Qualitative Polyline Similarity Testing with Applications to Query-by-Sketch, Indexing and Classification

Bart Kuijpers
Theoretical Computer Science
Hasselt University &
Transnational University of
Limburg, Belgium
bart.kuijpers@uhasselt.be

Bart Moelans
Theoretical Computer Science
Hasselt University &
Transnational University of
Limburg, Belgium
bart.moelans@uhasselt.be

Nico Van de Weghe
Geography Department
Ghent University
B-9000 Ghent, Belgium
nico.vandeweghe@ugent.be

ABSTRACT

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, the algorithm first computes their generalized polygons, 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 show that its sequential time complexity is bounded by $O\left((\frac{max(N_1,N_2)}{\varepsilon})^2\right)$, where N_1 and N_2 are the number of vertices of the given polylines. 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.

Categories and Subject Descriptors

H.2.1 [**Database Management**]: Logical Design; H.2.8 [**Database Management**]: Database Applications—*Spatial databases and GIS*

General Terms

Algorithms, Theory, Design

Keywords

Polygons, Polylines, Similarity, Double-Cross Calculus, Qualitative Calculi

1. INTRODUCTION AND SUMMARY

In several domains, such as computer vision, image analysis and GIScience, shape recognition and retrieval are central problems. Roughly speaking, there are two approaches to shape recognition and retrieval: the *quantitative* approach and the *qualitative* approach. Traditionally, most attention has gone to the quantitative methods [?, ?, ?, ?] and only recently, the qualitative approach has gained more attention, supported by cognitive studies that provide evidence that qualitative models of shape representation tend to be much more expressive than their quantitative counterpart [?]. It is widely accepted that shapes are one of the most complex phenomena that have been dealt with using a qualitative representation of space [?].

Within the qualitative approaches to describe two-dimensional shapes there are two major approaches: the region-based and the boundary-based approach. The region-based approach uses global properties such as circularity, eccentricity and axis orientation to describe shapes. As a result, this approach can only discriminate shapes with large dissimilarities [?]. The boundary-based approach describes the type and position of localized features (such as vertices, extremes of curvature and changes in curvature) along the polyline representing the shape [?]. Among the boundary-based approaches are [?, ?, ?, ?, ?, ?, ?, ?].

In this paper, we assume shapes to be described by their boundaries, which we assume to be polygons. As a generalization of polygons, we consider polylines and use this more general term in the remainder of the exposition, also to include polygons. At the basis of polyline recognition and retrieval lie algorithms to determine similarity between polylines. Here, we follow the qualitative formalism provided by the *double-cross calculus* [?, ?], which is used in the qualitative trajectory calculus for shapes [?], to test for polyline similarity. The double-cross matrix of a polyline contains for each pair of line segments in the polyline a 4-tuple of $-$, $+$, and 0 that describes the relative qualitative position of these line segments to each other. We remark that there are 81 possible such 4-tuples, but that only 65 of these 4-tuples are physically realizable.

For two given polylines, our method computes, if the polylines have equally many vertices, the double-cross matrices of the polylines and returns some distance value between these matrices as a measure of dissimilarity. A problem might be that the two given polylines of which we want to test similarity, may be given with a different number of vertices. To overcome this difference, our algorithm first computes for each of the two polylines a series of so-called *generalized polylines*, that consist of $2, 4, 8, ..., 2^n, ...$ line segments.

The nth generalized polyline has its vertices exactly at fractions $0, \frac{1}{2^n}, \frac{2}{2^n}, ..., \frac{2^n}{2^n}$ of the length of the original polyline. The generalized polylines have the property that they converge (also in length) to the given polylines and that they tend to consist of equally long line-segments. The latter property has the advantage that line segments analogously subdivided polylines, point to relatively analogous places on the polylines. The idea of using generalized polylines was introduced by two of the present authors in [?] and used in a naive way. In this paper, we optimize this approach and give a theoretical analyzis and complexity considerations of this idea.

Our algorithm for testing the similarity of two polylines, given by a sequence of vertices, works grosso modo as follows. For two given polylines, the algorithm computes their respective generalized polylines until their lengths approximate that of the given polylines sufficiently (this is formalized by an error-percentage ε that is part of the input). Once we have the generalized polylines, we compute their double-cross matrices and output the distance between these matrices. There are many possible ways to define a distance function between matrices. The one that we discuss in this paper, has shown to give the best results in practical experiments. To compute this distance, we first create for each of the two double-cross matrices a 65-ary vector of natural numbers that counts for each of the 65 realizable 4-tuples of $-$, $+$ and 0, the number of times they occur in the matrix. The distance-function then counts the average difference between the 65 count values.

We prove several properties of our algorithm for testing polyline similarity. The algorithm is guaranteed to terminate on any input and when it terminates the generalized polyline tends to consist of equally long line segments. It computes at most $\lfloor log\left(\frac{max(N_1,N_2)}{\varepsilon}\right)\rfloor + 1$ generalized polylines, where N_1 and N_2 are the number of vertices of the given polylines. We also show a theoretical upper bound on the overall sequential time complexity of our algorithm, which is $O\left((\frac{max(N_1,N_2)}{\varepsilon})^2\right)$. Although our algorithm contains large parts that are parallellizable, this only gives a gain by a factor of 2 compared to the sequential time complexity.

We apply our algorithm to three problems and give empirical test results on geographic data. Firstly, we consider the problem of similar polygons. We apply our implementation to various (polygonalized) sketches of a map of Belgium and it correctly orders the sketches in terms of increasing similarity.

Secondly, we take a look at the query-by-sketch problem, which allows a user to visually describe shapes or movements, in order to retrieve them from a database [?, ?]. It turns out that our algorithm performs well in classifying sketches of maps and also in retrieving geographic data given a query that is expressed as a two-dimensional sketch. In this experiment we also show the above mentioned 65-dimensional vector of count values can be used as an index on a database containing polygonal figures. We consider the problem of retrieving cities in Belgium by sketch and the output, which is a top 4 of best fitting cities, seems to correspond to what the human eye would recognize.

Thirdly, we apply our algorithm to polylines that represent terrain features (such as plateau, mesa, flat-floored valley, u-shape valley, depression and canyon) [?]. Experiments on fairly simple terrains give good results, but it seems that for more complex profiles, a division is needed according to the local maxima or minima of the terrain to obtain fits for the corresponding parts of the terrain.

Organization. This paper is organized as follows. In Section **??**, we define polylines and generalized polylines. Also the double-cross formalism is explained there. In Section **??**, the algorithm for testing polyline similarity is given and some basic properties are proven. We also give an analysis of its computational complexity. In Section **??**, we discuss the experiments with query-by-sketch, indexing and classification of terrain features.

2. DEFINITIONS AND PRELIMINARIES

In this section, we define polylines and generalized polylines and explain the double-cross formalism.

2.1 Polylines and polygonal shapes

Let $\mathbb{R}$ denote the set of real numbers and $\mathbb{R}^2$ the real plane.

Definition 1. A *polyline* P in $\mathbb{R}^2$ is a piecewise linear curve that is given by an ordered list of its *vertices*, i.e., $P = \langle(x_0, y_0), (x_1, y_1), ..., (x_N, y_N)\rangle$. A *polygonal shape* (or *polygon*) is a polyline $P = \langle(x_0, y_0), (x_1, y_1), ..., (x_N, y_N)\rangle$ for which $(x_0, y_0) = (x_N, y_N)$. $\square$

Let $P = \langle(x_0, y_0), (x_1, y_1), ..., (x_N, y_N)\rangle$ be a polyline. The vertices (x_0, y_0) and (x_N, y_N) are respectively the *start* and *end vertex* of P. We denote the line segment between (x_i, y_i) and (x_{i+1}, y_{i+1}) by $L_i(P)$ and its length by $\ell(L_i(P))$. We call $\ell(P) := \ell(L_0(P)) + \ell(L_1(P)) + \cdots + \ell(L_{N-1}(P))$ the *length of* P. The vector from (x_i, y_i) to (x_{i+1}, y_{i+1}) will be denoted $\overrightarrow{\ell_i}(P)$. If P is clear from the context, we will omit (P) from the above notations.

The *semantics of the polyline* $P = \langle(x_0, y_0), (x_1, y_1), ..., (x_N, y_N)\rangle$ is the subset of $\mathbb{R}^2$ consisting of all line segments between consecutive vertices of P, and we write

$$\mathsf{sem}(P) := \bigcup_{1 \leq i < N} L_i(P).$$

Often, when confusion is not possible, we will just talk about P when we mean $\mathsf{sem}(P)$.

We remark that the above definition does not exclude that three or more consecutive vertices may be collinear. In fact, different polylines, given by their vertices, may have the same semantics. We also remark that the line segments, appearing in the semantics, may intersect, as is illustrated in the polyline shown in Figure **??**. As geographic input data, we only consider polylines and polygons that are not self-intersecting, but polylines that occur in our algorithms may be self-intersecting. For this reason we allow them in the definition. For the sake of finite representability, we assume that the vertices have rational coordinates.

When we describe our algorithm to test polyline similarity, we will use, for a given polyline P, a number of so-called *generalized polylines* $P^2, P^4, P^8, ..., P^{2^n}, ...$ 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 $\lambda_P : [0, \ell(P)] \rightarrow \mathbb{R}^2$ that maps a length τ to the unique point of $L_k(P)$ at distance τ' from (x_k, y_k), where $\tau = \sum_{i=0}^{k-1} \ell(L_i(P)) + \tau'$ with $0 \leq \tau' < \ell(L_k(P))$ (with $k \geq 0$).

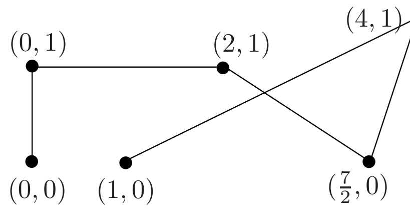


Figure 1: An example of a polyline: $P = \langle(0,0),(0,1),(2,1),(\frac{7}{2},0),(4,1),(1,0)\rangle$. Its semantics, drawn by means of the lines, is self-intersecting.

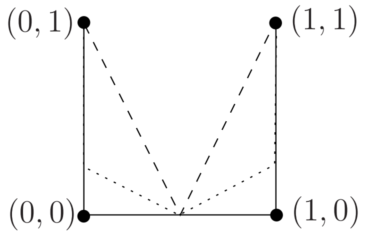


Figure 2: An example of a polyline $P = \langle(0,1),(0,0),(1,0),(1,1)\rangle$, drawn in full lines. Next, P^2 is shown in dashed lines and P^4 in dotted lines.

Definition 2. 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. 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, i.e., $(u_i,v_i) = \lambda_P(\frac{i}{2^n}\cdot\ell(P))$, for $i = 0,1,...,2^n$. $\square$

We remark that, in general, $\mathsf{sem}(P)$ and $\mathsf{sem}(P^{2^n})$ are *not* the same. They may even be different for arbitrary n. An example of this fact is given in Figure **??**. 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 $\frac{1}{3} \neq \frac{i}{2^n}$ for any n and any $0 \leq i \leq 2^n$.

2.2 The double-cross formalism

The double-cross calculus [**?**, **?**] is an expressive way of qualitatively representing a configuration of two vectors by means of a 4-tuple that expresses the orientation of both vectors with respect to each other. The double-cross formalism is, for instance, used in the "qualitative trajectory calculus for shapes" [**?**, **?**, **?**, **?**, **?**, **?**].

Above, we have associated to a polyline $P = \langle(x_0,y_0),(x_1, y_1),...,(x_N,y_N)\rangle$ the vectors $\overrightarrow{\ell_i}(P)$, representing oriented line segments between (x_{i-1},y_{i-1}) and (x_i,y_i), for $i = 1,..., N$.

We use the double-cross formalism to qualitatively present the orientation between $\overrightarrow{\ell_i}$ and $\overrightarrow{\ell_j}$ by means of a 4-tuple $(C_1\ C_2\ C_3\ C_4)$ from $\{-,0,+\}^4$ (traditionally this 4-tuple is written without commas). Consider two vectors $\overrightarrow{\ell_i}$ and $\overrightarrow{\ell_j}$. First of all, we define a "double cross" for these two vectors, determined by three lines:

- RL: the Reference Line connecting (x_{i-1},y_{i-1}) and (x_{j-1}, y_{j-1});
- PL_i: the Perpendicular Line on RL through (x_{i-1},y_{i-1});
- PL_j: the Perpendicular Line on RL through (x_{j-1},y_{j-1}).

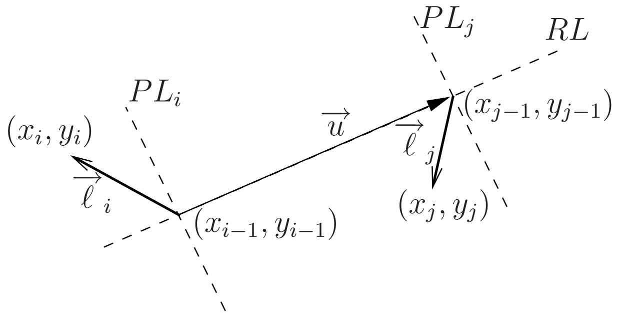


Figure 3: Double-cross design: the lines RL, PL_i, PL_j.

We refer to the vector between (x_{i-1},y_{i-1}) and (x_{j-1},y_{j-1}) as $\overrightarrow{u}$. The lines RL, PL_i, PL_j and the vector $\overrightarrow{u}$ are illustrated in Figure **??**.

Definition 3. For $\overrightarrow{\ell_i}, \overrightarrow{\ell_j}$ with $(x_{i-1},y_{i-1}) = (x_{j-1},y_{j-1})$, we define, for reasons of continuity [**?**], $DC(\overrightarrow{\ell_i},\overrightarrow{\ell_j}) = (C_1\ C_2\ C_3\ C_4) = (0\ 0\ 0\ 0)$. For $\overrightarrow{\ell_i}, \overrightarrow{\ell_j}$ with $(x_{i-1},y_{i-1}) \neq (x_{j-1},y_{j-1})$, we define the 4-tuple

$$DC(\overrightarrow{\ell_i},\overrightarrow{\ell_j}) = (C_1\ C_2\ C_3\ C_4)$$

as follows:

$C_1 = -$	*iff*	(x_i,y_i) lies on the same side of PL_i as (x_{j-1},y_{j-1}) and $(x_i,y_i) \notin PL_i$;
$C_1 = 0$	*iff*	$(x_i,y_i) \in PL_i$;
$C_1 = +$	*iff*	else;
$C_2 = -$	*iff*	(x_j,y_j) lies on the same side of PL_j as (x_{i-1},y_{i-1}) and $(x_j,y_j) \notin PL_j$;
$C_2 = 0$	*iff*	$(x_j,y_j) \in PL_j$;
$C_2 = +$	*iff*	else;
$C_3 = -$	*iff*	(x_i,y_i) lies on the left of $\overrightarrow{u}$;
$C_3 = 0$	*iff*	$(x_i,y_i) \in RL$;
$C_3 = +$	*iff*	else;
$C_4 = -$	*iff*	(x_j,y_j) lies on the right of $\overrightarrow{u}$;
$C_4 = 0$	*iff*	$(x_j,y_j) \in RL$; and
$C_4 = +$	*iff*	else. $\square$

For example, the 4-tuple for the vectors $\overrightarrow{\ell_i}$ and $\overrightarrow{\ell_j}$, shown in Figure **??**, is $(+ - - -)$.

In the algorithms below, we shall compute, for given $\overrightarrow{\ell_i}$ and $\overrightarrow{\ell_j}$, the value $DC(\overrightarrow{\ell_i},\overrightarrow{\ell_j}) = (C_1\ C_2\ C_3\ C_4)$ algebraically as is described in the following property. By sign : $\mathbb{R} \to \{-,0,+\}$ we denote the function that maps strictly negative numbers to $-$, 0 to 0 and strictly positive numbers to $+$.

Proposition 1. Let $\overrightarrow{\ell_i}$ and $\overrightarrow{\ell_j}$ have coordinates as described in Figure **??**. Then we have

$$\begin{aligned}
C_1 &= -\mathrm{sign}((x_{j-1}-x_{i-1})\cdot(x_i-x_{i-1}) \\
&\qquad +(y_{j-1}-y_{i-1})\cdot(y_i-y_{i-1})); \\
C_2 &= \mathrm{sign}((x_{j-1}-x_{i-1})\cdot(x_j-x_{j-1}) \\
&\qquad +(y_{j-1}-y_{i-1})\cdot(y_j-y_{j-1})); \\
C_3 &= -\mathrm{sign}((x_{j-1}-x_{i-1})\cdot(y_i-y_{i-1}) \\
&\qquad -(y_{j-1}-y_{i-1})\cdot(x_i-x_{i-1}));
\end{aligned}$$

and

$$\begin{aligned}
C_4 &= \mathrm{sign}((x_{j-1}-x_{i-1})\cdot(y_j-y_{j-1}) \\
&\qquad -(y_{j-1}-y_{i-1})\cdot(x_j-x_{j-1})).
\end{aligned}$$

PROOF. For $\overrightarrow{\ell_i} = \overrightarrow{\ell_j}$, it is clear that the four above formulas evaluate to zero. So, let's assume $\overrightarrow{\ell_i} \neq \overrightarrow{\ell_j}$. Let us consider the following vectors in $\mathbb{R}^3$: $\overrightarrow{u} = (x_{j-1} - x_{i-1}, y_{j-1} -$

$y_{i-1}, 0)$, $\overrightarrow{\ell'_i} = (x_i - x_{i-1}, y_i - y_{i-1}, 0)$ $\overrightarrow{\ell'_j} = (x_j - x_{j-1}, y_j - y_{j-1}, 0)$ and $\overrightarrow{w} = (0, 0, 1)$.

The well-known formula to calculate the angle θ between the two vectors $\overrightarrow{u}$ and $\overrightarrow{v}$ is $\cos(\theta) = \frac{\overrightarrow{u}.\overrightarrow{v}}{|\overrightarrow{u}|.|\overrightarrow{v}|}$ (the $\cdot$ denotes the scalar product of two vectors). So, we have $\cos(\theta) = 0$ if and only if $\overrightarrow{u}.\overrightarrow{v} = 0$ if and only if $\theta = \pm 90°$. So $\overrightarrow{u}.\overrightarrow{v} = 0$ means that $\overrightarrow{u}$ is perpendicular to $\overrightarrow{v}$. On the other hand, we have $cos(\theta) > 0$ when $\theta \in]-90°, 90°[$, so when $\overrightarrow{u}.\overrightarrow{v} > 0$. And finally $\overrightarrow{u}.\overrightarrow{v} < 0$ is equivalent to $\theta \in]90°, 270°[$. When we apply this reasoning to $\overrightarrow{v} = \overrightarrow{\ell'_i}$ and $\overrightarrow{v} = \overrightarrow{\ell'_j}$, we obtain the expressions for C_1 and C_2: $C_1 = -\text{sign}(\overrightarrow{u} \cdot \overrightarrow{\ell'_i})$ and $C_2 = -\text{sign}(-\overrightarrow{u} \cdot \overrightarrow{\ell'_j})$.

For C_3, we consider the block-product $(\overrightarrow{u} \times \overrightarrow{\ell'_i}) \cdot \overrightarrow{w}$. This product is zero if and only if $\overrightarrow{u}$ and $\overrightarrow{\ell_i}$ are collinear. The vectorial product $\overrightarrow{u} \times \overrightarrow{\ell'_i}$ is a vector pointing in the direction of positive z-coordinate if and only if $\overrightarrow{\ell'_i}$ is on the right of $\overrightarrow{u}$, which means that $(\overrightarrow{u} \times \overrightarrow{\ell'_j}) \cdot \overrightarrow{w} < 0$ in this case.

The case of C_4 is completely similar to that of C_3, after interchanging left and right. So, we have $C_3 = -\text{sign}((\overrightarrow{u} \times \overrightarrow{\ell'_i}) \cdot \overrightarrow{w})$ and $C_4 = \text{sign}((\overrightarrow{u} \times \overrightarrow{\ell'_j}) \cdot \overrightarrow{w})$. $\square$

The following property says how $DC(\overrightarrow{\ell_j}, \overrightarrow{\ell_i})$ can be derived from $DC(\overrightarrow{\ell_i}, \overrightarrow{\ell_j})$ in a straightforward way.

Proposition 2. If $DC(\overrightarrow{\ell_i}, \overrightarrow{\ell_j}) = (C_1\ C_2\ C_3\ C_4)$, then we have $DC(\overrightarrow{\ell_j}, \overrightarrow{\ell_i}) = (C_2\ C_1\ C_4\ C_3)$. $\square$

If we write $--$ for $+$; $-+$ for $-$ and -0 for 0, we easily obtain the following property.

Proposition 3. If $DC(\overrightarrow{\ell_i}, \overrightarrow{\ell_j}) = (C_1\ C_2\ C_3\ C_4)$ and $\sigma : \mathbb{R}^2 \to \mathbb{R}^2$ is a reflection along a line, then we have the equality $DC(\sigma(\overrightarrow{\ell_i}), \sigma(\overrightarrow{\ell_j})) = (C_1\ C_2\ -C_3\ -C_4)$. $\square$

2.3 The double-cross matrix *DCM*

Based on the double-cross formalism, we now define the double-cross matrix of a polyline.

Definition 4. A *double-cross matrix* $DCM(P)$ of a polyline $P = \langle (x_0, y_0), (x_1, y_1), ..., (x_N, y_N) \rangle$ is a $N \times N$ matrix with $DCM(P)[i, j] = DC(\overrightarrow{\ell_i}, \overrightarrow{\ell_j})$. $\square$

Because of Proposition **??** and because the diagonal entries of $DCM(P)$ are all $(0\ 0\ 0\ 0)$, it suffices to consider only the upper triangle of the matrix $DCM(P)$, i.e., the $\frac{N^2-N}{2}$ elements $DCM(P)[i, j]$ with $i < j$. The following property is easily verified.

Proposition 4. Let P be a polyline and let $\alpha : \mathbb{R}^2 \to \mathbb{R}^2$ be an orientation-preserving isometry (i.e., a composition of a rotation and translation) or a point-scaling (i.e., a mapping of the form $(x, y) \mapsto a \cdot (x, y)$ with $a \neq 0$, then $DCM(P) = DCM(\alpha(P))$. $\square$

We remark that the double-cross matrix is not invariant under scalings of the form $(x, y) \mapsto (a \cdot x, b \cdot y)$ with $a \neq b$. More specifically, C_1 and C_2 are not invariant under these scalings, but C_3 and C_4 are invariant.

3. POLYLINE SIMILARITY ALGORITHM

3.1 The algorithm DC-SIMILAR$_\Delta$

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

```
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 **??**). To define the distance measure $\Delta_H(M_1, M_2)$ between DoubleCross matrices, we first construct, for both matrices M_1 and M_2, vectors $\gamma(M_1), \gamma(M_2) \in \mathbb{N}^{65}$ 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. Then

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

The function Δ_H counts the average difference between the 65 count values.

3.2 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).

Proposition 5. 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 $\ell(P^{2^n}) < \ell(P^{2^{n+1}}) \leq \ell(P)$. 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 $|\ell(P) - \ell(P^{2^n})| < \varepsilon \cdot \ell(P)$. $\square$

Proposition 6. 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 $\sigma \leq \frac{\varepsilon}{2^n} \cdot \ell(P)$. $\square$

3.3 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 7. The algorithm DC-SIMILAR$_{\Delta_H}$, on input $P_1 = \langle(r_0, s_0), (r_1, s_1), ..., (r_{N_1}, s_{N_1})\rangle$, $P_2 = \langle(u_0, v_0), (u_1, v_1), ..., (u_{N_2}, v_{N_2})\rangle$ and $0 < \varepsilon \leq 1$, needs $O\left((\frac{max(N_1,N_2)}{\varepsilon})^2\right)$ sequential time to return its output.

PROOF. Let $P = \langle(x_0, y_0), (x_1, y_1), ..., (x_N, y_N)\rangle$ 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 $N.\ell(P)$ 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 scan each tuple of M_1 and M_2 in parallel once and needs maximum $O(\frac{M}{\varepsilon})$ sequential time. $\square$

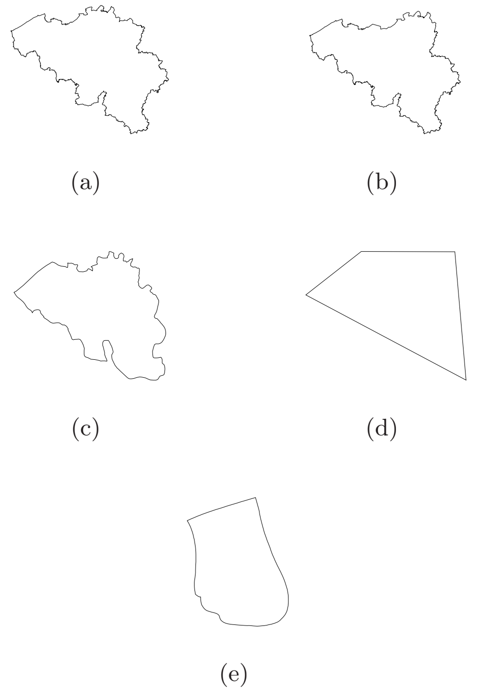


Figure 4: A map of Belgium and some sketches.

4. 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.

4.1 Experiment 1: Polygon similarity

For the first experiment, we used five figures representing Belgium (see Figure **??**). Figure **??** is the correct representation of Belgium, consisting of 2047 line segments. Figure **??** 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 **??**: real representation, Figure **??**: abstract representation) and a non-specialist (Figure **??**).

Figure	??	??	??	??	??
??	100%	99%	87%	66%	60%
??		100%	87%	67%	60%
??			100%	80%	69%
??				100%	84%
??					100%

Table 1: Similarity between polygons of Figure **??** using Δ_H with threshold 5%.

We applied the algorithm DC-SIMILAR$_\Delta$ for $\Delta = \Delta_H$ to each two of these five figures with a threshold $\varepsilon = 5\%$. The result are given in Table **??**. 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 **??**. Figure **??**, 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.

We remark that Δ_H , 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.

4.2 Experiment 2: Query by sketch

Because several experiments indicated that Δ_H gives good, start-vertex-independent results, we used 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.

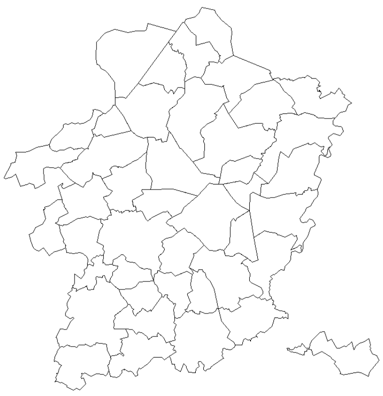


Figure 5: The 43 cities and villages of Limburg.

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%. 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 here 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 **??**. This province consists of 43 villages and cities and it is not connected. Building the index for this sample has costed 0.8 seconds on the above-mentioned configuration.

The first query is: "Give the 4 cities of Limburg looking the most as the bowler hat sketched on the left in Figure **??**." The resulting cities are colored dark gray in Figure **??**, with Lommel, the most similar city, shown in the north. We remark that Proposition **??** guarantees that similarity is tested invariantly 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 was: "Give the 4 cities of Limburg looking the most as the bow tie sketched on the right in Figure **??**." The resulting cities are colored light gray in Figure **??**. The best fit is the village at the bottom right.

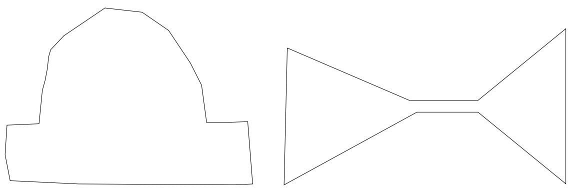


Figure 6: Bowler hat and bow tie sketch.

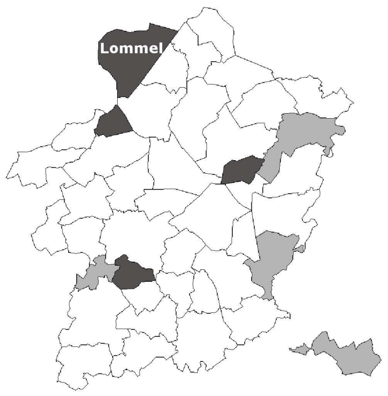


Figure 7: Result of bowler hat query (dark gray) and bow tie query (light gray).

4.3 Experiment 3: Classification of terrain features

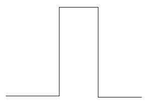
Our last experiment deals with *terrain features.*The figures in this experiment are inspired by work of Kulik and Egenhofer [?]. We used the seven figures in Figure **??** as primitive terrain features to classify some silhouettes of terrains.

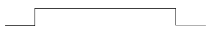
We use our Δ_H measure to classify the figures in Figure **??**.

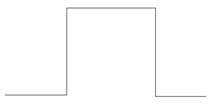
Our Δ_H algorithm classifies the three silhouettes of Figure **??**, as (a) Mesa, (b) U-Valley and (c) Mesa (or U Valley) respectively (see Tables **??**,**??** and **??**). For (a) and (b) this corresponds to our visual observations. Figure **??** (c) is more complicated and needs more attention. We divided this figure according to its local maxima and minima (see Figure **??**). The resulting classifications, summarized in Tables **??** and **??**, give a more precise description of these terrains.

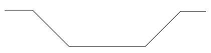
	Fig. **??**	Fig. **??**	Fig. **??**
Butte	58%	52%	58%
Plateau	62%	62%	64%
Mesa	**71%**	64%	**70%**
Flat Valley	48%	56%	50%
U Valley	63%	**77%**	**68%**
Depression	47%	49%	50%
Canyon	42%	50%	43%

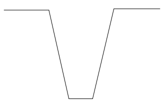
Table 2: Classification by Δ_H of Figure **??** using the primitives sketched in Figure **??**.

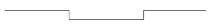

(a) Butte


(b) Plateau


(c) Mesa


(d) Flat-floored Valley


(e) U-shape Valley


(f) Depression

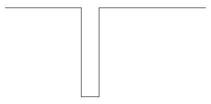

(g) Canyon

Figure 8: Basic shapes of terrain features.

	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 3: Classification by Δ_H of Figure **??** using the primitives sketched in Figure **??**.

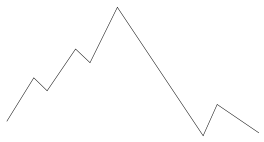
5. DISCUSSION

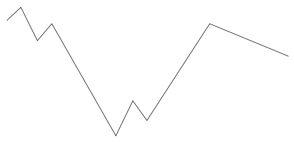
We have given a number of basic properties and described the time complexity of the algorithm DC-SIMILAR$_\Delta$ for testing polyline similarity. This algorithm depends on a function Δ that measures the difference between double-cross matrices. We have experimented with a number of Δ's and Δ_H, used 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 $\{-,0,+\}$: $\delta(-,-) := \delta(+,+) := \delta(0,0) := 0$, $\delta(-,0) := \delta(+,0) := 1$, $\delta(-,+) := 2$ 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

$$\Delta_E(DCM(P_1), DCM(P_2)) :=$$


(a)


(b)

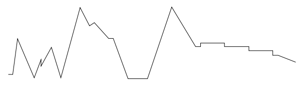

(c)

Figure 9: Some silhouettes of terrains.

	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: Classification by Δ_H of Figure **??** using the primitives sketched in Figure **??**.

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

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.

6. ACKNOWLEDGMENTS

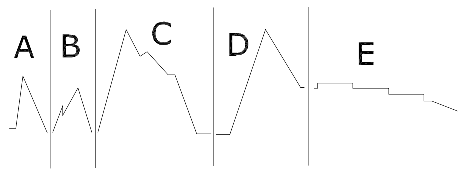
This research has been partially funded by the European Union under the FP6-IST-FET programme, Project n. FP6-14915, GeoPKDD: Geographic Privacy-Aware Knowledege Discovery and Delivery", (www.geopkdd.eu) and by the Research Foundation Flanders (FWO-Vlaanderen), Research Project G.0344.05.

7. REFERENCES

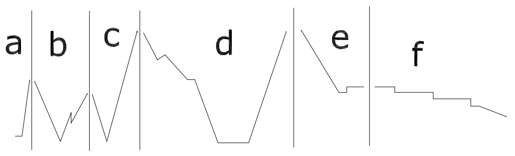
[1] F.L. Bookstein. Size and shape spaces for landmark data in two dimensions. *Statistical Science*, 1:181–242, 1986.

[2] N. Van de Weghe. *Representing and Reasoning about Moving Objects: A Qualitative Approach.* PhD thesis, Ghent University (Belgium), 2004.

[3] N. Van de Weghe, A. G. Cohn, and Ph. De Maeyer. A qualitative representation of trajectory pairs. In



(a)



(b)

Figure 10: Figure **??** divided by his maxima (left) and minima (right).

R. López de Mántaras and L. Saitta, editors, *Proceedings of the 16th Eureopean Conference on Artificial Intelligence (ECAI'04)*, pages 1103–1104, 2004.

[4] N. Van de Weghe, A.G. Cohn, G. de Tré, and Ph. De Maeyer. A qualitative trajectory calculus as a basis for representing moving objects in geographical information systems. To appear in *Control and Cybernetics*, 2006.

[5] N Van de Weghe, A.G. Cohn, Ph. De Maeyer, and F. Witlox. Representing moving objects in computer based expert systems: the overtake event example. *Expert Systems with Applications*, 29(4):977–983, 2005.

[6] N. Van de Weghe and Ph. De Maeyer. Conceptual neighbourhood diagrams for representing moving objects. In J. Akoka et al., editor, *ER (Workshops)*, volume 3770 of *Lecture Notes in Computer Science*, pages 228–238, 2005.

[7] N. Van de Weghe, G. De Tré, B. Kuijpers, and Ph. De Maeyer. The double-cross and the generalization concept as a basis for representing and comparing shapes of polylines. In R. Meersman et al., editor, *Proceedings of the 1st International Workshop on Semantic-based Geographical Information Systems (SeBGIS'05)*, volume 3762 of *Lecture Notes in Computer Science*, pages 1087–1096. Springer, 2005.

[8] I. Dryden and K.V. Mardia. *Statistical Shape Analysis*. Wiley, 1998.

[9] M. Egenhofer. Query Processing in Spatial-Query-by-Sketch. *Journal of Visual Languages and Computing*, 8(4):403–424, 1997.

[10] M. Erwig and M. Schneider. A visual language for the evolution of spatial relationships and its translation into a spatio-temporal calculus. *Journal of Visual Languages and Computing*, 14(2):181–211, 2003.

[11] K. D. Forbus. Qualitative physics: Past, present, and future. In D. S. Weld and J. de Kleer, editors, *Readings in Qualitative Reasoning about Physical Systems*, pages 11–39. Kaufmann, San Mateo, California, 1990.

[12] Christian Freksa. Using orientation information for qualitative spatial reasoning. In A. Frank et al., editor, *Spatio-Temporal Reasoning*, volume 639 of *Lecture Notes in Computer Science*, pages 162–178. Springer, 1992.

[13] J.S. Gero. Representation and reasoning about shapes: cognitive and computational studies in visual reasoning in design. In *Proceedings of the International Conference on Spatial Information Theory (COSIT'99)*, pages 315–330, 1999.

[14] B. Gottfried. Tripartite line tracks qualitative curvature information. In Kuhn et al., editor, *Proceedings of the International Conference on Spatial Information Theory (COSIT'03)*, volume 2825 of *Lecture Notes in Computer Science*, pages 101–117. Springer, 2003.

[15] E. Jungert. Symbolic spatial reasoning on object shapes for qualitative matching. In *Proceedings of the International Conference on Spatial Information Theory (COSIT'93)*, pages 444–462, 1993.

[16] J.T. Kent and K.V. Mardia. Shape, procrustes tangent projections and bilateral symmetry. *Biometrika*, 88:469–485, 2001.

[17] L. Kulik and M. Egenhofer. Linearized terrain: Languages for silhouette representations. In Kuhn et al., editor, *Proceedings of the International Conference on Spatial Information Theory (COSIT'03)*, volume 2825 of *Lecture Notes in Computer Science*, pages 118–135. Springer, 2003.

[18] L.J. Latecki and R. Lakämper. Shape similarity measure based on correspondence of visual parts. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(10):1185–1190, 2000.

[19] M. Leyton. A process-grammar for shape. *Artif. Intell.*, 34(2):213–247, 1988.

[20] R.C. Meathrel. *A General Theory of Boundary-Based Qualitative Representation of 2D Shape*. Phd thesis, University of Exeter (UK), 2001.

[21] F. Mokhtarian and A. K. Mackworth. A theory of multiscale, curvature-based shape representation for planar curves. *TPAMI*, 14:789–805, 1992.

[22] Chr. Schlieder. *Geographic Objects with Indeterminate Boundaries*, chapter Qualitative shape representation, pages 123–140. Taylor & Francis, 1996.

[23] K. Zimmermann and Chr. Freksa. Qualitative spatial reasoning using orientation, distance, and path knowledge. *Appl. Intell.*, 6(1):49–58, 1996.