.4: An ellipse with focal points (x_1, y_1) and (x_2, y_2), semi-major axis L and semi-minor axis ℓ (in red) and points a and b on the major and minor axis, respectively. The length H of the line segment connecting a focal point with b is indicated in blue.

The goal of the following theorem and its main contribution is to give an explicit description of the bounding box of an ellipse in the plane, given the focal points (x_1, y_1) and (x_2, y_2) of the ellipse and the semi-major axis $L > 0$ of the ellipse. Since the axes of the ellipse are not necessarily parallel to the coordinate axes, we also derive a formula that describes the ellipse (since in textbooks such formulas are generally only given for "standard" ellipses).

Theorem 8.3. *Let (x_1, y_1) and (x_2, y_2) be points in $\mathbf{R}^2$ and let $L > 0$ be a real number. The equation of the ellipse with foci (x_1, y_1) and (x_2, y_2) and semi-major axis L and its bounding box are given by the following expressions:*

- *If $(x_1, y_1) = (x_2, y_2)$, then*

$$(x - x_c)^2 + (y - y_c)^2 = L^2$$

  is the equation of the ellipse and the bounding box is given by $X_1 = x_c - L$, $X_2 = x_c + L$, $Y_1 = y_c - L$ and $Y_2 = y_c + L$;

- *If $x_1 = x_2$ and $y_1 \neq y_2$, then*

$$\frac{(x - x_c)^2}{\ell^2} + \frac{(y - y_c)^2}{L^2} = 1$$

is the equation of the ellipse and the bounding box is given by $X_1 = x_c - \ell$, $X_2 = x_c + \ell$, $Y_1 = y_c - L$ *and* $Y_2 = y_c + L$;

- *If* $x_1 \neq x_2$ *and* $y_1 = y_2$, *then*

$$\frac{(x - x_c)^2}{L^2} + \frac{(y - y_c)^2}{\ell^2} = 1$$

is the equation of the ellipse and the bounding box is given by $X_1 = x_c - L$, $X_2 = x_c + L$, $Y_1 = y_c - \ell$ *and* $Y_2 = y_c + \ell$;

- *If* $x_1 \neq x_2$ *and* $y_1 \neq y_2$, *then*

$$\frac{(y - y_c - s(x - x_c))^2}{\ell^2} + \frac{(s(y - y_c) + (x - x_c))^2}{L^2} = (1 + s^2)$$

is the equation of the ellipse and the bounding box is given by

  - $X_1 = x_c - \sqrt{\frac{s^2\ell^2 + L^2}{1+s^2}}$,
  - $X_2 = x_c + \sqrt{\frac{s^2\ell^2 + L^2}{1+s^2}}$,
  - $Y_1 = y_c - \sqrt{\frac{s^2L^2 + \ell^2}{1+s^2}}$, *and*
  - $Y_2 = y_c + \sqrt{\frac{s^2L^2 + \ell^2}{1+s^2}}$.

Proof. Let (x_1, y_1) and (x_2, y_2) be points in $\mathbf{R}^2$ and let $L > 0$ be a real number. In each of the four cases of the theorem, we first want to find the equation of the ellipse with foci (x_1, y_1) and (x_2, y_2) and semi-major axis L and then determine its projections on the x- and the y-axis.

The first three cases are trivial (high-school geometry). Only the forth case requires some work. We remark that Case 3 coincides with Case 4 for $s = 0$.

Case 1. We assume $(x_1, y_1) = (x_2, y_2)$. In this case the ellipse is a circle with centre $(x_c, y_c) = (x_1, y_1) = (x_2, y_2)$ and radius L. It is given by the equation

$$(x - x_c)^2 + (y - y_c)^2 = L^2.$$

The bounding box of this circle is determined by $X_1, X_2 = x_c \pm L$ and $Y_1, Y_2 = y_c \pm L$.

Case 2. We assume $x_1 = x_2$ and $y_1 \neq y_2$. In this case we have an ellipse where the major axisis in the direction of the y-axis and the minor axis is in the direction of the x-axis. The equation of this ellipse is

$$\frac{(x - x_c)^2}{\ell^2} + \frac{(y - y_c)^2}{L^2} = 1.$$

The bounding box of this circle is determined by $X_1, X_2 = x_c \pm \ell$ and $Y_1, Y_2 = y_c \pm L$.

Case 3. We assume $x_1 \neq x_2$ and $y_1 = y_2$. In this case we have an ellipse where the major axis points in the direction of the x-axis and the short axis points in the direction of the y-axis. The equation of this ellipse is

$$\frac{(x - x_c)^2}{L^2} + \frac{(y - y_c)^2}{\ell^2} = 1.$$

The bounding box of this circle is determined by $X_1, X_2 = x_c \pm L$ and $Y_1, Y_2 = y_c \pm \ell$.

Case 4. We assume $x_1 \neq x_2$ and $y_1 \neq y_2$. So, for the slope s, we have $s \neq 0$.

The line "F" connecting the foci (x_1, y_1) and (x_2, y_2) has equation $F(x, y) = 0$, where

$$F(x, y) = y - y_c - \frac{y_1 - y_2}{x_1 - x_2}(x - x_c) = y - y_c - s(x - x_c).$$

The line "P", perpendicular to F and through (x_c, y_c) has equation $P(x, y) = 0$, where

$$P(x, y) = \frac{y_1 - y_2}{x_1 - x_2}(y - y_c) + (x - x_c) = s(y - y_c) + (x - x_c).$$

The ellipse with foci (x_1, y_1) and (x_2, y_2) and semi-major axis L, then has equation $E(x, y) = 0$, with $E(x, y) = \frac{F(x,y)^2}{A^2} + \frac{P(x,y)^2}{B^2} - 1$ or

$$E(x, y) = \frac{(y - y_c - s(x - x_c))^2}{A^2} + \frac{(s(y - y_c) + (x - x_c))^2}{B^2} - 1,$$

with $A, B > 0$.

We find B by requiring that the two intersection points of the ellipse with the major axis are at distance $2L$ from each other. These intersection points are the solutions of the system of equations $E(x, y) = 0 \wedge F(x, y) = 0$. From $F(x, y) = 0$, we get $y - y_c = s(x - x_c)$. If we use this equality in $E(x, y) = 0$, we get $(1 + s^2)^2(x - x_c)^2 = B^2$, or $x = x_c \pm \frac{B}{1+s^2}$ for the x-coordinates of the two intersection points. The corresponding y-coordinates are $y = y_c \pm \frac{sB}{1+s^2}$. If we

set the distance between the points $(x_c - \frac{B}{1+s^2}, y_c - \frac{sB}{1+s^2})$ and $(x_c + \frac{B}{1+s^2}, y_c + \frac{sB}{1+s^2})$ to $2L$, we obtain $B^2 = L^2(1+s^2)$.

Similarly, we find A by requiring that the solutions of the system $E(x,y) = 0 \wedge P(x,y) = 0$ are 2ℓ apart. If we set the distance between the points $(x_c - \frac{sA}{1+s^2}, y_c + \frac{A}{1+s^2})$ and $(x_c + \frac{sA}{1+s^2}, y_c - \frac{A}{1+s^2})$ to 2ℓ, we obtain $A^2 = \ell^2(1+s^2)$.

$$\begin{cases} A^2 &= \ell^2(1+s^2) \\ B^2 &= L^2(1+s^2), \text{ and} \end{cases}$$

Therefor, the equation of the ellipse with foci (x_1, y_1) and (x_2, y_2) and semi-major axis L is given by the equation $E(x,y) = 0$, where

$$E(x,y) = \frac{(y - y_c - s(x - x_c))^2}{\ell^2} + \frac{(s(y - y_c) + (x - x_c))^2}{L^2} - (1+s^2).$$

To determine the bounding box of this ellipse, we consider the vector

$$\left(\frac{\partial E}{\partial x}, \frac{\partial E}{\partial y}\right),$$

which for a point (x_0, y_0) on the ellipse (that is, for which $E(x_0, y_0) = 0$), gives the direction perpendicular on the ellipse, when evaluated in (x_0, y_0).

When we set $\frac{\partial E}{\partial x} = 0$, this perpendicular is in the direction of the y-axis. The equation $\frac{\partial E}{\partial x} = 0$ is

$$s(y - y_c)(\ell^2 - L^2) + (x - x_c)(s^2L^2 + \ell^2) = 0$$

or

$$x - x_c = \frac{s(L^2 - \ell^2)}{s^2L^2 + \ell^2}(y - y_c),$$

which determines a line which intersects the ellipse in the two points. When we substitute $x - x_c$ from the equation of this line in the equation of the ellipse, we obtain the lower and upper bounds of the bounding box:

$$Y_1 = y_c - \sqrt{\frac{s^2L^2 + \ell^2}{1+s^2}},$$

and

$$Y_2 = y_c + \sqrt{\frac{s^2L^2 + \ell^2}{1+s^2}}.$$

When we set $\frac{\partial E}{\partial y} = 0$, the perpendicular to the ellipse is in the direction of the x-axis. The equation $\frac{\partial E}{\partial y} = 0$ is

$$(y - y_c)(L^2 + s^2\ell^2) + (x - x_c)s(\ell^2 - L^2) = 0$$

or

$$y - y_c = \frac{s(L^2 - \ell^2)}{s^2\ell^2 + L^2}(x - x_c),$$

which determines a line which intersects the ellipse in the two points. When we substitute $y - y_c$ from the equation of this line in the equation of the ellipse, we obtain the left and right bounds of the bounding box:

$$X_1 = x_c - \sqrt{\frac{s^2\ell^2 + L^2}{1 + s^2}},$$

and

$$X_2 = x_c + \sqrt{\frac{s^2\ell^2 + L^2}{1 + s^2}}.$$

This completes the proof. □

8.3.2 Using bounding boxes of the projection of the space-time prisms in map matching

Now, we give two examples of how the bounding box of the projection of the space-time prism can help to limit the number of road segments that have to be considered in the map matching process. But first, we explain how we model *road networks*.

Definition 8.4. A *road network* RN is a graph embedding in $\mathbf{R}^2$ of a labeled (directed) graph given by a finite set of vertices $\mathsf{V} = \{(x_i, y_i) \in \mathbf{R}^2 \mid i = 1, \ldots, N\}$ and a set of edges $\mathsf{E} \subseteq \mathsf{V} \times \mathsf{V}$ that are labeled by a *speed limit*. Vertices are embedded in $\mathbf{R}^2$ by the points that have their coordinates and edges are embedded as straight line segments between the embedded vertices.[2] These straight line segments are called *road segment*. □

Figure 8.5 shows the bounding box of a space-time prism projection computed for two points A and B, that are 13 meters apart. The (projection of the) space-time prism is extremely large in this case, because the traveling distance from A to B is 35 seconds (possibly due to a traffic light stop). The projection of the space-time prism represents the region where the car could have been when it would travel for 35 seconds at a maximum speed of 120 km/h. This results in a huge bounding box of the projection of a space-time prism that contains many streets. On the other hand, Figure 8.6 shows the projection of a space-time prism for two points A and B, which are 11 meters apart, with a travel time of one second. Here, the bounding box of the projection of the space-time prism includes only two road segments.

[2]These edge embeddings may intersect to model bridges and tunnels.

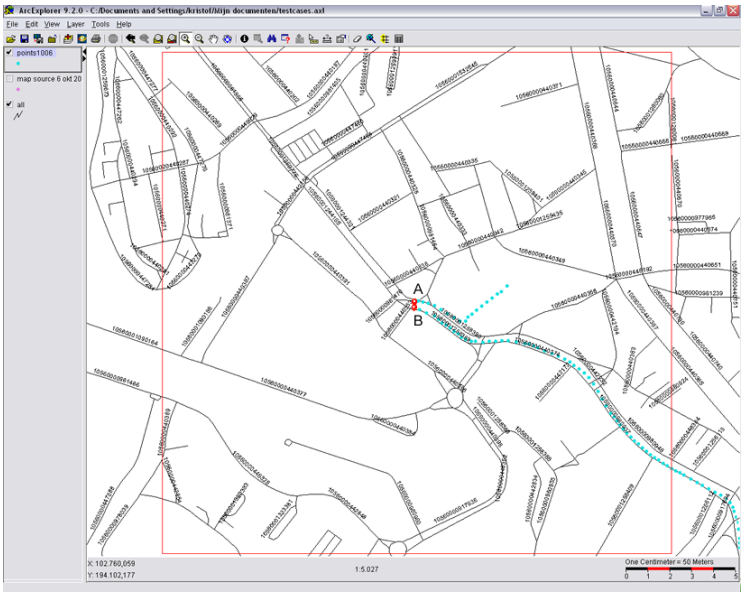


Figure 8.5: An example of the bounding box for two consecutive sample points with a time gap of 35 seconds (from [Bam12])

These examples illustrate how the use of space-time prisms can limit the number of streets or road segments that have to be considered in the map matching process.

We remark that a simple approach to perform the further map matching consists in using a geometric algorithm, as described in Section 7.4.1. In spite of its simplicity, which makes it very (computationally) efficient, this method has some drawbacks, given the characteristics of real-world data discussed in Section 7.2, which, as we see later in our experiments (Chapter 9), sometimes prevent obtaining a matched trajectory (for instance, if there are large gaps in the data). Thus, we need a more involved algorithm to overcome these problems.

8.3.3 An algorithm for k-shortest path routing

Since our new algorithm uses k-shortest path routing, we start with explaining this algorithm. The k-shortest path problem is a well-known problem in networks. It does not only find the shortest path, but also $k-1$ other paths in order of increasing cost. Here, k is the number of shortest paths to found. This problem can be adapted to find the k-shortest paths without loops or with loops. For our purposes, we adapt Yen's algorithm [Yen72] to rank the k-shortest paths *without loops*.

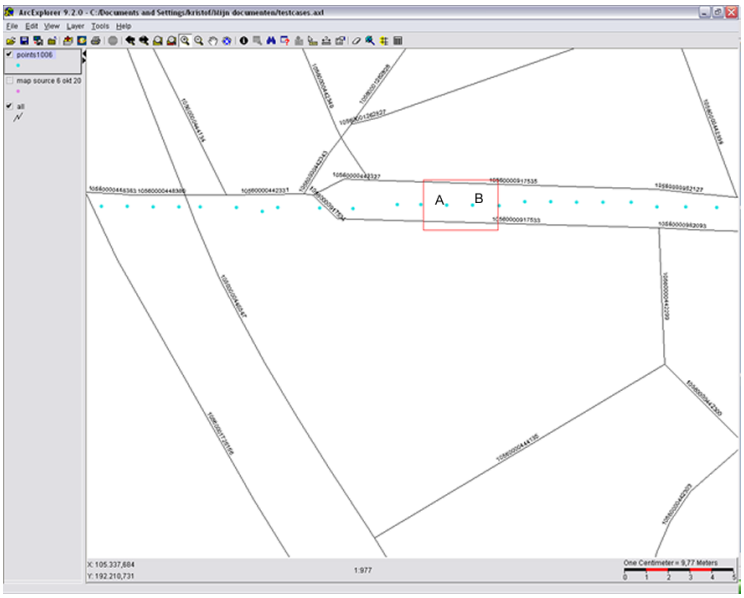


Figure 8.6: An example of the bounding box for two consecutive points with a time gap of 1 second (from [Bam12])

The algorithm described in [Yen72], first computes a shortest path between two vertices using the A^*-algorithm [HNR68]. Then, it takes the n-th vertex in the shortest path, starting with $n = 1$, until $n = k - 1$, and calculates a shortest path from the n-th vertex to the end vertex, called a *spur path*. The path from the start vertex to the n-th vertex is called a *root path*. Two restrictions are placed on a spur path of a vertex:

(1) It must not pass through any vertex on the root path of that vertex (to ensure that the paths are loop-less); and

(2) It must not branch from the current vertex on any edge used by a previously found k-shortest path.

Item (2) means that the spur path cannot start with an edge that is already in a previously found shortest path. For example, if we already have found the shortest paths $A \to B \to C \to D$ and $A \to B \to E \to F \to D$ and we have $A \to B$ as a current root path, then we cannot use the edges $B \to C$ or $B \to E$, because these would result in an already found path.

If a new spur path is found, it is appended to the root path for that vertex to form a complete path from start to end vertex. We illustrate the working of Yen's algorithm [Yen72] in Example 8.5.

The complexity of the algorithm is $O(N^3)$, where N is the number of nodes in the network. Calculating the spur paths from each vertex is $O(N)$ and using

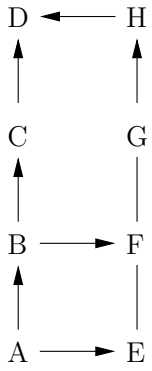


Figure 8.7: A road network for Example 8.5.

Dijkstra's shortest-path algorithm [Dij59] $O(N^2 + |E|)$ with $|E|$ the number of road segment (edges). Also the A^*-algorithm has a $O(N^2)$ upper time complexity bound. Since the time complexity for both Dijkstra's algorithm and the A^*-algorithm is $O(N^2)$ and we have, in the worst-case, to compute it for all N vertices, we get that the complexity of the algorithm is $O(N^3)$.

Example 8.5. We consider the road network of Figure 8.7. We assume that all road segments (arrows) are equally weighted. The problem consists in finding the shortest path from A to D. It is clear that the shortest path is $A \to B \to C \to D$, so we include this path in the result path. Now, we look for other paths starting from the shortest one. We start with root path A, and look for a path from A to D that is not already in the result list. The only possible path, not including the edge from A to B, is $A \to E \to F \to G \to H \to D$. We add this path to the result list. Now, we start with $A \to B$ as the new root path and find $A \to B \to F \to G \to H \to D$. These are all the possible paths and the algorithm ends. □

Why are we using a k-shortest algorithm? There are several special cases of map matching inputs, which are difficult to handle just using a shortest path algorithm. For example, Figure 8.8 depicts a moving object that has followed the road indicated in thin lines (three sample points are shown). Algorithms, based in shortest path algorithms, would likely chose the thick line road (shown in red). We call this problem the *triangle problem*.

In the next section, we discuss how to solve this issue. What we actually do is calculate for each edge a weight. Suppose, for instance, that weights 6, 5 and 6 are given to road segments 1, 2, and 3, respectively. We then calculate for each k-shortest path, for $k = 2$ in this case, its total weight and select the path with the largest weight. In this case, the path follows road segments 1 and 2.

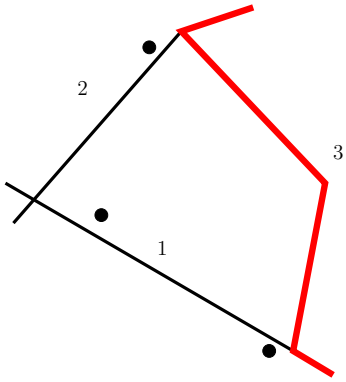


Figure 8.8: A problem with shortest path approaches.

8.3.4 Description of the space-time prisms map matching algorithm

Now, we show how we can use the k-shortest path algorithm, and avoid, for instance, problems like in Figure 8.8, by using the notion of *space-time prisms* (introduced in Section 8.2) and by adding weights to the edges in the road network. Additionally, using space-time prisms allows us to use data sets that contain outliers, since the weight that the related edges receive is negligible. Firstly, the algorithm computes the road segments closest to the recorded space-time sample points, as follows. For each two consecutive time-space points, we compute the bounding box of the projection of their space-time prism, as explained in Section 8.3. Then, we give weights to the road segment. Let m be the number of road segments that we want to give a weight to. We give weights in a way such that the road segment closest to a given space-time sample point gets weight m, the second closest road segment weight $m-1$, continuing until the m-th closest road segment, which receives weight 1. We remark that only the road segments included in the bounded box are taken into account, avoiding including roads that are very unlikely to have been followed. So the number of road segment getting a weight can be less then n.

Then the k-shortest path are calculated, and for each path the total weight is calculated. The path with the highest weight is selected as the map-matched path.

Example 8.6. An example of the working of this algorithm, using as maximum weight 3, can be found in Table 8.1 and Figure 8.9. We describe the working of our algorithm, assuming that the space-time prisms have already been computed (that is, we know which edges are relevant). Starting from point A, we assign a weight to each road segment according to the closeness to this point. Therefore, road segment with $id = 1$ receives a score of 3, and the

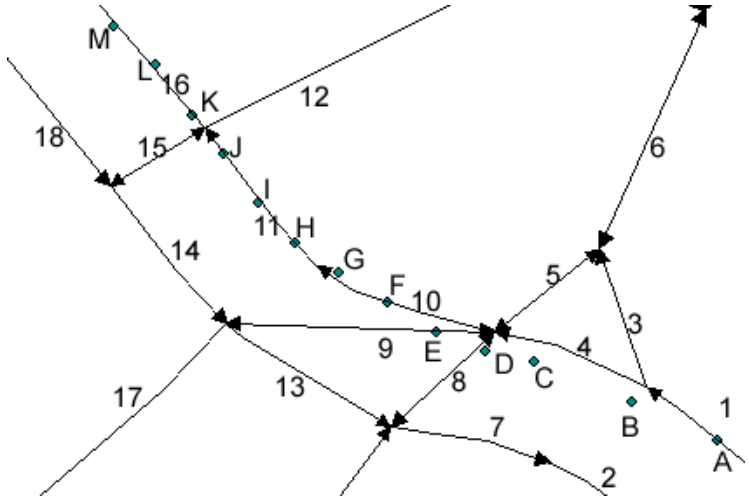


Figure 8.9: The symbols A, ..., M represent sample points and the symbols 1, ..., 18 are identifiers of road segments.

road segments with $id = 3$ and $id = 4$ receive weights 2 and 1, respectively. These weights can be found in the first column of Table 8.1. Thus, A will likely be matched to road segment with $id = 1$. We continue in this way until all points have been analysed and weights assigned. Table 8.1 shows the outcome of the algorithm that gives the weights.

Suppose we calculate 2-shortest paths, then we get route 1 is $1 \rightarrow 4 \rightarrow 10 \rightarrow 11 \rightarrow 16$ and route 2 is $1 \rightarrow 3 \rightarrow 5 \rightarrow 10 \rightarrow 11 \rightarrow 16$] Although route 2 has more edges, its total weight (44) is smaller than that of route 1 (46). We can see that the outcome of our algorithm, route 1, indeed is the actual route taken by the moving object. □

In summary, our map matching algorithm proceeds as follows.

Summary of the algorithm

Step 1. First, the algorithm selects parts of the road network by calculating, for each pair of consecutive points, which road segments the moving object could have driven on (using the bounding boxes of the projections of space-time prisms, as described in Section 8.3).

Step 2. Then, for each sample point, it computes the closest road segment, as described in Section 8.3.4, and assigns scores to each road segment. A score for a segment s is computed by adding up the weights of all the segments that match s.

Table 8.1: An example of the algorithm for the assignment of weights.

Id	Init	A	B	C	D	E	F	G	H	I	J	K	L	M
1	0	3	5	5	5	5	5	5	5	5	5	5	5	5
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	2	3	3	3	3	3	3	3	3	3	3	3	3
4	0	1	4	7	7	7	7	7	7	7	7	7	7	7
5	0	0	0	2	2	2	2	2	2	2	2	2	2	2
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	1	4	5	5	5	5	5	5	5	5	5
9	0	0	0	0	2	5	7	8	9	9	9	9	9	9
10	0	0	0	0	1	3	6	9	11	13	13	13	13	13
11	0	0	0	0	0	0	1	3	6	9	12	12	12	12
12	0	0	0	0	0	0	0	0	0	1	3	4	5	5
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	1	3	5	5
16	0	0	0	0	0	0	0	0	0	0	0	3	6	9
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Step 3. Finally, the algorithm computes, within this limited road network (as determined in Step 1), the k-shortest paths, taking the shortest path with the highest score computed in Step 2.

We end by listing some remarks on this algorithm.

If there are two paths, with the same weight, then the first path found in the k-shortest paths algorithm is selected.

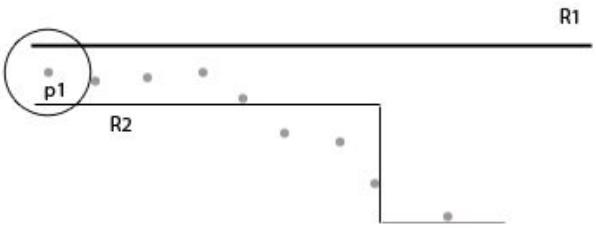


Figure 8.10: An example of an ambiguous trajectory start.

Sometimes, by simply adopting a starting and ending vertex from the road segments closest to the first and last time-space point respectively, may not result in a correct match. This is illustrated in Figure 8.10. Here, the road segment closest to p_1 would be $R1$, although the trajectory clearly follows $R2$.

The ending point will be matched to $R2$, eventually preventing finding a route for this trajectory. To solve this problem, and taking into account that the maximum measurement error is about 10 meters, the algorithm looks at all the possible starting segments (instead of only one) within a circle with a radius of 5 meters around the first time-space point, and selects all road segments that intersect with this circle. The same procedure is followed for the end segment. Then the algorithm looks for possible k-shortest paths between all the possible start segments and end segments within these boundaries (that is, the circles).

In Chapter 9, we will evaluate the above map matching algorithm against the algorithms described is Section 7.4.

9

Experimental evaluation of map matching algorithms

The goal of this chapter is twofold: we propose a novel method to measure the accuracy of a map matching algorithm and we present a number of tests of different map matching algorithms on a variety of trajectory sample data sets.

Several map matching algorithms have been developed for trajectory samples with specific properties. We refer to Chapter 7 for an overview. To accurately compare the performance of algorithms against each other, we need several trajectory samples with different properties and characteristics. In Section 9.1, we give an overview of possible properties of trajectory samples. Ways to obtain these properties in data sets are described in Section 9.2. We also discuss a number of existing methods to measure the accuracy of a map matching algorithm. In Section 9.3.2, we propose a novel accuracy measure that does not suffer from the drawbacks of existing methods.

We conclude this chapter with a number of tests of different map matching algorithms on a variety of trajectory sample data sets. The aim is to figure out which type of map matching algorithm works best on a certain type of trajectory samples. We have implemented a number of existing map matching algorithms and compare these algorithms with our own uncertainty-based map matching algorithm, that we presented in Chapter 8.

9.1 Data properties

We take following properties of trajectory samples into account to compare map matching algorithms:

Sampling rate: The average number of time-space points per second of a trajectory sample. We need to evaluate each map-matching algorithm using trajectory samples with a variety of sampling rates.

Road length: The accuracy of an algorithm may vary for different road segment lengths. For instance, we may consider the average length of highways versus the average length of roads in a city center.

Sampling errors: It is possible for measured trajectory sample points to have an error that is as high as tens of meters. It is also possible that there is noise in the sample (when the distance between two consecutive time-space points is further apart than is physically possible). In order to know how well the algorithm can handle these errors, we consider different test sets.

Size of the route: The total length of a trajectory sample can impact the algorithm accuracy. An incorrectly chosen road segment can cause big errors in, for instance, incremental algorithms.

Road density: The amount of road segments per area of the map (for instance, per square kilometer). When we compare a city center with a highway, this difference can also be observed.

Now that we know the different parameters that describe the properties of trajectory sample data, we next need appropriate testing data.

9.2 Sources of data

In this section, we give an overview of possibilities to generate trajectory samples to test map matching algorithms.

9.2.1 Human labeled data

Using the time-space points measured during a trajectory of a moving person, this person identifies her- or himself the road segment that each time-space point resides on. So, actually this person does a manual map-matching that can be consider as 100% correct. We consider the trajectory created in this way to be the correct one.

9.2.2 Computer generated data

To accurately compare algorithms against each other, we need lots of trajectory samples, especially, since we need trajectory samples with a variety of properties. To generate all this trajectory samples, using the human labeled method would be ideal, but it would take too much time. Therefore, there are methods to automatically generate these trajectory samples.

One way of generating trajectory data is using an existing road network, and randomly selecting 2 nodes in this network. By using a k-shortest path algorithm, k paths are generated. From this collection of paths, one is randomly selected. This path is chosen as a real trajectory produced by a moving object.

The generation of a trajectory sample is done by generating points close to the road segment (within a certain distance). How many points are generated is based on the preferred sampling rate.

In our implementation, we have included an algorithm that generates trajectory samples as described in this section.

9.2.3 Unknown source data

In case the actual trajectory followed for a trajectory sample is not known, it is very hard to produce a "correct" matching. Since, to evaluate one particular map matching algorithm, we need a "correct" matching, one way would be to generate the correct labeling using another map matching algorithm. Since the accuracy of this second algorithm is also not known, the results obtained from comparing the first to the second algorithm may be inaccurate and have limited meaning.

9.3 Methods to measure the quality of a map matching algorithm

There are multiple ways to measure the correctness or accurateness of a map matching algorithm on a given trajectory sample. We list the most important ones below.

Accuracy by length: [LZZ+09] This method uses the total length of the correctly matched road segments. A benefit of this method is that this accuracy measure is directly related to the length of roads correctly matched. Because of this, a long road (or big error) is more important than a small road (or small error). *Accuracy by length* is calculated by the following expression:

$$\frac{\text{total length of correctly matched road segments}}{\text{total length of the matched trajectory}}.$$

Accuracy by number: [LZZ+09] This method uses the number of correctly matched road segments. A downside of this method is that a mismatch of a long road segment is treated the same as the mismatch of a short road segment. This method is beneficial when the amount of correctly matched segments is more important than the correctly matched length. For instance, an algorithm with high accuracy by number will be a good algorithm to apply to a city center. *Accuracy by number* is calculated by the following expression:

$$\frac{\text{number of correctly matched road segments}}{\text{number of road segments in matched trajectory}}.$$

A possible alternative to this would be to generalise the road network (as we did for double-cross descriptions of polylines — see Chapter 4), making each segment roughly the same size.

Accuracy by length minus erroneous road: [WBK00] This method uses the total length of the erroneously added and removed road segments. It has the same benefits as accuracy by length. The main difference is that incorrectly added road segments have a bigger penalty here. It is calculated as shown in Figure 9.1. In this figure, d_- is the total length of erroneously subtracted road segments (from the correct trajectory) and d_+ is the total length of erroneously added road segments (compared to the correct trajectory).

Since this in itself does not give us an idea of how accurate the calculated trajectory is, we combine this with accuracy by length. This gives us the following formula for *accuracy by length minus erroneous road*: we take the value

$$\frac{\text{length of correctly matched road segments} - d_- - d_+}{\text{total length of the matched trajectory}},$$

if this value is positive and we take 0, otherwise.

Weak, average and strong Fréchet distance: [BPSW05] This method uses the Fréschet distance measure(s) between two curves. It is calculated as explained in Section 7.4.2.5. The average Fréchet distance is the average of the weak and strong Fréchet distance. A benefit of using the Fréchet distance method is that we look at the curve itself, instead of at the

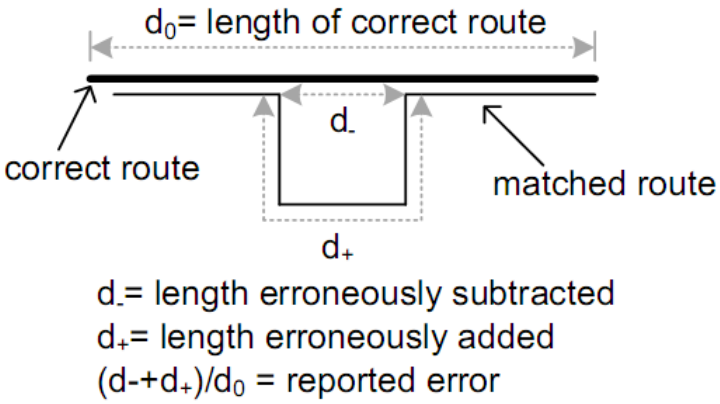


Figure 9.1: Calculation of the error using the length of erroneous road method. (From Bernstein and Kornhauser (2000) [WBK00]).

road segments. A matched trajectory might have a low accuracy with the other measures, but can still be very close to the actual path. Using this distance method allows us to calculate how close these paths are from each other. A downside of this method is that we can get a high accuracy rating for a path that does not have any road segment in common with the original path (for instance, a parallel path).

9.3.1 Flaws in measuring accuracy

The above listed methods to measure the quality of a map matching algorithm might seem to provide an accurate way to measure the correctness of a matched trajectory, but each of them has one or more flaws which may cause them to give an incorrect score in certain cases. In this section, we discuss these flaws.

In Figure 9.2, an example is given. The road network is colored purple and the correct trajectory is colored blue. The selected road segments for the time-space points (the dots) are colored yellow. In the map matching algorithm used to generate the matched trajectory, incorrect road segments are selected. The road segments, that were selected, are parallel to the actual trajectory The *accuracy by length*, *accuracy by number*, as well as the *length of erroneous road* quality measures give no score to segments that are parallel to the correct path, or even penalize those segments. This causes matched trajectories that are parallel to the correct trajectory, but still very close to it, to (incorrectly, in some sense) get a too low accuracy. Quality measures that use the curve distance between the correct trajectory and the matched trajectory (for example, the Fréchet distance) are able to handle this case more adequately, since they do not look at which exact road segment was matched.

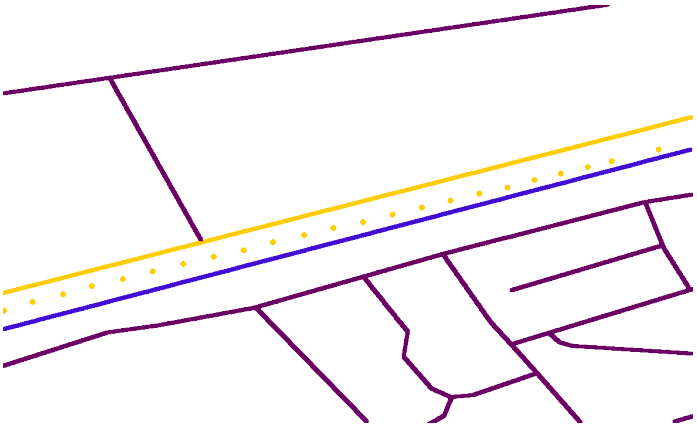


Figure 9.2: A matched trajectory which is parallel to the correct trajectory.

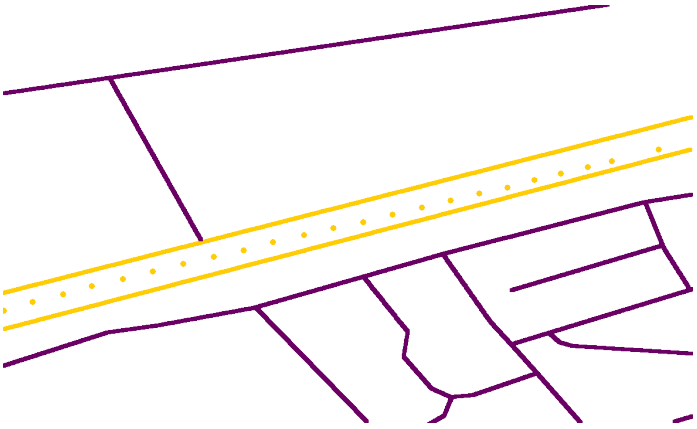


Figure 9.3: A matched trajectory where too many segments are selected.

On the other hand, quality measures using curve distance have a different flaw. This is illustrated by the example shown in Figure 9.3. The road network is colored purple. The selected road segments for the time-space points are colored yellow. The map matching algorithm, used to generate the matched trajectory, selects too many segments. In addition to each correct segment, a parallel segment is incorrectly selected. This causes the matched trajectory to be twice as long as the correct one. Quality measures using curve distance to measure the accuracy would give this trajectory a very good score, since each selected segment is either on the correct curve, or very close to it. However, this is inaccurate, due to the big amount of incorrectly selected road segments.

9.3.2 A new accuracy measure: CL-accuracy

In order to accurately measure the correctness of an algorithm, we propose another quality measure that is able to handle the flaws mentioned in Section 9.3.1. We do this by combining the strengths of the curve distance measure with the strengths of the other measures. This is possible by calculating a curve distance between the correct and the matched trajectory, and afterwards taking the lengths of both trajectories into account. This new method to measure accuracy is named *CL-accuracy*, standing for *Curve-and-Length-accuracy*, as it combines both these measures.

We calculate a score for each selected segment, which is between 0 and 100. This score is calculated by calculating the distance to the closest segment on the correct trajectory. This score is the Euclidean distance (in meters) between these two segments (with a maximum of 100 meters). We take a hard coded limit of 100 meter based on the accuracy of GPS devices nowadays. If a time-space point is more then 100 meter away, it almost certainly is an outlier.

The total score for the matched trajectory is calculated by adding up all the segment scores and then using the following formula:

$$\text{score} = \frac{\text{maxScore} - \text{score}}{\text{maxScore}}.$$

The quantity maxScore in the above formula is the maximum score possible for the matched trajectory (which is 100 times the number of segments). Using this formula, we have calculated a score for the curve distance between the correct and matched trajectory. A score of 100% means each segment of the matched trajectory is on the correct trajectory. A score of 0% means each segment is 100 meters or more away from the correct trajectory. The closer the matched trajectory is to the correct trajectory, the higher the score.

After calculating a score for the curve distance, we still need to take the length of both trajectories into account. We do this by multiplying this score with the length of the smallest trajectory divided by the length of the largest one. If O is the original trajectory and M its map matched version, then O can be longer or shorter than M (depending on the algorithm that is used). We want to obtain a similarity measure between the values 0% and 100%. So, we cannot just use $\frac{|O|}{|M|}$. So, if $|O| > |M|$, we use $\frac{|M|}{|O|}$ instead. As a consequence, if, for instance, M is 10% longer than O, this is penalised in exactly the same way as if O is 10% longer than M.

By doing this, a matched trajectory that is twice as long as the correct trajectory only gets half the score.

9.4 Overview of the experimental evaluation of map matching algorithms

In the remainder of this chapter, we present a number of tests of different map matching algorithms on a variety of trajectory sample data sets. The aim is to figure out which type of map matching algorithm works best on a certain type of trajectory samples. We have implemented a number of existing map matching algorithms and compare these algorithms with our own uncertainty-based map matching algorithm, that we presented in Chapter 8.

We have selected algorithms from each category discussed in Section 7.4, in order to compare our own approach to representatives of each category.

The algorithms that we selected for the *geometric analysis* category are

- the curve-to-point algorithm; and
- the curve-to-curve algorithm,

but we have made a few changes in them. In the geometric analysis algorithms we have implemented, we use Dijkstra's algorithm when two consecutive matched segments are not connected. This ensures there are no gaps in the calculated trajectory.

We have selected the

- Greenfeld algorithm

for *topological analysis* category, since this was one of the first algorithms of its kind, and many other algorithms resemble this method (using an initial mapping and sequential mapping subroutine).

For the *probabilistic* category, we have implemented

- the algorithm by Ochieng, Quddus and Noland.

For the *low sampling rate* category, we have implemented

- ST-matching algorithm.

Off course, we have also implemented our

- space-time prisms combined with k-shortest-paths algorithm,

as described in Chapter 8. In the experiments we use as maximum weight $m = 50$.

In Section 9.5, we show test of these algorithms on trajectory samples generated by a GPS-equipped device. After this, in Section 9.6, we show test of these algorithms on computer generated trajectory samples. Finally, in Section 9.7, we combine all these results and formulate a conclusion on the discussed map matching algorithms. All tests in this chapter are run on a Macbook with 2.16 GHz Intel Core 2 Duo processor and 1 GB RAM.

9.5 Tests on human labeled data

For the tests on human labeled data, we work with trajectory samples generated by a GPS-equipped device, that are manually labeled (that is, consistent with the sample). In these data, timestamps, speed and heading information were included.

9.5.1 Data from the police force of Ghent

The data used for this test are trajectory samples originating from the police force of Ghent. The trajectory samples were located in the city it self and collected by intervention cars of the police. In Figure 9.4 and Figure 9.5, two trajectory samples of this data set are shown.

Figure 9.4: The time-space points of trajectory sample 1 in blue and the human labeled trajectory in red.

Both samples contains around 400 time-space points and are about 4 km long. They also contain a couple data gaps, which are usually caused by driving through tunnels (where GPS reception is absent). The results on these two test samples are shown in Figure 9.6 and Figure 9.7. In Figure 9.8, we show the average result on twenty trajectory samples from the police force of Gent. An overview of average running times in seconds is given in Table 9.1.

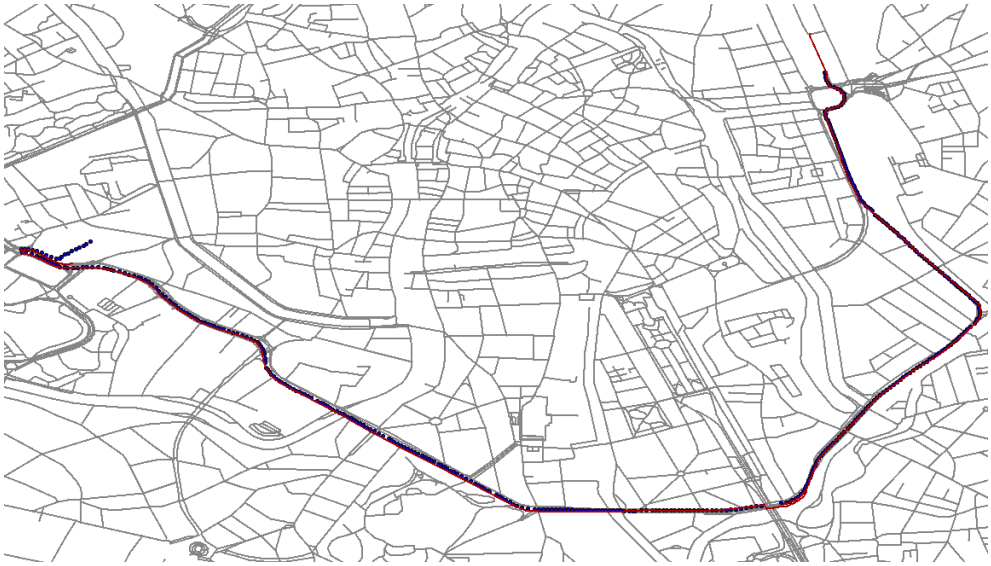


Figure 9.5: The time-space points of trajectory sample 2 in blue and the human labeled trajectory in red.

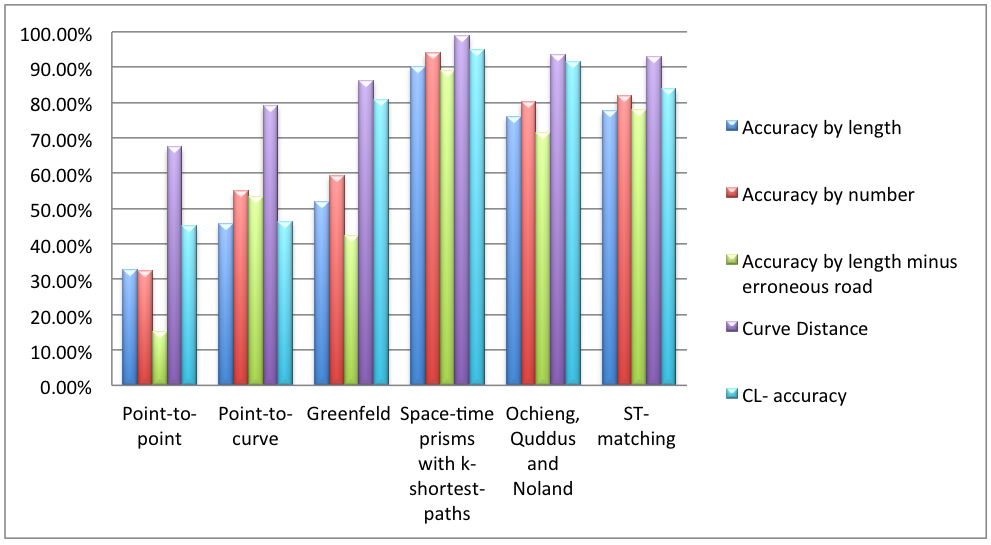


Figure 9.6: Results of the map matching algorithms on trajectory sample 1 of the Ghent police data set.

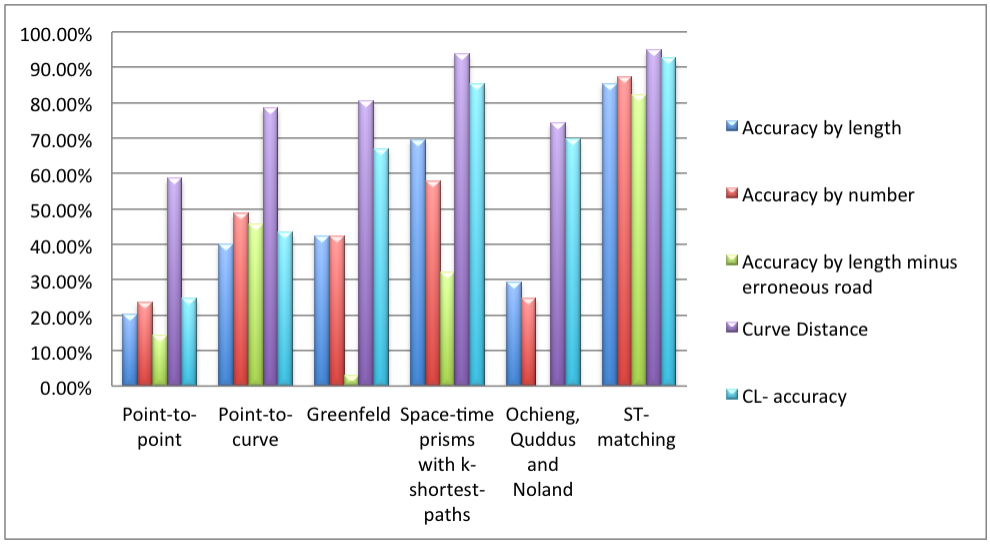


Figure 9.7: Results of the map matching algorithms on trajectory sample 2 of the Ghent police data set.

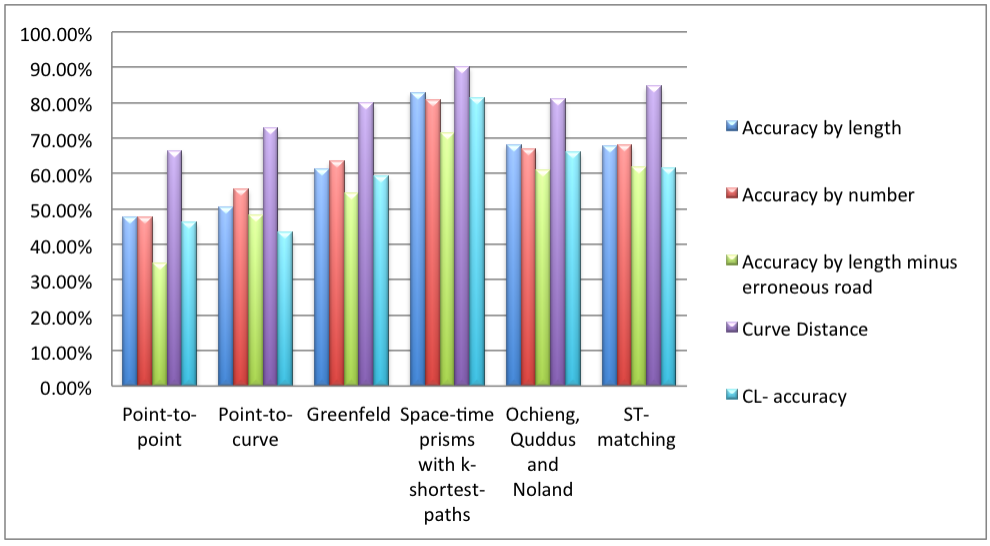


Figure 9.8: Average results of the map matching algorithms on 20 trajectory samples of the Ghent police data set.

Algorithm	Sample 1	Sample 2	Average on 20 Samples
Point-to- point	12,941	14,827	7,7725
Point-to- curve	10,174	10,828	6,99015
Greenfeld	1,544	1,454	1,68845
Space-time prisms with k-shortest- paths	9,890	9,454	6,015
Ochieng, Quddus and Noland	3,861	6,789	7,21515
ST- matching	126,005	220,798	76,12535

Table 9.1: Average running time in seconds of the map matching algorithms on the three test cases from the date set of the police force of Ghent.

9.5.2 Conclusions for human labeled data

The measures accuracy by length, accuracy by number and length of erroneous road, quality measures that do not take into account if a parallel road is being chosen, proved to not always be a good choice to show how accurate a trajectory sample is. For example, Greenfeld's algorithm on Sample 2 (see Figure 9.7) has low scores for these quality measures, while being relatively close to the correct trajectory, as shown by the curve distance and CL-accuracy. On the other side, the curve distance measure in Figure 9.6, gives a high score on the point-to-point and point-to-curve algorithms which choose too many segments, resulting in a trajectory that has a close curve distance to the actual trajectory, but has bad scores in every other measure. At a first glance, geometric analysis algorithms performs the worst of all algorithms. This makes sense, since they do not look at anything else but the location of the time-space point. Information about where previous points are matched, or how the road network is connected, are all ignored.

The other types of algorithms all seem to perform decently in the average case scenario. In the average case, our space-time prisms combined with k-shortest-paths algorithm appears to perform best. In Section 9.6, we discuss tests of these algorithms for many different test cases to figure out which algorithm works best in which scenario.

The running times of each of the tested algorithms lie closely together. All of these algorithms are able to be run in real-time on a route planner. The exception to this is the ST-matching algorithm, which cannot be run in real-time if the sampling rate is too high. This is due to the fact that the ST-matching algorithm is specifically designed to run on data with a low sampling rate. Something that might seem counter-intuitive is that the geometric analysis algorithms perform slower than most of the other algorithms. This is due to a

modification we have done in these algorithms. In our implementation of the geometric analysis algorithms, we use Dijkstra's shortest path algorithm when two consecutive matched segments are not connected. This ensures there are no gaps in the calculated trajectory. Without this change, these algorithms would run substantially faster than they do now, but still give worse results.

9.6 Tests on computer generated data

In this section, we use data that are generated by our trajectory sample generator. Using this generator, we are able to create trajectory samples that have one or more of the properties discussed in Section 9.1 (for instance, data with a low sample rate). After running our implemented algorithms on these data, we are able to determine which algorithm suits which situation best.

9.6.1 High sampling rate

In this section, we will focus on simulated trajectory samples having a high sampling rate. This means that the time difference between two consecutive time-space points is less then 10 seconds.

9.6.1.1 Simulated trajectory samples with no recording errors

For this test, we created trajectory samples with time-space points being recorded every couple of seconds. The trajectory samples have lengths of a few kilometers. In Figure 9.9, the results are shown for one trajectory that contains 658 time-space points.

Figure 9.10 shows the average result on ten computer-generated trajectories.

As expected, none of the algorithms had any problem detecting the correct path. The point-to-point algorithm selects too many segments, resulting in a less than optimal score. Besides this algorithm, any other algorithm can be said to be accurate enough to be used with this type of data.

9.6.1.2 Simulated trajectory samples with recording errors

For this test, we created trajectory samples with time-space points being recorded with an interval between 3 to 10 seconds, with an error between 0 and 15 meters for each point. In Figure 9.11, the results are shown for one trajectory that contained 631 time-space points, with an interval of 3 seconds and an error between 0 to 10 meters.

Figure 9.12 shows the average test result on ten trajectories.

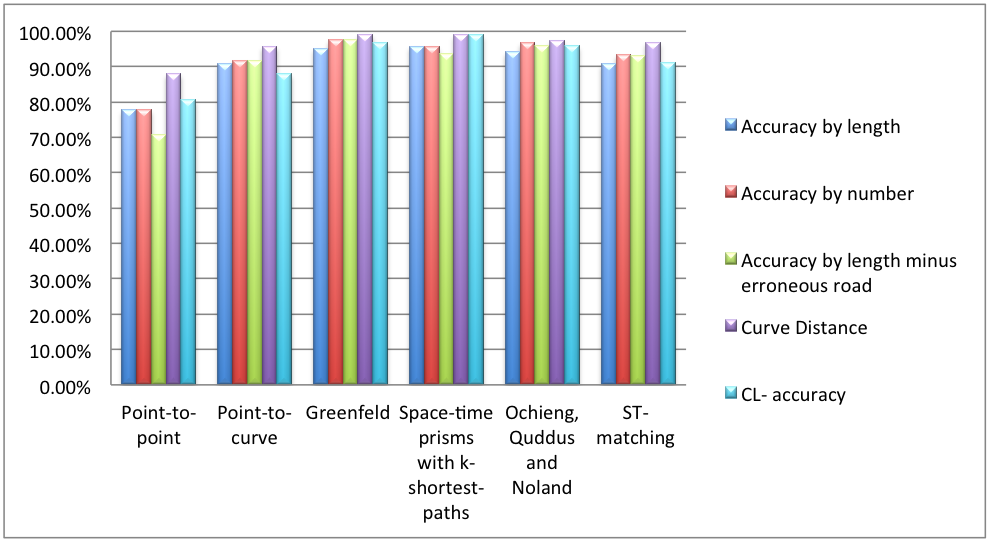


Figure 9.9: Results of the map matching algorithms on one computer generated trajectory sample with no measurement errors.

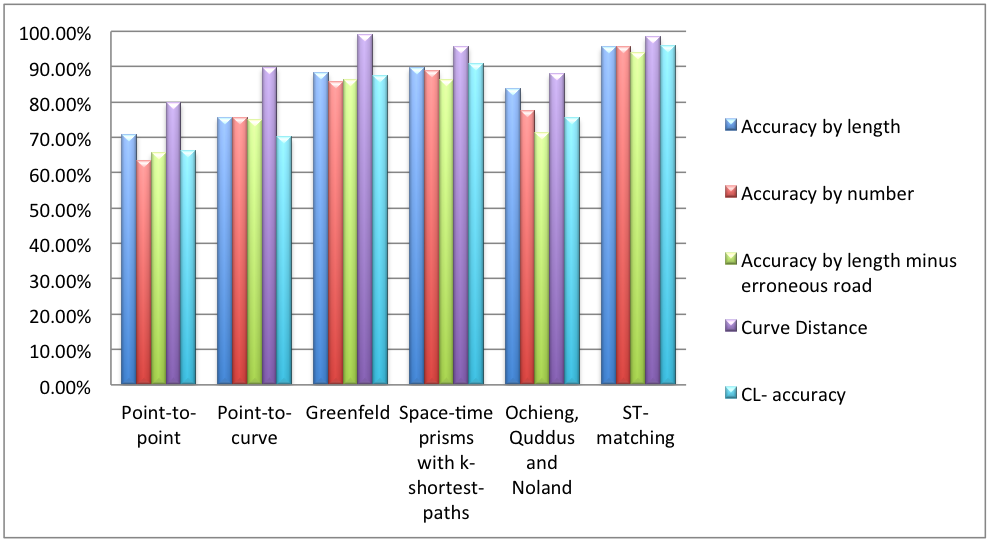


Figure 9.10: Average results of the map matching algorithms on ten computer generated trajectory samples with no measurement errors.

The algorithms using geometric analysis perform poorly, in this case. They generate a lot of false positives, resulting in a poor score. The other types of algorithms perform about equally on this type of data as on data without errors, and generally have no problem handling these errors.

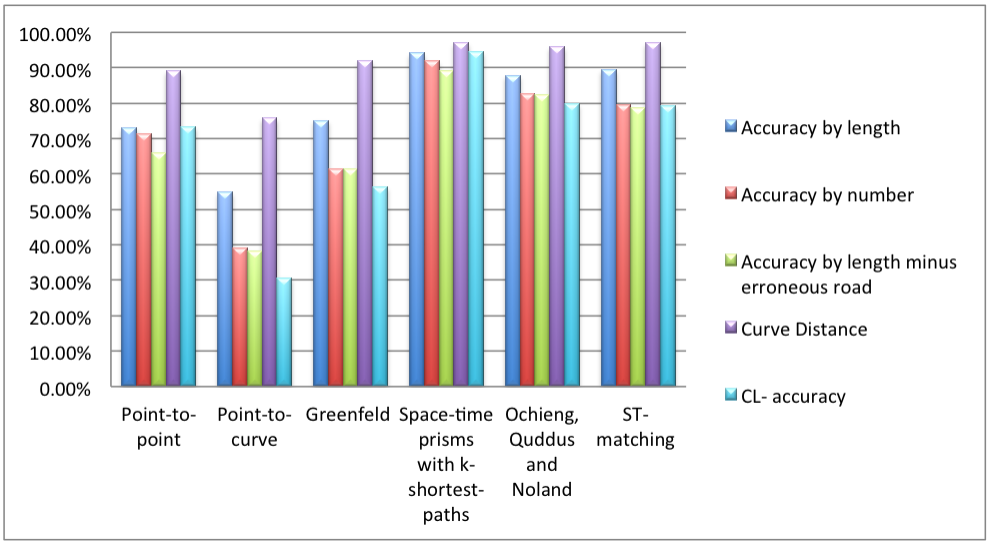


Figure 9.11: Results of the map matching algorithms on one computer generated trajectory sample with measurement errors.

9.6.1.3 Simulated trajectory samples with recording errors and outliers

Data collected with a GPS-equipped device can not only contain measurement errors, but also *outliers*. In Figure 9.13, the results are shown for a trajectory sample containing 520 measurement, measured every 3 seconds. The measurement error of each point varies between 0 and 10 meters. In this trajectory sample, two outliers were created, 15 and 100 meters removed from any other point in the trajectory sample.

Figure 9.14 shows the average result on 10 trajectory samples with a high sampling rate (3-10 secs between time-space points), with measurement errors (between 0 and 15 meters per point) and with outliers (1-3 outlier per dataset, ranging from 10-250 meters away from the trajectory).

The geometric algorithms are not able to handle the outliers very well. This is caused by calculating the shortest path between each two succeeding points and adding this route to the calculated trajectory. Due to this, a lot of road segments leading to the outliers are incorrectly added to the calculation,

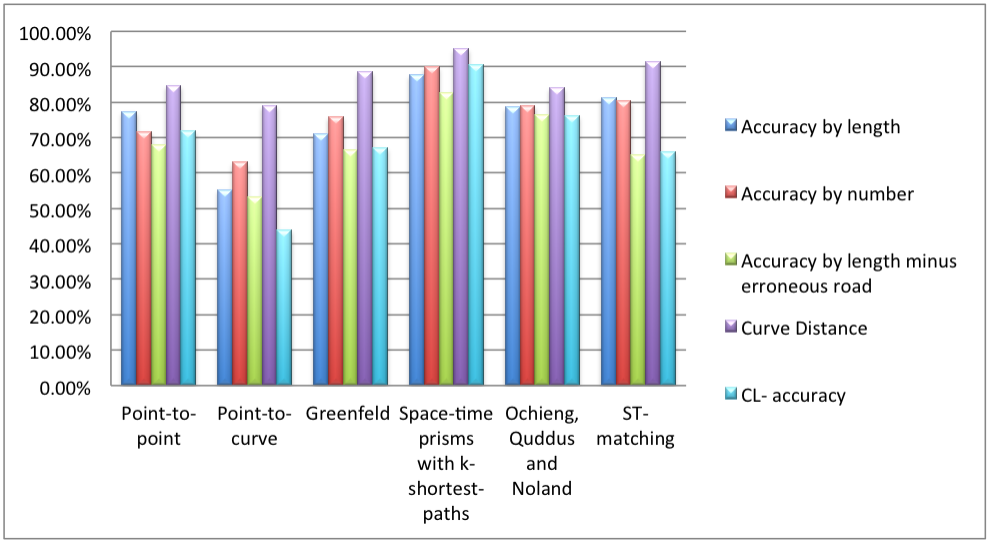


Figure 9.12: Average results of the map matching algorithms on ten computer generated trajectory samples with measurement errors.

resulting in a poor score. The ST-matching algorithm has a similar method of calculating the trajectory, and also has a poor score caused by this. The other algorithms have no problems with the outliers, and are able to calculate the trajectories nearly as well as without outliers. The space-time prisms with k-shortest-paths algorithm performs very well in this case, and gets a perfect score in the above test.

9.6.1.4 Simulated trajectory samples with measurement errors and gaps

When passing through tunnels, there is no connection with GPS satellites to calculate a moving object's location. This causes gaps in the trajectory samples. To test how well map matching algorithms are able to handle these gaps, we have generated trajectory samples with gaps, to simulate passing through tunnels.

Figure 9.15 shows the result on a trajectory sample containing 535 points, measurement errors between 0 and 10 meters per point and 15 consecutive missing points. We also generated 10 other trajectory samples with a high sampling rate (3-10 secs between points), measurement errors (between 0 and 15 meters per point) and with gaps (1-3 gaps per sample, ranging from 50-200 meters in size). The average result of the map matching algorithm on those samples is shown in Figure 9.16.

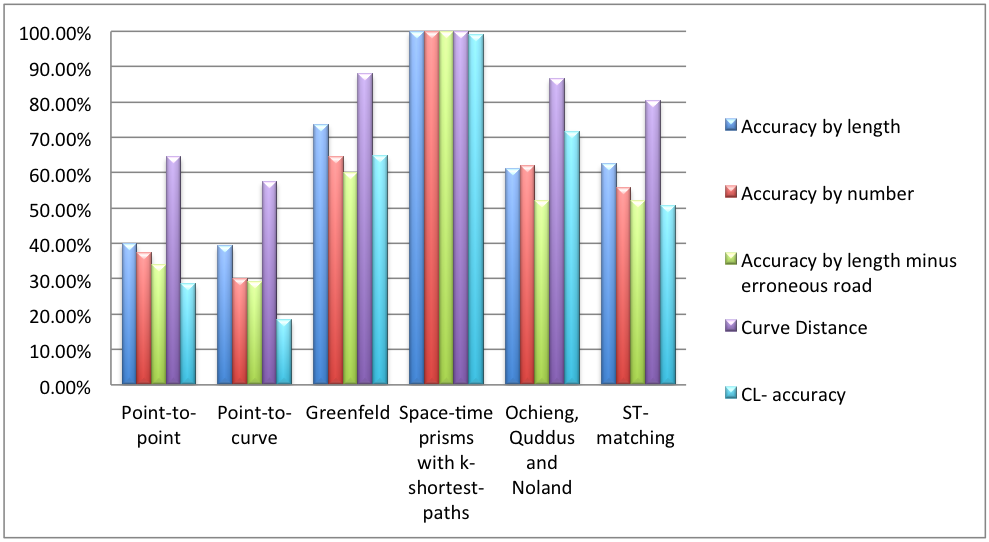


Figure 9.13: Results of the map matching algorithms on one computer generated trajectory sample with measurement errors and 2 outliers.

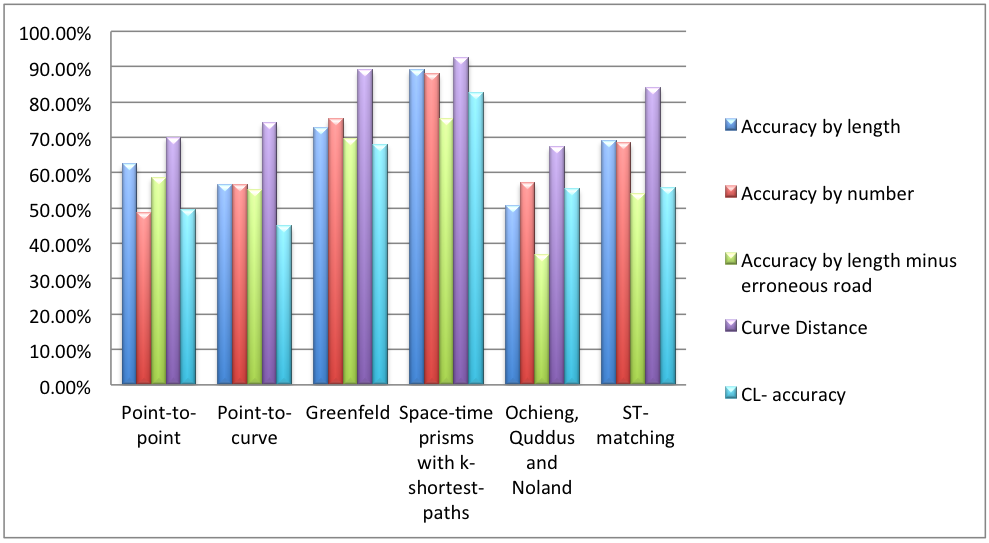


Figure 9.14: Average results of the map matching algorithms on 10 computer generated trajectory samples with measurement errors and outliers.

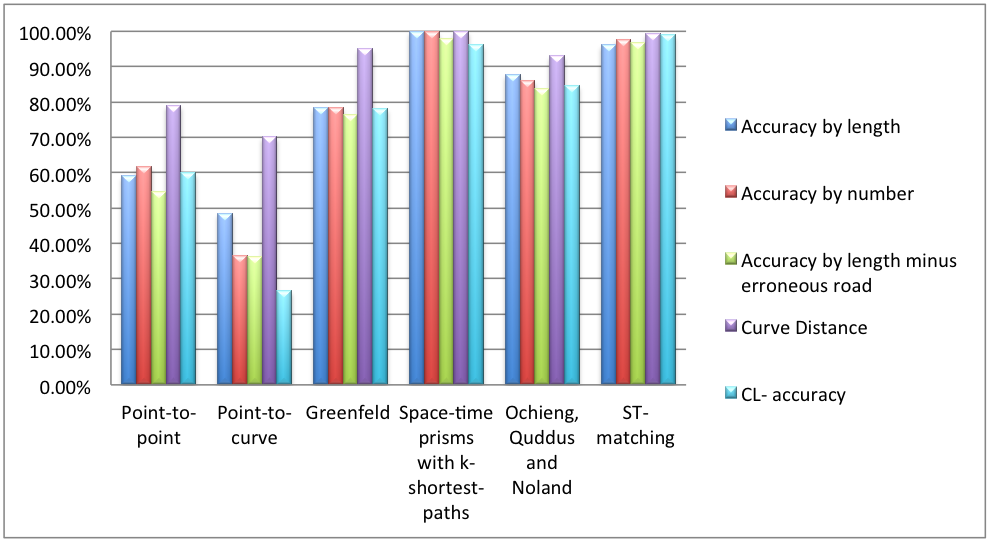


Figure 9.15: Results of the map matching algorithms on one computer generated trajectory sample with measurement errors and a gap.

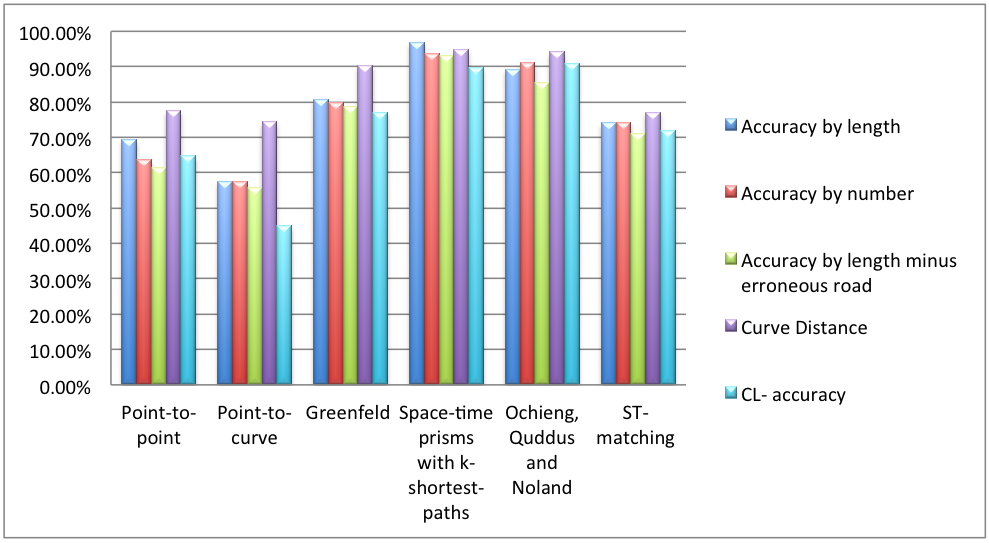


Figure 9.16: Average results of the map matching algorithms on ten computer generated trajectory samples with measurement errors and gaps.

Data containing gaps did not cause a problem to any of the algorithms. The space-time prism with k-shortest-paths algorithm performed very well in this case. The ST-matching algorithm also did very well in most cases, but was unable to calculate a trajectory for two of the used data sets. The algorithm by Ochieng, Quddus and Noland was also able to handle this type of data very well.

9.6.1.5 Simulated trajectory samples on a highway with GPS errors

For this test, we have simulated trajectory samples on a highway, instead of near or in a town. On a highway, there are less possible road segments to choose from. In Figure 9.17, the results are shown for a trajectory sample of about 50 km long, containing 786 time-space points and a measurement error between 0 and 10 meters. In Figure 9.18, we show the average results on ten simulates trajectory samples with a high sampling rate (3-10 secs between GPS points), with measurements errors (between 0 and 15 meters per point).

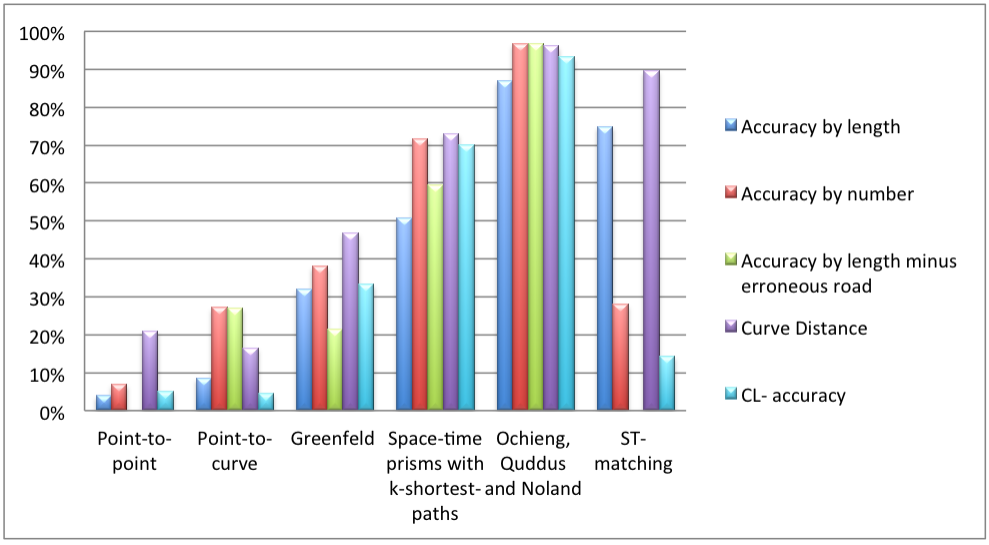


Figure 9.17: Results of the map matching algorithms on one computer generated trajectory simulating driving on a highway with measurement errors.

The poor score of most of the tested algorithms can be explained by the algorithms often choosing parallel roads, combined with non-highway roads near the highway. The actual trajectory or a parallel trajectory was selected in each case, but due to selecting a lot of incorrect road segments, they still got a poor score. The exceptions to this are the space-time prism with k-

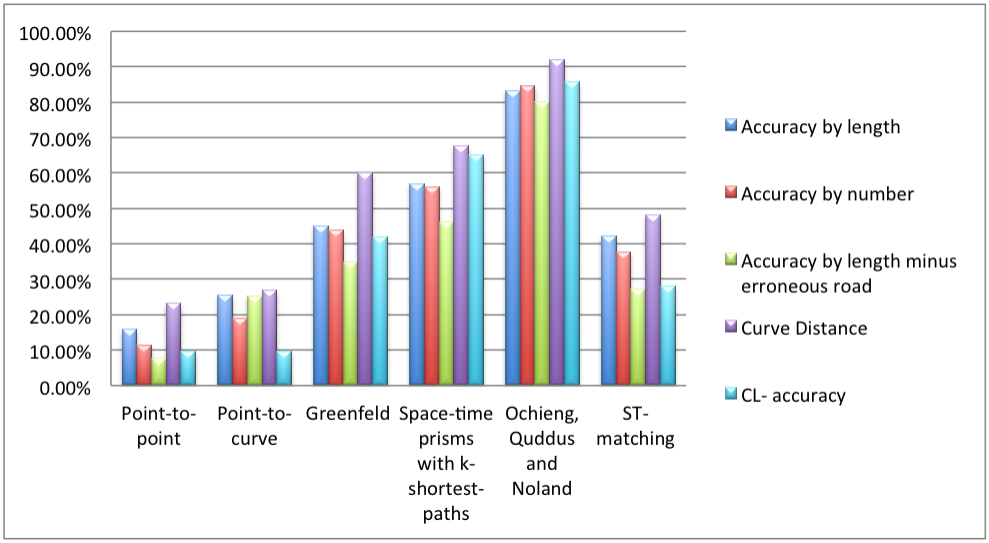


Figure 9.18: Average results of the map matching algorithms on ten computer generated trajectories simulating driving on a highway with measurement errors.

shortest-paths algorithm, which still worked fairly well, and the probabilistic algorithm, which was able to handle data on highways very well.

9.6.1.6 Simulated trajectory samples on a long distance with measurement errors

The data sets used in this section until now, only contained medium sized trajectory samples. But since some of the implemented algorithms are incremental, errors made in the beginning of the algorithm can cause a big difference in the following calculations. Therefore, in this section, we use trajectory samples containing several thousand of time-space points. The results on one trajectory sample, and the average result on ten trajectory samples with a high sampling rate (3-10 secs between time-space points), with measurement errors (between 0 and 15 meters per point) and several times more time-space points than other tested trajectories, are shown respectively shown in Figure 9.19 and Figure 9.20.

We remark that the poor score of the probabilistic algorithms by Ochieng, Quddus and Noland, as shown in Figure 9.19, is caused by calculating the score as a product of the distance score W_D with azimuth score W_{AZ}[1]. This results in the algorithm getting stuck on selecting one specific road segment, since the

[1] see Section 7.4.2.3 for details on these scores

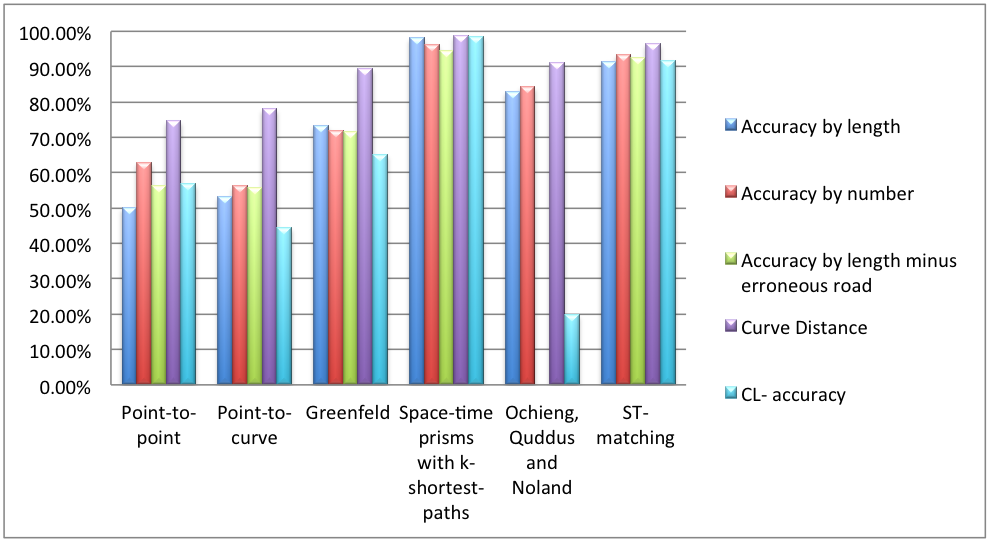


Figure 9.19: Results of the map matching algorithms on one long computer generated trajectory with measurement errors.

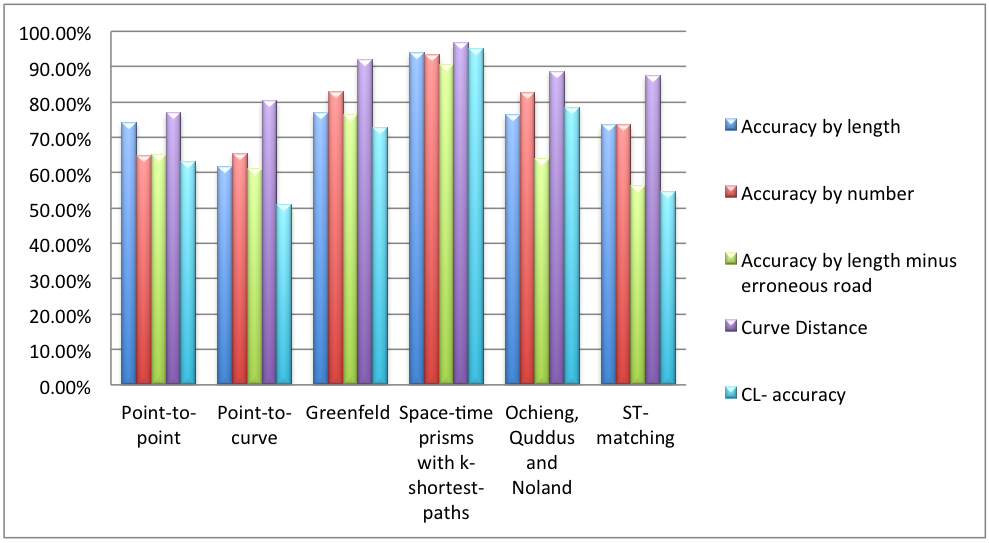


Figure 9.20: Average results of the map matching algorithms on ten long computer generated trajectories with measurement errors.

alternative has a very poor score and is never selected. Due to this problem, this algorithm does not progress further than this road segment. This causes a bad score. In the other tests, no such problems were encountered.

The other algorithms do not encounter any additional problems with trajectory samples of long length. They obtain scores similar to the ones for a shorter trajectory.

9.6.2 Low sampling rate

In the previous section, we discussed trajectory samples where the time difference between two consecutive time-space points is less than 10 seconds. In the following sections, we investigate what happens when the time difference is much longer.

9.6.2.1 Simulated trajectory samples with no measurement errors

The trajectory samples used in this test follow the same trajectory as the ones used in Section 9.6.1.1. In Figure 9.21, the results are shown for a trajectory sample where time-space points were only generated every 60 seconds, instead of every 3 seconds as in Figure 9.9. Further, we generated ten trajectories with a low sampling rate (60-150 secs between consecutive time-space point) and with no measurement errors. The results are shown in Figure 9.22.

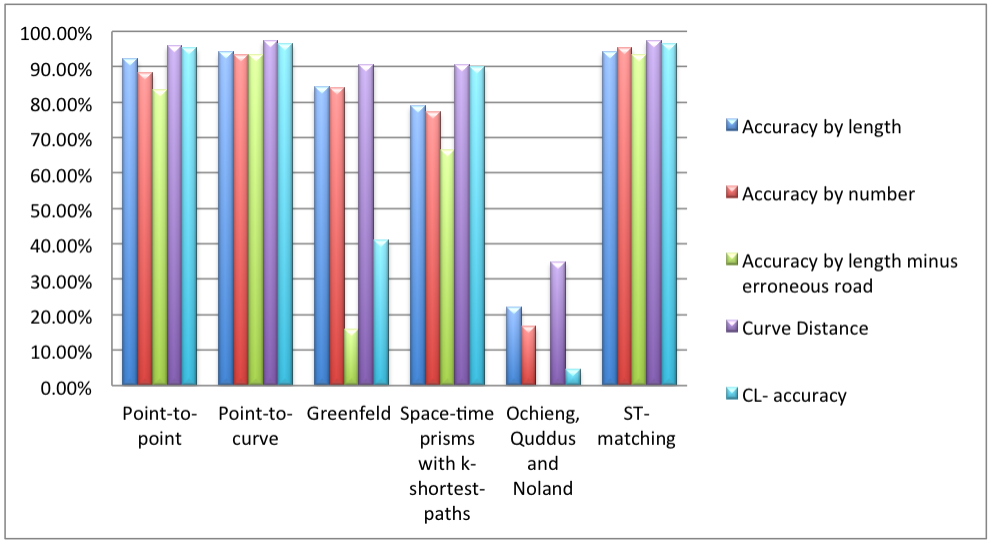


Figure 9.21: Results of the map matching algorithms on one computer generated trajectory with low sampling rate and no measurement errors.

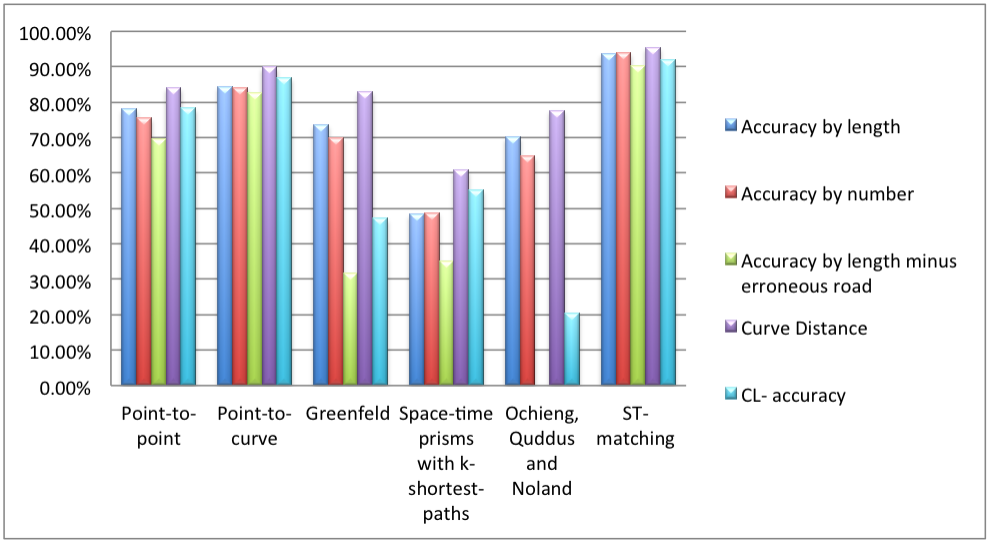


Figure 9.22: Average results of the map matching algorithms on ten computer generated trajectories with low sampling rate and no measurement errors.

As expected, the algorithm specifically designed for map matching data with low sampling rates score the best on this type of data. The geometric analysis algorithms, as well as the space-time prism with k-shortest-paths algorithm are able to handle data with low sampling rate as well in most cases. However, when the sampling rate drops too low, the space-time prism with k-shortest-paths algorithm fails to generate a path. This happens twice in our tests.

Greenfeld's algorithm and the probabilistic algorithm performed poorly on data with a low sampling rate. Since these algorithms only pick a maximum of one road segment per time-space point, not enough road segments are selected, causing poor result.

9.6.2.2 Simulated trajectory samples with measurement errors

The trajectory samples used in this test have the same trajectory as in Section 9.6.1.2. Figure 9.23 shows the results when time-space points are only generated every 60 seconds, instead of every 3 seconds as in Figure 9.11, and with measurement errors between 0 and 15 meters per point. To obtain Figure 9.24, we generate ten trajectory samples with a low sampling rate (60-150 secs between time-space points) and with measurement errors (between 0 and 15 meters per point).

Introducing errors in the data does not affect the results much. The

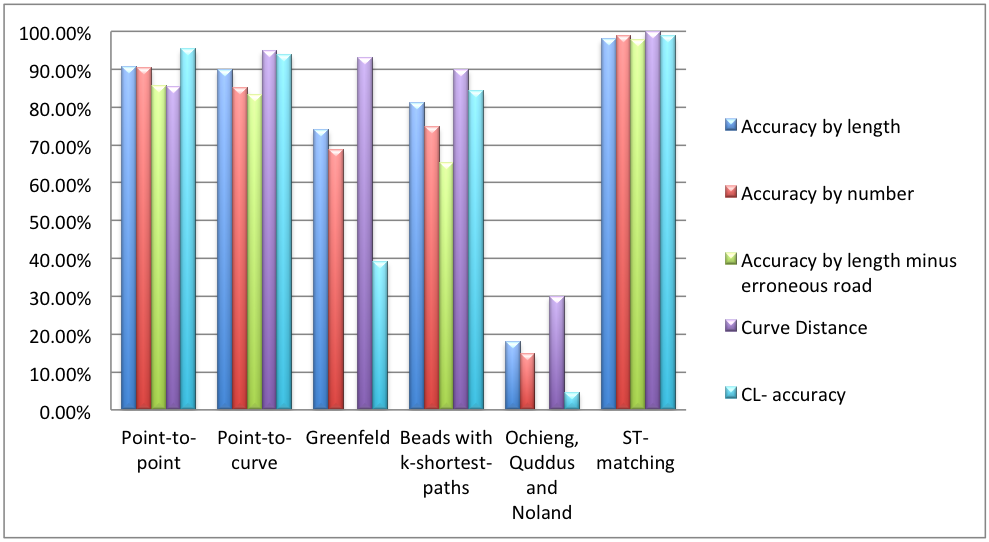


Figure 9.23: Results of the map matching algorithms on one computer generated trajectory with low sampling rate and measurement errors.

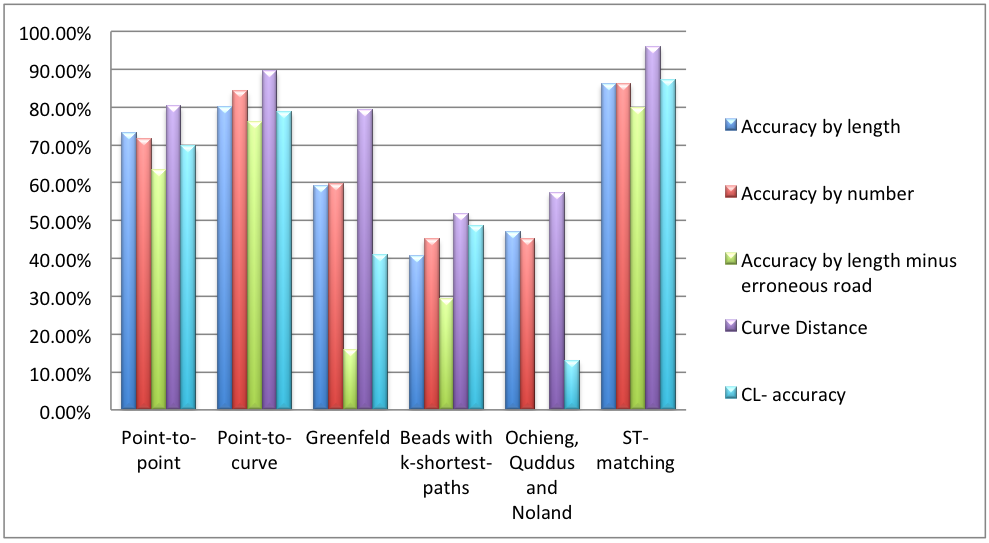


Figure 9.24: Average results of the map matching algorithms on ten computer generated trajectories with low sampling rate and measurement errors.

ST-matching algorithm, as well as the geometrical analysis algorithm, performs very well on this type of data. Greenfeld's algorithm, the space-time prism with k-shortest-paths algorithm, and the probabilistic algorithm perform poorly, as explained in Section 9.6.2.1.

9.7 Conclusion on map matching algorithms

In this section, we recapitulate and aggregate the above results and briefly discuss which algorithm suits which situation best. For a more detailed discussion of the results, we refer the reader to the previous sections in this chapter, which go into deeper detail on each of the test.

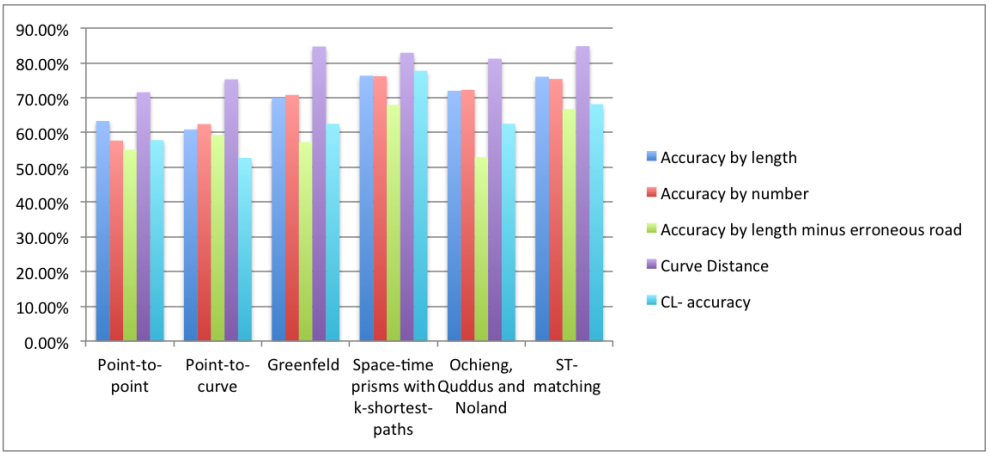


Figure 9.25: Average of the results of the data show in Figure 9.8, 9.10, 9.12, 9.14, 9.16, 9.18, 9.20, 9.22 and 9.24

To get an overview, we show in Figure 9.25 the average matching errors of all test on the dataset with multiple trajectory samples. In Figure 9.26, we present the average running time per case and in the last column an overall average.

In nearly every case, the geometric analysis algorithms (point-to-point and point-to-curve) perform the worst. The only case where using this type of algorithm is feasible is on devices with very lacking processing power. Since this type of algorithm requires less calculations than the other types of algorithms, they might be the only feasible ones in that case. In nearly any other case, the other algorithms perform as well or better than the geometric analysis algorithms.

The topological analysis algorithms, that we implemented, specifically space-time prism with k-shortest-paths algorithm, perform the best on av-

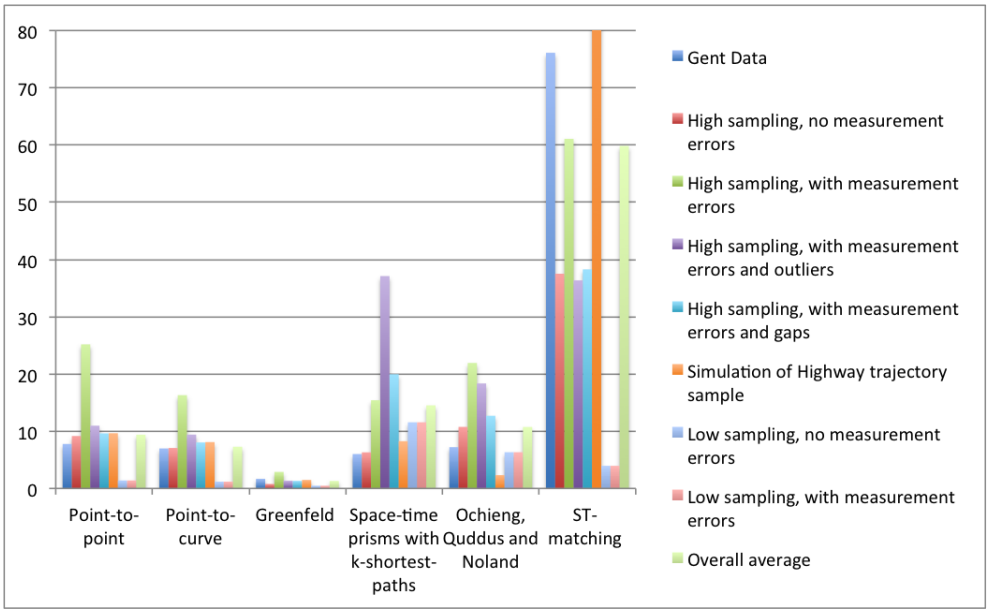


Figure 9.26: Average running times in seconds of the results of the data show in Figure 9.8, 9.10, 9.12, 9.14, 9.16, 9.18, 9.20, 9.22 and 9.24, where for visibility we cut the orange line at 80sec. where the running time was more then 200sec.

erage. It proves to work well in nearly any case, given a sufficiently high enough sampling rate. The only case where this algorithm might not perform well is when there are a lot of outliers in the data, or if the sampling rate is too low. Due to the way the algorithm calculates the path, outliers cause small hiccups in the calculation, which causes the algorithm to slow down a lot when encountering an outlier. In regular, everyday use, the space-time prism with k-shortest-paths algorithm seems to work best.

The probabilistic algorithms by Ochieng, Quddus and Noland also performed well in nearly every test case. The algorithm we implemented performed decently in most test cases, but had problems with long trajectories or trajectories with a low sampling rate. Its use should be avoided in these cases. It performed substantially better than the other algorithms when the data are located on a highway.

The ST-matching algorithm, an algorithm specifically created for data with a low sampling rate, had varying results on data with a high sampling rate. Due to these varying results, and the long run-time required to run this algorithm, it should be avoided for use on data with a high sampling rate. When encountering data with a low sampling rate however, the algorithm proves to calculate a very close match to the trajectory with a short running time.

Discussion

In the various chapters of this thesis, we already mention a number of open problems and directions for future research. This chapter adds some general comments to these previous observations.

The topic of this thesis is motivated by the abundance of spatio-temporal data that is collected nowadays. In particular, the presented research builds on *trajectory data of moving objects*, that is collected using GPS-equipped devices. In some particular areas, this data comes from moving animals that are typically unconstrained by physical, man-made networks for their movement. In the majority of cases, GPS-based data concerns humans. In average day situations, humans tend to move in constrained spaces, for example, on road networks.

The second part of this thesis concerns *map matching*, which is a preliminary step in the process of spatio-temporal and trajectory data analysis. Indeed, map-matching algorithms "clean" the often erroneous GPS-measured data by fitting it to the road network where human movement takes place. Measurement errors and outliers in the GPS-recorded spatio-temporal data points are corrected in this way and the data is ready for further analysis. In general, movement data hides a wealth of knowledge, which can be used, for instance, to predict traffic jams or determine different types of movement behaviour. Further examples of trajectory data analysis are trajectory clustering and predictive modeling by association rules.

A problem with trajectory data, even if it is already matched to a road network, is that it is only recorded at discrete moments in time. If the recording is frequent, the data may be readily usable. However, often, data is not recorded frequently enough. Urban tunnels, where high city buildings block satellite reception, are an example of a cause of missing or infrequent data. We show that the use of uncertainty techniques (such as space-time prisms) to cover unrecorded movement, is a valuable technique. Through experiments, we show that the incorporation of uncertainty improves the performance and accuracy of map-matching algorithms. The experimental results in Chapter 9

show that, on average, our space-time prism with k-shortest-paths algorithm can compete with the best map-matching algorithms. It even outperforms geometric map-matching algorithms.

Historically seen, the area of map matching is driven by experimental results. In this area there exists an abundance of publications that prove the superiority of some proposed algorithm over existing ones by experiments on some ad hoc data set of recorded trajectory samples. A general benchmark for the evaluation of map-matching algorithms is missing. In this field, there is also no repository of existing implementations. To compare our space-time prism with k-shortest-paths algorithm with existing map-matching techniques, we have implemented these existing algorithms ourselves (probably missing some optimalisations of other authors). The area of map matching is really missing a flexible benchmarking system, in a wide sense.

As mentioned before, matching recorded GPS points to a map is a cleaning step in the *Knowledge Discovering and Delivery* (KDD) process.

For some data mining techniques (like clustering), that are applied to this cleaned trajectory data, the study and design of distance or similarity measures on trajectory data is necessary. This brings us to the first part of this thesis. There, we study *double-cross similarity* of polylines that are the traces of trajectories. The main aim of this part is to gain insight in what double-cross similarity of polylines really means in a geometric sense. Locally, the double-cross formalism captures the turns made by a moving object. But its less clear what its meaning is globally (over complete polylines).

We use the double-cross matrices of (generalized) polylines and their differences to design measures of dissimilarity between polylines. Our algorithms, that are based on these dissimilarity measures, are guaranteed to terminate (in quadratic time) and when applied in various settings, like query-by-sketch, indexing and classification of terrain features, the experimental results prove to be very satisfactory. However, general principles, that indicate that this will always be the case, are still missing.

Another line of attack in our pursuit to gain understanding in double-cross similarity of polylines is illustrated by the question what the set of polylines is that have a given double-cross matrix. We approach this question algebraically and geometrically (by the concept of local carrier order). For a restricted class of polylines, namely polylines on a grid, this question is answerable. In this setting, it is easy to produce many polylines that have the same double-cross matrix as a given polyline and geometrically it is clear what double-cross similarity means.

We also study a related question that is relevant for the integrity of data: is a double-cross-like matrix realizable by a polyline in the plane? This problem will probably never be solvable in practice, but if we restrict the angles that can occur in a polyline to 90° or 45°, we do obtain efficient solutions. There

are a number of open problems related to this question. Can this problem be solved in practice for other restricted classes of polylines. We think of polylines in which all line segments are of equal length or polylines that take their angles from a wider class than the above.

Bibliography

[AAL+09] Divyakant Agrawal, Walid G. Aref, Chang-Tien Lu, Mohamed F. Mokbel, Peter Scheuermann, Cyrus Shahabi, and Ouri Wolfson (eds.), *17th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems, ACM-GIS 2009, November 4-6, 2009, Seattle, Washington, USA, Proceedings*, ACM, 2009.

[AB94] Ilan Adler and Peter A. Beling, *Polynomial algorithms for linear programming over the algebraic numbers*, Algorithmica **12** (1994), no. 6, 436–457.

[AERW03] Helmut Alt, Alon Efrat, Günter Rote, and Carola Wenk, *Matching planar maps*, SODA, ACM/SIAM, 2003, pp. 589–598.

[Bam12] Kristof Bamps, *Comparing study of map-matching algorithms*, Master's thesis, Hasselt University, 2012, Masterthesis (Hasselt University), Advisors: Bart Kuijpers and Bart Moelans.

[BK98] David Bernstein and Alain L. Kornhauser, *An introduction to map matching for personal navigation assistans*, Tech. report, Transportation Research Board, 1998.

[Boo86] Fred L. Bookstein, *Size and shape spaces for landmark data in two dimensions*, Statistical Science **1** (1986), no. 2, 181–222.

[BPR06] Saugata Basu, Richard Pollack, and Marie-Françoise Roy, *Algorithms in Real Algebraic Geometry*, Algorithms and Computation in Mathematics, Springer-Verlag New York, Inc., 2006.

[BPSW05] Sotiris Brakatsoulas, Dieter Pfoser, Randall Salas, and Carola Wenk, *On map-matching vehicle tracking data*, VLDB (Klemens Böhm, Christian S. Jensen, Laura M. Haas, Martin L. Kersten, Per-Åke Larson, and Beng Chin Ooi, eds.), ACM, 2005, pp. 853–864.

[Can88] John Canny, *Some algebraic and geometric computations in PSPACE*, Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing (STOC 1988), ACM, 1988, pp. 460–467.

[CLO97] David A. Cox, John Little, and Donal O'Shea, *Ideals, varieties, and algorithms - an introduction to computational algebraic geometry and commutative algebra (2. ed.)*, Undergraduate texts in mathematics, Springer, 1997.

[Col75] George E. Collins, *Hauptvortrag: Quantifier elimination for real closed fields by cylindrical algebraic decomposition*, Automata Theory and Formal Languages, 2nd GI Conference, Kaiserslautern, May 20-23, 1975 (H. Barkhage, ed.), Lecture Notes in Computer Science, vol. 33, Springer, 1975, pp. 134–183.

[Dij59] Edsger W. Dijkstra, *A note on two problems in connexion with graphs*, Numerische Mathematik **1** (1959), no. 1, 269–271.

[DM98] Ian L. Dryden and Kantilal Varichand Mardia, *Statistical shape analysis*, Wiley series in probability and statistics, John Wiley & Sons, Chichester, New York, 1998.

[DS97] Andreas Dolzmann and Thomas Sturm, *Redlog: Computer algebra meets computer logic*, SIGSAM Bull. **31** (1997), no. 2, 2–9.

[dW04] Nico Van de Weghe, *Representing and reasoning about moving objects: A qualitative approach*, Ph.D. thesis, Ghent University, Faculty of Sciences, Department of Geography, 2004.

[dWKpBM05] Nico Van de Weghe, Bart Kuijpers, Peter Bogaert, and Philippe De Maeyer, *A qualitative trajectory calculus and the composition of its relations*, GeoS (M. Andrea Rodríguez, Isabel F. Cruz, Max J. Egenhofer, and Sergei Levashkin, eds.), Lecture Notes in Computer Science, vol. 3799, Springer, 2005, pp. 60–76.

[Edd04] Sean R. Eddy, *What is a hidden Markov model?*, Nature Biotechnology **22** (2004), no. 10, 1315–1316.

[Ege03] Max J. Egenhofer, *Approximation of geospatial lifelines*, SpadaGIS, Workshop on Spatial Data and Geographic Information Systems (Elisa Bertino and Leila De Floriani, eds.), University of Genova, 2003.

[For90] Kenneth D. Forbus, *Readings in qualitative reasoning about physical systems*, ch. Qualitative physics: past present and future, pp. 11–39, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.

[Fre92] Christian Freksa, *Using orientation information for qualitative spatial reasoning*, Spatio-Temporal Reasoning (Andrew U. Frank, Irene Campari, and Ubaldo Formentini, eds.), Lecture Notes in Computer Science, vol. 639, Springer, 1992, pp. 162–178.

[Ger99] John S. Gero, *Representation and reasoning about shapes: Cognitive and computational studies in visual reasoning in design*, Spatial Information Theory: Cognitive and Computational Foundations of Geographic Information Science, International Conference COSIT '99, Stade, Germany, August 25-29, 1999, Proceedings (Christian Freksa and David M. Mark, eds.), Lecture Notes in Computer Science, vol. 1661, Springer, 1999, pp. 315–330.

[GH01] Marc Giusti and Joos Heintz, *Kronecker's smart, little black boxes*, Foundations of Computational Mathematics (Cambridge) (R. DeVore, Iserles A., and E. Suli, eds.), Cambridge University Press, 2001, pp. 69–104.

[Ghy07] Kristof Ghys, *Map matching tracking data*, Master's thesis, Hasselt University, 2007, Masterthesis (Hasselt University), Advisors: Bart Kuijpers and Bart Moelans.

[GKpM+09] Kristof Ghys, Bart Kuijpers, Bart Moelans, Walied Othman, Dries Vangoidsenhoven, and Alejandro A. Vaisman, *Map matching and uncertainty: an algorithm and real-world experiments*, in Agrawal et al. [AAL+09], pp. 468–471.

[Got03] Björn Gottfried, *Tripartite line tracks qualitative curvature information*, in Kuhn et al. [KWT03], pp. 101–117.

[GP08] Fosca Giannotti and Dino Pedreschi (eds.), *Mobility, Data Mining and Privacy - Geographic Knowledge Discovery*, Springer, 2008.

[Gre02] Joshua S. Greenfeld, *Matching gps observations to locations on a digital map*, 81th Annual Meeting of the Transportation Research Board **1** (2002), no. 3, 164–173.

[GS05] Ralf Hartmut Güting and Markus Schneider, *Moving objects databases*, Morgan Kaufmann, 2005.

[Häg70] Torsten Hägerstrand, *What about people in regional science?*, Papers Regional Science Association, vol. 24, 1970, pp. 7–21.

[HE02] Kathleen Hornsby and Max J. Egenhofer, *Modeling moving objects over multiple granularities*, Annals of Mathematics and Artificial Intelligence **36** (2002), no. 1-2, 177–194.

[HKp04] Joos Heintz and Bart Kuijpers, *Constraint databases, data structures and efficient query evaluation*, Constraint Databases and Applications (Bart Kuijpers and Peter Z. Revesz, eds.), Lecture Notes in Computer Science, vol. 3074, Springer, 2004, pp. 1–24.

[HKP13] Joos Heintz, Bart Kuijpers, and Andres Rojas Paredes, *Software engineering and complexity in effective algebraic geometry*, Journal of Complexity **29** (2013), no. 1, 92–138.

[HNR68] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael, *A formal basis for the heuristic determination of minimum cost paths*, IEEE Transactions on Systems Science and Cybernetics **4** (1968), no. 2, 100–107.

[Hon] Hoon Hong, *QEPCAD*, `www.usna.edu/CS/qepcadweb/B/QEPCAD.html`.

[Hum06] Britta Hummel, *Map Matching for vehicle guidance (draft)*, 2006.

[Hum10] Derk Humblet, *Kwalitatieve afstandsmaten voor trajecten*, Master's thesis, Hasselt University, 2010, Masterthesis (Hasselt University), Advisors: Bart Kuijpers and Bart Moelans.

[JBR98] Michel Coste Jacek Bochnak and Marie-Francoise Roy, *Real algebraic geometry*, Ergebenisse der Mathematik und ihrer Grenzgebiete. Folge 3., vol. 36, Springer-Verlag Berlin Heidelberg, 1998.

[Jun93] Erland Jungert, *Symbolic spatial reasoning on object shapes for qualitative matching*, COSIT, 1993, pp. 444–462.

[Kal60] Rudolph Emil Kalman, *A new approach to linear filtering and prediction problems*, Transactions of the ASME–Journal of Basic Engineering **82, Series D** (1960), no. Series D, 35–45.

[KE03] Lars Kulik and Max J. Egenhofer, *Linearized terrain: Languages for silhouette representations*, in Kuhn et al. [KWT03], pp. 118–135.

[Kho91] Askold G. Khovanskiĭ, *Fewnomials*, Translations of mathematical monographs, American Mathematical Society, 1991.

[KLH07] John Krumm, Julie Letchner, and Eric Horvitz, *Map matching with travel time constraints*, Society of Automotive Engineers (SAE) 2007 World Congress, April 2007.

[KM01] John T. Kent and Kanti V. Mardia, *Shape, procrustes tangent projections and bilateral symmetry*, Biometrika **88** (2001), 469–485.

[KpMdW06] Bart Kuijpers, Bart Moelans, and Nico Van de Weghe, *Qualitative polyline similarity testing with applications to query-by-sketch, indexing and classification*, GIS (Rolf A. de By and Silvia Nittel, eds.), ACM, 2006, pp. 11–18.

[KWT03] Werner Kuhn, Michael F. Worboys, and Sabine Timpf (eds.), *Spatial information theory. foundations of geographic information science, international conference, cosit 2003, ittingen, switzerland, september 24-28, 2003, proceedings*, Lecture Notes in Computer Science, vol. 2825, Springer, 2003.

[Ley88] Michael Leyton, *A process-grammar for shape*, Artif. Intell. **34** (1988), no. 2, 213–247.

[LL00] Longin Jan Latecki and Rolf Lakämper, *Shape similarity measure based on correspondence of visual parts*, IEEE Trans. Pattern Anal. Mach. Intell. **22** (2000), no. 10, 1185–1190.

[LZZ+09] Yin Lou, Chengyang Zhang, Yu Zheng, Xing Xie, Wei Wang, and Yan Huang, *Map-matching for low-sampling-rate gps trajectories*, in Agrawal et al. [AAL+09], pp. 352–361.

[Mea01] Richard C. Meathrel, *A general theory of boundary-based qualitative representation of two-dimensional shape*, Ph.D. thesis, University of Exeter, UK, 2001.

[MG06] Jirí Matousek and Bernd Gärtner, *Understanding and using linear programming*, Universitext, Springer, 2006.

[Mil05] Harvey J. Miller, *A measurement theory for time geography*, Geographical Analysis **37** (2005), 17–45.

[MM92] Farzin Mokhtarian and Alan K. Mackworth, *A theory of multiscale, curvature-based shape representation for planar curves*, IEEE Trans. Pattern Anal. Mach. Intell. **14** (1992), no. 8, 789–805.

[NK09] Paul Newson and John Krumm, *Hidden markov map matching through noise and sparseness*, in Agrawal et al. [AAL+09], pp. 336–343.

[Oth09] Walied Othman, *Uncertainty management in trajectory databases*, Ph.D. thesis, Hasselt University, 2009.

[PJ99] Dieter Pfoser and Christian S. Jensen, *Capturing the uncertainty of moving-object representations*, SSD (Ralf Hartmut Güting, Dimitris Papadias, and Frederick H. Lochovsky, eds.), Lecture Notes in Computer Science, vol. 1651, Springer, 1999, pp. 111–132.

[QON03] Mohammed A. Quddus, Washington Yotto Ochieng, and Robert B. Noland, *Map matching in complex urban road networks*, Brazilian Journal of Cartography Revista Brasileira de Cartografia **55** (2003), no. 2, 1–18.

[Qud06] Mohammed A. Quddus, *High integrity Map Matching algorithms for advanced transport telematics applications*, Ph.D. thesis, Imperial College, London, U.K., January 2006.

[QZON03] Mohammed A Quddus, L. Zhao, Washington Yotto Ochieng, and Robert B. Noland, *An extended Kalman Filter algorithm for integrating GPS and low-cost dead reckoning system data for vehicle performance and emissions monitoring*, The Journal of Navigation **56** (2003), 257–275.

[RN07] Jochen Renz and Bernhard Nebel, *Qualitative spatial reasoning using constraint calculi*, Handbook of Spatial Logics (Marco Aiello, Ian Pratt-Hartmann, and Johan van Benthem, eds.), Springer, 2007, pp. 161–215.

[SC04] Salman Syed and M. Elizabeth Cannon, *Fuzzy logic based map matching algorithm for vehicle navigation system in urban canyons*, Proceedings of the 2004 National Technical Meeting of The Institute of Navigation (2004), 982–993.

[Sch96] Christoph Schlieder, *Qualitative shape representation*, Geographic Objects with Indeterminate Boundaries, Taylor & Francis, 1996, pp. 123–140.

[SN01] Alexander Scivos and Bernhard Nebel, *Double-crossing: Decidability and computational complexity of a qualitative calculus for navigation*, Spatial Information Theory: Foundations of Geographic Information Science, International Conference, COSIT 2001, Morro Bay, CA, USA, September 19-23, 2001, Proceedings (Daniel R. Montello, ed.), Lecture Notes in Computer Science, vol. 2205, Springer, 2001, pp. 431–446.

[Tar51] Alfred Tarski, *A decision method for elementary algebra and geometry*, University of California Press, 1951.

[Vit67] Andrew J. Viterbi, *Error bounds for convolutional codes and an asymptotically optimum decoding algorithm*, IEEE Transactions on Information Theory **13** (1967), no. 2, 260–269.

[WBK00] Christopher E White, David Bernstein, and Alain L Kornhauser, *Some map matching algorithms for personal navigation assistants*, Transportation Research Part C: Emerging Technologies **8** (2000), no. 16, 91 – 108.

[Wik] Wikipedia, *Fuzzy logic*, `https://en.wikipedia.org/wiki/Fuzzy_logic`.

[Wol] Wolfram Research, *Mathematica 9*, `www.wolfram.com`.

[Wol02] Ouri Wolfson, *Moving objects information management: The database challenge*, NGITS (Alon Y. Halevy and Avigdor Gal, eds.), Lecture Notes in Computer Science, vol. 2382, Springer, 2002, pp. 75–89.

[Yen72] Jin Y. Yen, *Finding the lengths of all shortest paths in n-node nonnegative-distance complete networks using $\frac{1}{2}n^3$ additions and n^3 comparisons*, Journal of the ACM **19** (1972), no. 3, 423–424.

[YW04] Huabei Yin and Ouri Wolfson, *A weight-based map matching method in moving objects databases*, SSDBM, IEEE Computer Society, 2004, pp. 437–438.

[Zad65] Lotfi A. Zadeh, *Fuzzy sets*, Information and Control **8** (1965), no. 3, 338–353.

[ZF96] Kai Zimmermann and Christian Freksa, *Qualitative spatial reasoning using orientation, distance,and path knowledge*, Applied Intelligence **6** (1996), no. 1, 49–58.

[Zha97] Yilin Zhao, *Vehicle location and navigation systems: Intelligent transportation systems*, Navtech Seminars and GPS Supply, 1997.

Nederlandstalige samenvatting

Polylijnen komen op meerdere manieren naar voren in Geografische Informatiesystemen (GIS). Een voorbeeld zien we bij het verzamelen van gegevens van bewegende objecten, waar de trajecten van bewegende objecten (zoals bijvoorbeeld voetgangers, auto's, dieren, ...) die een met GPS uitgerust apparaat meedragen, worden verzameld in de vorm van tijd-ruimte punten die op bepaalde (on)regelmatige tijdstippen geregistreerd worden. Het ruimtelijk spoor van zo een beweging is een verzameling van punten in de twee-dimensionale geografische ruimte. Er bestaan verschillende methoden om trajecten uit te breiden tussen deze gemeten punten. Lineaire interpolatie is een populaire methode [GS05]. De resulterende curve in de twee-dimensionale geografische ruimte is een *polylijn*.

Alhoewel de meeste mensen een GPS gebruiken als navigatie-instrument, kan het ook gebruikt worden om de positie van bewegende objecten op te slaan voor latere *tijd-ruimtelijke data-analyse* (we verwijzen naar [GP08] voor een overzicht van tijd-ruimtelijke data-mining en -analyse). Bijvoorbeeld, we kunnen de routes gevolgd door een persoon of een groep personen analyseren om er verborgen patronen in proberen te ontdekken. Een groot nadeel van het gebruik van GPS is dat de geregistreerde coördinaten niet altijd zeer accuraat zijn. Ze zullen niet altijd overeenkomen met de wegen die een auto of een voetganger volgde. De meeste GPS-toestellen, die in auto's gebruikt worden, brengen deze meetfouten in rekening door de gemeten locaties op de gevolgde straten af te beelden in plaats van de satellietgegevens gewoon weer te geven. Het algemene probleem van het verbinden van gemeten GPS punten aan locaties op een wegennetwerk wordt *map matching* genoemd. Dit is het onderwerp van deel II van dit proefschrift (zie verder).

Een ander voorbeeld van het gebruik van polylijnen in GIS komt uit het gebied van *vorm-herkenning en -retrieval*, dat voorkomt in gebieden zoals computer-visie, beeld-analyse en GIS in het algemeen. Hier komen gesloten

polylijnen (waar het begin- en eindpunt samenvallen) of polygonen, vaak voor als de randen van twee-dimensionale vormen of gebieden. Vorm-herkenning en retrieval zijn centrale problemen in deze context.

In de gegeven voorbeelden, zijn er, ruw geschetst, twee zeer verschillende benaderingen om met polylijnen, polygonale curves en vormen om te gaan. Aan de ene kant is er de *kwantitatieve* aanpak en aan de andere kant is er de *kwalitatieve* aanpak. Aanvankelijk gaat de meeste onderzoeksaandacht uit naar de kwantitatieve methoden [Boo86, DM98, KM01, MM92]. Pas later krijgt de kwalitatieve benadering meer aandacht. Dit wordt vooral ondersteund door onderzoek in de cognitieve wetenschappen, waar argumenten aangebracht worden die aantonen dat kwalitatieve methoden om vormen voor te stellen veel expressiever zijn dan hun kwantitatieve tegenhangers en dat ze beter overeenstemmen met de manier waarmee mensen over hun ruimtelijke omgeving nadenken [Ger99]. Polygonale vormen en polygonale curves zijn complexe ruimtelijke fenomenen en het is algemeen aanvaard dat een kwalitatieve voorstelling ervan geschikter is om hen te behandelen [Mea01].

Binnen de kwalitatieve benaderingen om twee-dimensionele bewegingen of vormen te beschrijven, kunnen twee belangrijke benaderingen worden ondersheiden: de regio-gebaseerde en de omtrek-gebaseerde aanpak. De *regio-gebaseerde* aanpak maakt gebruik van concepten zoals circulariteit, oriëntatie ten opzichte van de coördinatenas en kan alleen onderscheid maken tussen vormen met grote verschillen [Sch96]. De *omtrek-gebaseerde* aanpak maakt gebruik van concepten zoals extrema in, en veranderingen van de kromming van de polylijn die de vorm voorstelt. Deze aanpak geeft aanleiding tot nauwkeurigere toepassingen waarmee polylijnen en polygonen kunnen worden onderscheiden. Voorbeelden van de omtrek-gebaseerde benaderingen kunnen gevonden worden in [Got03, Jun93, KE03, LL00, Ley88, Mea01, Sch96].

De principes achter de kwalitatieve benadering zijn ook gerelateerd aan het domein van spatial reasoning. *Spatial reasoning* (of ruimtelijk redeneren) heeft als één van haar belangrijkste doelstellingen om geografische informatie op een kwalitatieve manier te representeren, om zo hierover te kunnen redeneren. We verwijzen voor meer informatie naar hoofdstuk 12 in [GP08] (ook voor spatio-temporal reasoning). Dit kan worden gezien als de verwerking van informatie over een ruimtelijke omgeving die onmiddellijk beschikbaar is voor de mens (of dier) door middel van directe observatie. De reden voor het gebruik van een *kwalitatieve representatie* is dat de beschikbare informatie vaak onnauwkeurig, gedeeltelijke en subjectief is [Fre92]. Als we terugkeren naar het voorbeeld van traject-data, kunnen we zien dat als een persoon zich oriënteert op een bepaalde locatie in een stad en vervolgens dan weg gaat van deze plaats. De persoon zich positioneert op de huidige locatie met behulp van een mentale kaart met navigaties ten opzichte van het oorspronkelijke startpunt, en niet met exacte meetgegevens over zijn of haar traject. Voor het doel van navigatie,

zal een persoon zich bijvoorbeeld herinneren: “Ik verliet het station en ging meteen rechtdoor; ik passeerde een kerk aan mijn rechterkant; vervolgens sloeg ik twee keer af naar links; . . . ” Dit brengt ons bij deel I van dit proefschrift.

Deel I: De double-cross beschrijving van polylijnen. Eén van de formalismen om polylijnen in het vlak te beschrijven, wordt gegeven door de *double-cross calculus*. In deze methode beschrijft een *double-cross matrix* de relatieve positie van twee lijnstukken in een polylijn. Dit wordt gedaan door deze twee lijnstukken te beschrijven ten opzichte van een *double cross* (of dubbel kruis), dat gebaseerd is op de startpunten van deze lijnstukken. De double-cross calculus werd geïntroduceerd als een formalisme om kwalitatief een configuratie van vectoren te representeren in het vlak [Fre92, ZF96]. Voor een overzicht van het gebruik van “constraint calculi” binnen het gebied van kwalitatieve *spatial reasoning*, verwijzen we naar [RN07]. In het double-cross formalisme, wordt de relatieve positie (of richting) van twee (gelokaliseerde) vectoren gecodeerd door middel van een 4-tupel, waarvan de elementen uit de verzameling $\{0, +, -\}$ komen. Zo een 4-tupel beschrijft de relatieve oriëntatie van twee vectoren ten opzichte van elkaar. Het double-cross formalisme wordt bijvoorbeeld gebruikt in de *kwalitatieve trajectcalculus*, die op haar beurt, is gebruikt om de gelijkenis van polylijnen te testen in toepassingen als “query door sketch”, indexering en classificatie [KpMdW06].

Twee polylijnen worden *double-cross similar* (of gelijkend) genoemd als hun double-cross matrices identiek zijn. Twee polylijnen, die heel verschillend zijn vanuit een kwantitatief of metrisch oogpunt, kunnen toch dezelfde double-cross matrices hebben. Het idee is dat zij een vergelijkbaar patroon volgen van relatieve bochten. Dit weerspiegelt hoe mensen bewegingspatronen visualiseren en onthouden.

Algebraïsche en meetkundige karakterisaties van de double-cross matrices van polylijnen. In hoofdstuk 3, geven we een algebraïsche en meetkundige interpretatie van de double-cross matrix van een polylijn en double-cross gelijkenis van polylijnen. Om te beginnen drukken we de double-cross matrix van een polylijn uit als een collectie van beperkingen, in de vorm van veelterm-gelijkheden en -ongelijkheden op de coördinaten van de gemeten punten van een polylijn (de hoekpunten). Deze algebraïsche representatie kan worden gebruikt om effectief de double-cross gelijkenis van polylijnen te controleren. Dit kan verder ook gebruikt worden om met software-pakketten zoals MATHEMATICA [Wol] double-cross gelijkende polylijnen te genereren. De algebraïsche representatie van de double-cross matrix laat ons ook toe om een aantal eigenschappen van de double-cross matrices van polylijnen te bewijzen. Een voorbeeld van zo een eigenschap is de symmetrie van een double-cross matrix langs haar hoofddiagonaal.

Vervolgens gaan we naar een meetkundige interpretatie van de double-cross gelijkenis van twee polylijnen. Deze geometrische interpretatie is gebaseerd op lokale meetkundige informatie van de polylijn in haar hoekpunten. Deze informatie wordt de *local carrier order* genoemd en gebruikt (lokale) kompasrichtingen in de hoekpunten van een polylijn om de relatieve positie van de overige hoekpunten te lokaliseren. Ons belangrijkste resultaat in deze context zegt dat twee polylijnen double-cross similar zijn als en slechts als ze dezelfde *local carrier order* hebben.

Uit de definitie van een double-cross matrix van een polylijn is het duidelijk dat deze matrix niet verandert bij bijvoorbeeld, een rotatie of translatie van de polyline in het twee-dimensionale vlak. In een laatste deel van hoofdstuk 3, identificeren we de exacte verzameling van transformaties van het twee-dimensionale vlak die de double-cross matrices onveranderd laten. Ons belangrijkste resultaat is dat de grootste groep van transformaties van het vlak, waaronder de double-cross matrix invariant is, bestaat uit de *similariteiten* van het vlak. Dit resultaat laat ons bijvoorbeeld toe om alle eigenschappen van double-cross matrices aan te tonen met polylijnen die starten in de oorsprong van het vlak en waarvan het eerste lijnstuk als lengte één heeft.

Algoritmen om double-cross gelijkenis te testen. In hoofdstuk 4 geven we een algoritme, gebaseerd op het double-cross formalisme, om te testen of twee polylijnen (of polygonen) double-cross similar zijn. Om de mate van gelijkenis tussen twee polylijnen (niet noodzakelijk van dezelfde grootte) te bepalen, gaat het algoritme eerst hun "gegeneraliseerde polylijnen" berekenen. Een gegeneraliseerde polylijn van een polylijn bestaat uit bijna even lange lijnsegmenten en benadert de lengte van de gegeven polylijn binnen een ε-foutenmarge. In een volgende stap, bepaalt het algoritme de double-cross matrices van de gegeneraliseerde polylijnen en het verschil tussen deze matrices wordt gebruikt als basis om de mate van gelijkenis tussen de gegeven polylijnen te meten. We tonen ook aan dat ons algoritme steeds termineert en we geven de sequentiële tijdscomplexiteit.

Hoofdstuk 4 eindigt met een aantal toepassingen. We passen onze werkwijze toe op query-by-sketch, indexering van polylijn-databases en classificatie van terrein eigenschappen. We geven ook experimentele resultaten voor elk van deze toepassingen.

De double-cross matrix van polylijnen op een raster. In hoofdstuk 5 bestuderen we eigenschappen van de double-cross matrices van polylijnen die zijn gelegen op een *raster* (of rooster) in het twee-dimensionale vlak. We veronderstellen dat de rasterlijnen parallel zijn met de standaard x- en y-assen van het vlak. Polylijnen op rasters kunnen voortvloeien uit trajecten op een Manhattan-achtig wegennet.

We geven een effectieve karakterisering van wat double-cross gelijkenis be-

tekent voor polylijnen op een raster. Voor een polylijn op een raster, noemen we de verticale en horizontale rechte lijnen door haar hoekpunten de *verticale en horizontale dragers* en we noemen de volgorde waarin ze verschijnen als we de polylijn doorlopen van haar begin tot haar einde de V- en H-orde van de polylijn. We noemen twee polylijnen VH*-equivalent* als ze dezelfde V-orde en dezelfde H-orde hebben.

Aan een polylijn op een raster, associëren we ook een *canonieke polylijn*, die VH-equivalent is met de originele polylijn en dezelfde double-cross matrix heeft als de originele polylijn. Het blijkt dat VH-equivalentie de meetkundige informatie weerspiegelt die in de double-cross matrix beschreven is: twee polylijnen op een raster hebben dezelfde double-cross matrix als en slechts als ze VH-equivalent zijn (hoewel hun laatste lijnsegmenten kunnen verschillen in lengte). We geven ook een algoritme dat op input een double-cross(-achtige) matrix van grootte $N \times N$, in $O(N^2)$ tijd controleert of deze double-cross matrix werkelijk realiseerbaar is door een polylijn op een raster. In hoofdstuk 6 gaan we dieper in op het begrip realiseerbaarheid. Voor polylijnen waarvan alle hoekpunten, snijpunten zijn van het raster, kan het bovenstaande resultaat worden verbeterd. Hier kan, nadat de realiseerbaarheid van een matrix is gecontroleerd (weerom in $O(N^2)$ tijd), een polylijn worden gegenereerd in $O(N)$ tijd, die de gegeven matrix als haar double-cross matrix heeft.

Over de realiseerbaarheid van de double-cross matrices. In hoofdstuk 6, pakken we het probleem van de realiseerbaarheid van de double-cross-achtige matrices aan. Niet elke $N \times N$ matrix van 4-tuples uit de verzameling $\{-, 0, +\}$ is de double-cross matrix van een polylijn in het vlak met $N + 1$ hoekpunten. Dit geeft aanleiding tot het volgende *beslissingsprobleem*: Gegeven een $N \times N$ matrix van 4-tuples uit $\{-, 0, +\}$, beslis of het de double-cross matrix van een polylijn (met $N + 1$ hoekpunten) is. En zo ja, produceer een voorbeeld (of vele voorbeelden) van een polylijn met deze double-cross matrix.

Uit hoofdstuk 3 kennen we reeds de verzameling van veeltermige (on)gelijkheden over de coördinaten van de hoekpunten van een polylijn, die de informatie uitdrukken die in de double-cross matrix van een polylijn bevat is. Sinds de eerste-orde logica over de reële getallen (of de elementaire meetkunde) beslisbaar is [Tar51], kunnen we onmiddellijk inzien dat het bovenstaande beslissingsprobleem, ook effectief beslisbaar is. We blijven echter zitten met de vraag wat de tijdscomplexiteit van ons beslissingsprobleem juist is.

In de computationele algebraïsche meetkunde kan dit beslissingsprobleem worden aanzien als een test die nagaat of een stelsel van kwadratische vergelijkingen in $2(N + 1)$ variabelen voldaan kan worden. De bestaande en beste algoritmen voor ons probleem (inclusief het produceren van een voorbeeld) lossen dit probleem echter op tegen een exponentiële tijdskost. Ons beslissingsprobleem heeft vele bijzonderheden: de polynomen zijn homogeen van

graad twee en gebruiken weinig termen (of monomials) en elk van hen gebruikt slechts zes variabelen. Dit helpt echter niet om een betere tijdscomplexiteit te verkrijgen. Van ons probleem is geweten dat het NP-hard is.

In dit hoofdstuk richten we ons op deelklassen van bovengenoemd beslissingsprobleem waarvoor we beslissingsalgoritmen kunnen geven die in polynomiale tijd werken. Een eerste deelklasse wordt verkregen door ons te beperken tot polylijnen waarin de opeenvolgende lijnstukken hoeken maken die veelvouden van 90° zijn. Voor dit deelprobleem, geven wij een $O(N^2)$-tijd beslissingsprocedure. Ook het produceren van polylijnen kan in dezelfde tijd uitgevoerd worden.

Vervolgens richten we onze aandacht op polylijnen waarin de opeenvolgende lijnstukken hoeken maken die veelvouden van 45° zijn. Om het meer ingewikkelde geval van 45°-polylijnen op te lossen, introduceren we de pool--coördinaten-representatie van de double-cross matrices van polylijnen. We geven vertalingen (in beide richtingen) tussen de representatie in het Cartesisch coördinatenstelsel en het pool-coördinatenstelsel. Met behulp van pool-coördinaten kan ons beslissingsprobleem worden teruggebracht tot een *lineair programmeerprobleem* (echter met algebraïsche coëfficiënten). Ook hier geven we een beslissingsprocedure in polynomiale tijd (die ook voorbeelden als bij-product produceert). Dit resultaat heeft een aantal gevolgen voor de *convexiteit* van de oplossingsverzameling die bestaat uit alle 45°-polylijnen die een double-cross matrix realiseren.

Part II: Map-matching-technieken voor traject-gegevens.

Een veelvoorkomend probleem in het gebied van *moving object databases* (MOD) is de reconstructie van een traject dat enkel als traject-monster (of trajectory sample) gemeten wordt door, bijvoorbeeld, een GPS-toestel. Trajectory samples worden automatsich verzameld door toestellen die de geografissche locatie kunnen bepalen (zoals GPS en binnenkort Galileo). Tijdens het voorbije decenium zijn GPS-gebaseerde navigatie-systemen alsmaar populairder geworden. Meer dan eens vallen de posities die door deze toestellen gementen worden buiten het wegennetwerk dat door het bewegend object gevolgd wordt. Typisch zullen ongeveer vijfennegentig procent van de gemeten tijd-ruimtelijke punten buiten het wegennetwerk vallen, wanneer de positie van een wagen of een voetganger met GPS geregistreerd wordt. Behalve meetfouten zijn er nog andere problemen met gegevens die in de praktijk gemeten worden. We denken hier aan verkeersopstoppingen en gaten in een traject tussen gemeten locaties die veroorzaakt kunnen worden door tunnels of zogeheten "stadstunnels" (dit zijn plaatsen in een stad waar hoge gebouwen de goede ontvangst van satellieten verhinderen). Bijgevolg is er een frequente noodzaak om de positie van een bewegend object te koppelen aan een digitale

kaart (van een wegennetwerk). Het algemene probleem van het koppelen van GPS-locaties aan locaties op een wegennetwerk noemt men *map matching.*

Inleiding tot map matching. In hoofdstuk 7 geven we een vrijwel volledig overzicht van de belangrijkste soorten van bestaande map matching algoritmen. We klasseren deze algoritmen ook naargelang de operationele manieren waarop ze gebruikt worden (online versus offline; lage versus hoge sampling frequentie; etc.). In dit hoofdstuk geven we ook een overzicht van de belangrijkste karakteristieken van verzamelingen van traject-gegevens.

Een onzekerheid-gebaseerd algoritme voor map matching. Er zijn reeds vele algoritmen voorgesteld om het map matching probleem op te lossen [BK98, WBK00]. Geen van de voorgestelde oplossingen beschouwt echter de onzekerheid die veroorzaakt wordt door het ontbreken van informatie over de locatie waar een bewegend object zich kan bevinden tussen gemeten GPS-loacties. Lineaire interpolatie tussen gemeten locaties veronderstelt dat het object zich tussen twee meetpunten aan constante, minimale snelheid verplaatst. Dit is een klassieke methode die echter niet zeer realistisch is. Een meer realistisch model is gebaseerd op de notie van *onzekerheid* (of uncertainty). Een voorbeeld van zo een model maakt gebruik van ruimte-tijd prisma's (of space-time prisms) [Ege03, Häg70, HE02, PJ99, Mil05]. In dit model worden de onbekende, maar mogelijke locaties van een bewegend object tussen twee gemeten locaties bepaald door gebruik te maken van achtergrond-informatie, zoals fysische of wettelijke snelheidsbeperkingen.

In dit deel van het proefschrift bestuderen we de relatie tussen map matching en onzekerheid. We stellen een nieuw map matching algoritme voor dat "weighted k-shortest paths" [Yen72] combineert met space-time prisms. Op deze manier bekomen we een algorime dat op vele soorten traject-gegevens toepasbaar is en dat beter presteert dan bestaande algoritmen [GKpM$^+$09].

We passen ons algoritme toe op echte traject data, alsook op computergegenereerde data die bestaat uit verschillende soorten trajecten met uiteenlopende eigenschappen. In sommige data worden observaties frequent en regelmatig geregistreerd. In andere data worden ze onregelmatigere of minder frequent geregistreerd. Zo verkrijgen we verschillende data die overeenkomen met het klassieke onderscheid tussen lage en hoge sampling frequentie, die we dikwijls in de literatuur tegenkomen. Bovendien vergelijken we de resultaten van ons nieuw space-time prism en k-shortest path algoritme met een aantal bestaande algoritmen en dit op een ruime variatie van trajectory sample gegevensverzamelingen met verschillende eigenschappen.

We tonen aan dat het inbouwen van onzekerheid in het map matching-proces tot betere resultaten en meer positieve matchings leidt. Dit positief resultaat compenseert ruimschoots voor de langere uitvoeringstijden, die echter redelijk blijven.

De voornaamste bijdrage van hoofdstuk 8 is het nieuwe map matching algoritme dat space-time prisms in combinatie met het weighted k-shortest path algoritme gebruikt. Een belangrijke component in dit algoritme is de berekening van de "bounding box" van de ruimtelijke projectie van een space-time prism. Deze bounding box beperkt het aantal kandidaat wegsegmenten dat we in het map matching proces moeten beschouwen.

Experimentele evaluatie van map matching algoritmen. In hoofdstuk 9, bestuderen we, om te beginnen, het accuraat vergelijken van de prestaties van verschillende map matching algoritmen. We geven een overzicht van verschillende eigenschappen van trajectory samples en bespreken manieren om deze eigenschappen te realiseren in computer-gegenereerde gegevens.

We overlopen evenzeer een aantal bestaande methoden om de accuraatheid van map matching algoritmen te meten op verschillende soorten datatypes [LZZ+09, WBK00]. We stellen ook een nieuwe manier voor om de accuraatheid van algoritmen te meten. Deze nieuwe methode lijdt niet aan de nadelen van bestaande methoden.

Verder is hoofdstuk 9 gewijd aan experimentele resultaten. We tonen een aantal tests van de verschillende map matching algoritmen op een variëteit van trajectory sample data. Het doel is om vast te stellen welk type map matching algoritme het meest geschikt is voor een bepaald type van trajectory samples. We hebben een aantal bestaande map matching algoritmen geïmplementeerd en vergelijken deze implementaties met die van ons eigen algoritme dat op onzekerheid gebaseerd is. De experimentele resultaten tonen aan dat ons space-time prisms in combinatie met weighted k-shortest path algoritme zeer robuust is. Het presteert beter dan de bestaande algoritmen op een variëteit van trajectory samples van uiteenlopend type.

Ten slotte willen we opmerken dat de twee onderwerpen van dit proefschrift (map matching en double-cross gelijkenis van polylijnen) beide deel uitmaken van het bredere gebied van *knowledge discovery in trajectory data*. Map matching kan aanzien worden als een data-cleaning stap die voorafgaat aan het toepassen van data mining algoritmen. Map matching algoritmen verwijderen meetfouten uit de GPS-metingen door de gemeten punten op het wegennetwerk af te beelden waar de beweging echt plaatsgevonden heeft. Voor vele data mining technieken, zoals clustering, die toegepast worden op gecleande data-verzamelingen is de studie en ontwikkeling van afstandsfuncties op trajectgegevens noodzakelijk. De double-cross gelijkenis van polylijnen (die van trajecten afkomstig zijn) past hier in het knowledge discovery proces. Zowel kwantitatieve als kwalitatieve afstandsmaten kunnen bij het clusteren van trajectgegevens gebruikt worden.

Publications by Bart Moelans

Chapters in books

- Leticia I. Gómez, Bart Kuijpers, Bart Moelans, Alejandro A. Vaisman. A State-of-the-Art in Spatio-Temporal Data Warehousing, OLAP and Mining. in *Integrations of Data Warehousing, Data Mining and Database Technologies*, 200–236, IGI-Global, 2011.
- Leticia I. Gómez, Bart Kuijpers, Bart Moelans, Alejandro A. Vaisman. A Survey on Spatio-Temporal Data Warehousing. In *Business Information Systems: Concepts, Methodologies, Tools and Applications*, Volume 4, 949–977, IGI-Global, 2010.
- Chiara Renso, Simone Puntoni, Elias Frentzos, Andrea Mazzoni, Bart Moelans, Nikos Pelekis, Fabrizio Pini. Wireless Network Data Sources: Tracking and Synthesizing Trajectories. In *Mobility, Data Mining and Privacy*, Fosca Giannotti and Dino Pedreschi (Eds.), 73–100, Springer, 2008.
- Dino Pedreschi, Francesco Bonchi, Franco Turini, Vassilios S. Verykios, Maurizio Atzori, Bradley Malin, Bart Moelans, Yücel Saygin. Privacy Protection: Regulations and Technologies, Opportunities and Threats. In *Mobility, Data Mining and Privacy*, Fosca Giannotti and Dino Pedreschi (Eds.), 101–119, Springer, 2008.

Journal papers

- Bart Kuijpers and Bart Moelans. On the realisability of double-cross matrices by polylines in the plane. (37 pages), submitted, 2015.
- Bart Kuijpers and Bart Moelans. Algebraic and geometric characterizations of double-cross matrices of polylines. (26 pages), submitted, 2015.

- Bart Kuijpers, Bart Moelans, Walied Othman and Alejandro Vaisman. An uncertainty-based map matching algorithm. Manuscript, 2015.

- Leticia I. Gómez, Bart Kuijpers, Bart Moelans, Alejandro A. Vaisman. A Survey of Spatio-Temporal Data Warehousing. In *International Journal of Data Warehousing and Mining*, Volume 5, Issue 3, 28–55, IGI-Global, 2009.

Conference and workshop papers

- Kristof Ghys, Bart Kuijpers, Bart Moelans, Walied Othman, Dries Vangoidsenhoven, Alejandro A. Vaisman. Map matching and uncertainty: an algorithm and real-world experiments. In *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM GIS 2009)*, 468–471, ACM, 2009.

- Bart Kuijpers, Bart Moelans, Walied Othman, Alejandro A. Vaisman. Analyzing Trajectories Using Uncertainty and Background Information. In *Proceedings of the 11th International Symposium on Spatial and Temporal Databases (SSTD 2009)*, 135–152, LNCS, Springer, 2009.

- Bart Kuijpers, Bart Moelans: Towards a geometric interpretation of double-cross matrix-based similarity of polylines. In *Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM GIS 2008)*, paper 32, ACM, 2008.

- Arend Ligtenberg, Ramona van Marwijk, Bart Moelans and Bart Kuijpers. Recognizing patterns of movements in visitor flows in nature areas. In *Proceedings of the 4th International Conference on Monitoring and Management of Visitor Flows in Recreational and Protected Areas (MMV4)*, 422–427, 2008.

- Vania Bogorny, Bart Moelans, and Luis Otavio Alvares. Filtering Frequent Spatial Patterns with Qualitative Spatial Reasoning. In *Proceedings of 19th Belgian-Dutch Conference on Artificial Intelligence (BNAIC 2007)*, 319–320, 2007.

- Luis Otávio Alvares, Vania Bogorny, Bart Kuijpers, José Antônio Fernandes de Macêdo, Bart Moelans, and Alejandro Vaisman. A Model for Enriching Trajectories with Semantic Geographical Information. in *Proceedings of the ACM 15th International Symposium on Advances in Geographic Information Systems (ACM GIS 2007)*, paper 22, ACM, 2007.

- Luis Otávio Alvares, Vania Bogorny, José Antônio Fernandes de Macêdo, Bart Moelans, and Stefano Spaccapietra. Dynamic Modeling of Trajectory Patterns using Data Mining and Reverse Engineering. In *Proceedings of the 28th International Conference on Conceptual Modeling (ER 2007), Tutorials, Posters, Panels and Industrial Contributions*, 149–154, Australian Computer Society, 2007.

- Vania Bogorny, Bart Moelans, Luis Otávio Alvares. Filtering Frequent Spatial Patterns with Qualitative Spatial Reasoning. In *Workshop Proceedings of the IEEE 23rd International Conference on Data Engineering (ICDE 2007) – Workshop on Spatio-Temporal Data Mining (STDM 2007)*, 527–535, IEEE, 2007. 2006

- Bart Kuijpers, Bart Moelans, Nico Van de Weghe. Qualitative polyline similarity testing with applications to query-by-sketch, indexing and classification. In *Proceedings of the 14th ACM International Symposium on Geographic Information Systems (ACM GIS 2006)*, 11–18, ACM, 2006.

Posters

- Luis Otávio Alvares, Vania Bogorny, José Antônio Fernandes de Macêdo, Bart Moelans, Stefano Spaccapietra: Dynamic Modeling of Trajectory Patterns using Data Mining and Reverse Engineering. ER (Tutorials, Posters, Panels & Industrial Contributions) 2007: 149-154

Others

- Bart Kuijpers, Vanessa Lemmens, Bart Moelans, Karl Tuyls: Privacy Preserving ID3 over Horizontally, Vertically and Grid Partitioned Data. CoRR abs/0803.1555 (2008)

- Vania Bogorny, Luis Otávio Alvares, Bart Kuijpers, José Antônio Fernandes de Macêdo, Bart Moelans and Andrey Tietbohl Palma. Towards Semantic Trajectory Knowledge Discovery. Technical Report, UHasselt, 2007.