A study of quantitative and qualitative methods for trajectories

Jelle Van Hoof

promotor :
Prof. dr. Bart KUIJPERS

Eindverhandeling voorgedragen tot het bekomen van de graad
Master in de informatica databases

A study of Quantitative and qualitative methods for trajectories

Jelle Van Hoof

Promotor:
Prof. dr. Bart Kuijpers

Begeleiders:
Bart Moelans Walied Othman

Thesis voorgedragen tot het behalen van de graad
van master in de informatica.

Universiteit Hasselt - transnationale Universiteit Limburg

Academiejaar 2006–2007

Abstract

In this work, we discuss two qualitative and two quantitative methods for similarity search for trajectories in $\mathbb{R}^2 \times \mathbb{T}$. The two qualitative methods that are discussed are the Double–Cross and the Twisted–Cross and the quantitative methods are the Fréchet Distance and the Hausdorff Distance.

Distance measures are very important in computer science, since they are, for example, used for similarity search, query by example and clustering.

Before we start with this study, we will first take a look at the objects we like to compare: trajectories in $\mathbb{R}^2 \times \mathbb{T}$. We will define and discuss them and give a definition and an algorithm to generalize these trajectories.

Then we will give an explanation on how these different methods work, their differences and their similarities and their use in similarity search.

The focus in this work, however, will be on the Fréchet–based Distance for trajectories: a new distance measure, inspired by the Fréchet Distance. We will give an algorithm for this new distance measure and we will prove that it stops and that it has an optimal running time.

In the end, we will use the algorithm for the Fréchet–based Distance to calculate similarities between different trajectories and we will use it for clustering. All these tests are performed with our own program, designed to test these algorithms, named Algorithmtester. We will see that the algorithm returns excellent results with very little computational time needed.

Acknowledgements

A work like this one, is not one that just pops into existence. It requires a lot of time, a lot of work and a lot of support. When any of these three ingredients is missing, it is very difficult to complete a decent work.

I will not go into detail about the time or the work, but I would like to thank those people who has helped me with the creation of this work, whether it be by just believing in me or guiding me and helping me to avoid many of the pitfalls that exist.

In the first group, I like to thank my parents, in the first place for giving me the opportunity to study. Without them I would never have been able to start this work in the first place. I would also like to thank my girlfriend, for being there when I needed her.

Second, I like to thank the people in the last group: Prof. dr. Bart Kuijpers, Bart Moelans and Walied Othman, who helped me, supported me, aided me, agreed and disagreed with me and gave me a chance to work on the interesting subject of this thesis.

I like to end with thanking all those people, not mentioned here, who helped with little things and who gave me little tips and such. Your input, how small it may have been, was surely appreciated.

Dutch summary Nederlandse samenvatting

Inleiding

GSM. PDA. GPS. De tijd staat niet stil en elke dag komen er meer en meer van deze toestellen in omloop, maar niet alleen het aantal toestellen stijgt, ook de hoeveelheid data gegenereed door deze toestellen neemt continu toe.

Deze data kan gebruikt worden om nieuwe kennis te vinden, kennis met een economische of sociale impact. Maar om deze kennis te vinden in de enorme massa rauwe data, hebben we technieken nodig, technieken die rekening houden met de privacy van de bron van de data.

Het Europese GeoPKDD project [geo] – *Geographic Privacy-aware Knowledge Discovery and Delivery* – is een project dat zich als doel heeft gesteld om dergelijke methodes te vinden die zoveel mogelijk kennis kunnen halen uit verzamelde data waarbij de privacy van de bron van deze data ten alle tijde gerespecteerd wordt. Hierbij wordt gebruik gemaakt van datamining technieken zoals similarity search.

Deze technieken worden in dit project aangewend in het GIS – *Geographic Information Science* – domein, maar ze kunnen net zo goed in andere domeinen toegepast worden. Voorbeelden van het gebruik van deze technieken in andere domeinen, kan men terugvinden in Hoofdstuk 1 en meerbepaald in Figuur 1.1 en Figuur 1.2.

In dit werk concentreren we ons op similarity search zoals het gebruikt wordt in het vermelde project. Meer specifiek, nemen we een kijkje naar twee grote groepen van methodes: *kwalitatieve* en *kwantitatieve* methodes.

Op de verschillen tussen deze twee groepen van methodes komen we later terug, maar beide methodes maken gebruik van een bepaalde notie van afstand: een afstandsmaat. Afstandsmaten zijn niet alleen belangrijk en nuttig voor similarity search, maar worden o.a. ook gebruikt om bij *clustering* en *query by example.*

De focus in dit werk ligt op deze afstandsmaten en het maakt het daarom ook perfect mogelijk om dit werk te bekijken vanuit een andere invalshoek dan louter en alleen similarity search.

Trajecten

Voor we deze methodes kunnen gebruiken, moeten we eerst het type data definiëren waarop we deze willen toepassen. In deze thesis werken we met *trajecten.* Een traject is een polylijn waarbij elke vertex voorzien is van een tijdstip.

Dit tijdstip kan eender welke vorm aannemen, maar we kunnen elk tijdstip steeds mappen naar een waarde in $\mathbb{R}$. Een precieze definitie kan men terugvinden in hoofstuk 2, Definitie 1.

Het generalizatie principe en kwalitatieve methodes

Het generalizatie principe

Indien we met deze trajecten willen werken en similarity search toepassen op deze trajecten, zijn er enkele problemen die kunnen optreden die we gelukkig vrij makkelijk kunnen oplossen met een generalizatie.

- Bepaalde afstandsfuncties toegepast op de kwalitatieve voorstelling van trajecten aan de hand van een Double–Cross of een Twisted–Cross, vereisen dat de te vergelijken trajecten evenveel vertices hebben.
- Generalizeren normaliseert de trajecten.
- Het voorgestelde concept maakt het mogelijk om onze technieken ook op trajecten met curves toe te passen, zoals getoond in Figuur 3.1.
- Het lost de problemen op die kunnen opduiken bij een Boundary–Based benadering, geïllustreerd in Figuur 3.2.

Een precieze definitie kan gevonden worden in Sectie 3.1 en het algoritme in Listing 3.1. Het principe was voorgesteld in [VDKD05], maar de code zoals die in dit werk staat komt uit [KM06].

In [KM06], was het bewijs gegeven dat dit algoritme stopt op eender welke input, maar het bewijs bevatte een fout. We hebben deze fout uit het bewijs gehaald en het bewijs overeenkomstig aangepast. Details staan in Sectie 3.1.2.

Kwalitatieve methodes

Kwalitatieve methodes [VDKD05, KMV06, KM06] voor similarity search zijn nog erg jong in vergelijking met kwantitative methodes [AG95, HKR93] en het aantal publicaties hierover is nog vrij klein. We beperken ons in dit werk dan ook tot een kortere inleiding in dit principe.

Kwalitatieve methodes maken geen gebruik van de absolute posities van de te vergelijken objecten, maar van de onderlinge relatieve posities van de segmenten van een object. Dit zorgt er o.a. voor dat deze technieken ongevoelig zijn voor transformaties zoals translaties, rotaties en schaling [KM06].

Een nadeel van deze techniek is dat het bijna onmogelijk is om, gegeven een kwalitatieve representatie van een traject, het originele traject terug te vinden. Maar dit kan ook als een voordeel gezien worden, omdat dit meer bescherming biedt voor de privacy van de oorspronkelijke data.

De twee methodes die in dit werk besproken worden zijn de *Double–Cross Methode*, besproken in Sectie 3.2.1 en de *Twisted–Cross Methode*, besproken in Sectie 3.2.2, beide methodes stellen een traject voor als een matrix, waarbij elke cel een code bevat die de relatieve positie tussen twee vertices in dit traject voorstelt.

Het verschil tussen de twee methodes zit voornamelijk in de manier waarop de codes gevonden worden. Hoewel beide methodes een matrix opleveren, is het onzin om een matrix bekomen door de ene methode te vergelijken met een matrix van de andere methode.

Kwantitatieve methodes

Kwantitatieve methodes voor similarity search, maken gebruik van gekende lengtematen om zo de afstand, oftewel het verschil tussen twee objecten uit te drukken.

In dit werk bespreken we twee van de meer bekender methodes: *Hausdorff afstand* in Sectie 4.1 en *Fréchet afstand* in Sectie 4.2. Maar het zwaartepunt van dit werk ligt op een nieuw gevonden afstandsmaat, gebaseerd op de Fréchet afstand: de *Fréchet–based afstand voor trajecten.*

De Hausdorff afstand [Bla] is een kwantitatieve afstandsmaat waar al heel wat onderzoek naar verricht is in het domein van similarity search. Deze afstandsmaat wordt gebruikt vanwege zijn snelheid en de goede resultaten die deze oplevert in de meeste gevallen.

De definitie kan teruggevonden worden in Sectie 4.1.1, Definitie 7. Deze afstandsmaat is een metriek voor elke gesloten ruimte [Hen99] en een pseudo–metriek voor elke niet–gesloten ruimte. Dit laatste wordt aangetoond in Figuur 4.1.

Gebaseerd op Definitie 7, hebben we een algoritme gevonden dat de assymmetrische Hausdorff afstand berekent voor onze trajecten in $\mathbb{R}^2 \times \mathbb{T}$. Dit algoritme kan teruggevonden worden in Listing 4.1. Dieper zijn we in dit werk niet ingegaan op de afstandsmaat, door de uitwerking van de uiterst interessante Fréchet–based afstand.

Deze Fréchet–based afstand is gebaseerd op het principe de gekende Fréchet afstand [God98] waarvan de definitie staat in Sectie 4.2.1, Definitie 9. Een eenvoudigere omschrijving van deze afstandsmaat wordt meestal als volgt gegeven: neem een man en zijn hond die aan het wandelen zijn. Beiden volgen een eigen route en de Fréchet afstand zoekt de kortst mogelijke halsband die nodig is tussen de man en zijn hond tijdens deze wandeling.

In de praktijk zoekt deze afstandsmaat naar een ideale tijdsparametrisatie om deze afstand te vinden, maar dit is een zeer tijdrovende operatie [AG95, AHK$^+$06]. Onze trajecten zijn al voorzien van een tijdswaarde: elke vertex bevat immers een tijdstip.

Als we deze bestaande tijdswaardes zouden gebruiken om onze kortste leiband te vinden, dan kunnen we veel tijd besparen doordat we geen tijdsparametrizatie hoeven te zoeken. Hoewel deze bestaande tijdsfunctie niet de ideale is, levert ze wel uitstekende testresultaten op, wat we later zullen bespreken, in Hoofdstuk 5.

Deze nieuwe afstandsmaat wordt voorgesteld in Sectie 4.2.2, waar we meteen een algoritme terugvinden in Listing 4.2. Dit algoritme werkt uiterst snel: de complexiteit van dit algoritme is in de orde van $O(n + m)$, wat veel sneller is dan het klassieke, algemene algoritme voor de Fréchet afstand [AG95]. Het is zelfs geweten dat, voor objecten in een ruimte met een dimensie van 4 of groter, dit een NP-hard probleem is [God98].

De bewijzen dat ons nieuwe algoritme stopt, zijn complexiteit en het optimaal zijn van deze complexiteit, staan in detail in Sectie 4.2.5. We zijn ook nagegaan of deze nieuwe afstandsmaat ook een metriek is, maar hoewel de niet-negativiteit, scheidingseigenschap gelden, geldt symmetrie alleen indien de twee trajecten een gelijk aantal vertices hebben en geldt driehoeksongelijkheid niet. Bewijzen hiervan staan gedetailleerd uitgelegd in Sectie 4.2.7.

Similarity search

Similarity search is de zoektocht naar gelijkaardigheid. Dit kan, bijvoorbeeld, het zoeken zijn naar twee objecten of figuren die erg hard op elkaar lijken of het zoeken naar bepaalde patronen in complexe figuren. De mogelijkheden zijn legio en similarity search ken dan ook toepassingen in verschillende domeinen, van de medische wereld [Kel06, NCV$^+$03] tot het domein van GIS [geo, BPSW05].

Wij beperken ons tot similarity search zoals het gebruikt wordt binnen het GeoPKDD project: om te zoeken naar trajecten die erg op elkaar gelijken en om clusters van interessante trajecten te vinden.

Similarity search met kwalitatieve methodes

Kwalitatieve methodes komen in dit werk alleen als illustratie naar voren en zijn niet in detail besproken. Ook in dit stuk van het werk wordt er niet te diep op ingegaan, maar geven we slechts een beschrijving van het principe.

Zowel met de Double–Cross methode als met de Twisted–Cross methode, wordt er van elk traject een matrix aangemaakt, waarbij elke cel de relatieve positie tussen twee vertices van dit traject bevat. Met behulp van deze matrices, kan men dan een graad van gelijkaardigheid tussen twee trajecten bepalen. Meer details en de precieze formule kan men in Sectie 5.1 vinden.

Similarity search met de Hausdorff afstand

De Hausdorff afstand wordt meestal gebruikt om te zoeken naar specifieke figuren binnen een complex patroon [HKR93], bijvoorbeeld naar kleine cirkels in een grotere afbeelding.

Hiervoor is het toegestaan om de te vergelijken figuren te roteren of schalen om de afstand te minimaliseren. Formeel kan men zeggen dat similarity search met deze afstandsmaat op de volgende manier gebeurt: Bepaal een translatie, rotatie of schaling of een combinatie, die de Hausdorff afstand tussen de te vergelijken objecten minimaliseert [HKR93].

Indien de dan gevonden afstand kleiner is dan een gegeven drempelwaarde, worden de objecten als gelijkaardig beschouwd. In Figuur 5.2 vinden we een voorbeeld van een succesvol gebruik van de Hausdorff afstand. Helaas houdt deze afstand geen rekening met de oriëntatie van de objecten en indien we naar Figuur 5.1 kijken, zien we twee figuren die een kleine Hausdorff afstand hebben, hoewel ze verre van gelijkaardig zijn.

Om dit probleem te vermijden, wordt er vaak gebruik gemaakt van de Fréchet afstand.

Similarity search met de Fréchet afstand

Similarity search met de Fréchet afstand is erg gelijkend op deze met de Hausdorff afstand. In het algemeen wordt similarity search met deze afstandsmaat beschouwd als een beslissingsprobleem: is de afstand tussen twee objecten kleiner dan een gegeven drempelwaarde of niet? [AKW01]

Helaas is het algoritme om de Fréchet afstand te bereken niet bruikbaar in de praktijk [AG95] en wordt er meestal gebruik gemaakt van algoritmes of definities, die specifiek zijn aangepast aan het domein van het probleem [AHK$^+$06].

Similarity search met de Fréchet–based afstand

Om onze nieuwe afstandsmaat te testen volgen we het het idee dat gebruikt wordt voor alle Fréchet gebaseerde afstandsmaten: we zoeken of de afstand tussen twee objecten kleiner is dan een gegeven drempelwaarde.

Onze eerste test bestond uit het clusteren van een verzameling trajecten. Deze trajecten zijn gevisualiseerd in Figuur 5.3. De testresulaten zijn erg afhankelijk van het gebruikte systeem, dus willen we er even op wijzen dat alle testen zijn gebeurd met een systeem met een Intel© Centrino© Duo 1.86 GHz processor en 2GB RAM met Microsoft Windows™XP Professional als besturingssysteem.

Als testprogramma wordt er gebruik gemaakt van *Algorithmtester*, een programma dat we speciaal voor deze testen ontworpen en geïmplementeerd hebben met als clusteralgoritme een k–means algoritme [HK01]. Meer details over dit programma staan in Appendix B.2.

Resultaten van het clusteren kunnen teruggevonden worden in de Figuren 5.4 en 5.5. Het is duidelijk te zien dat het algoritme de verwachte clusters ook terugvindt en dat dit uiterst snel gebeurd: respectievelijk in gemiddeld 375ms voor 5 clusters en 406.5ms voor 6 clusters.

Om meer rechtstreeks de gelijkaardigheid tussen twee trajecten te bepalen, hebben we code toegevoegd om te zoeken naar een goede combinatie van translaties, rotaties en schaling zodat de figuren zo goed mogelijk passen. De exacte werking van dit algoritme staat, met illustratie, in Sectie 5.4.3.

Onze code is niet de “ideale” oplossing, maar wel een erg snelle oplossing. Het vinden van de “ideale” translatie is immers geen sinecure: in [AKW01] wordt een techniek voorgesteld om deze te vinden, maar deze houdt geen rekening met de mogelijkheid van rotaties of schaling.

De complexiteit van dit voorstel is $O\left((mn)^3(m+n)^2\right)$, terwijl ons voorstel slechts een enkele translatie en rotatie nodig heeft. Het grootste nadeel is dat onze eigen methode niet de perfecte transformaties gebruikt, zoals aangetoond in Figuur 5.8, maar het resultaat is goed genoeg voor onze doeleinden en door de grote snelheid heeft het zeker een uitstekend praktisch nut.

Toekomstig onderzoek

Hoewel we reeds verschillende vragen over de Fréchet–based afstand beantwoord hebben, blijven er veel vragen onbeantwoord. Welke methode is de beste? Wanneer is welke methode de beste en waarom? Is er een verband tussen de methodes? Deze en nog andere vragen blijven voorlopig onbeantwoord. Zoals we kunnen zien in Figuur 6.1, vermoeden we dat er een verband bestaat tussen onze nieuwe afstandsmaat en de kwalitatie methodes. Een specifiek onderzoek van dit vermoeden, ligt echter buiten dit werk.

Contents

Chapter 1

Introduction

Every day, more and more data pertinent to moving objects, is collected from mobile phones, GPS–devices and other location–aware devices. This increasing collection of data contains enormous amounts of knowledge, both in space and time. Knowledge that can be used to find new applications with economical and social impact. Of course, we need techniques to find useful knowledge in this massive amount of raw data.

One important problem with the study of these datasets, is privacy. Extracting knowledge from this raw data, must be done with privacy–preserving methods and the European *GeoPKDD project – Geographic Privacy-aware Knowledge Discovery and Delivery* [geo] – concentrates on finding useful knowledge, with respect to the privacy of the providers of this data by using datamining techniques, like similarity search, to find interesting patterns.

The GeoPKDD project uses the mentioned techniques to find interesting information in the domain of *Geographic Information Science (GIS)*. But similarity search or the distance measures and techniques used for similarity search, can also be used for other applications in the GIS domain, like map matching [BPSW05], clustering [HK01] or query-by-sketch [KMV06].

Even in domains completely different from GIS, such as medical science, can we find these techniques where they are used, for example, to find anomalies in an ECG [Kel06] or to find cancercells in human tissue [KYM$^+$05, NCV$^+$03]. Examples can be found in Figure 1.1 and Figure 1.2.

When used with an ECG, the pattern recorded by the machine is compared with a database with known problems. By matching the recorded pattern with patterns in a database, it can be easier and faster to diagnose a patient correctly.

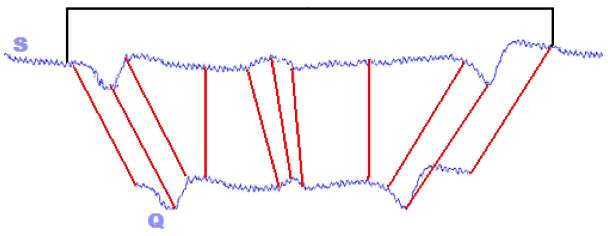


Figure 1.1: Example of similarity search for an ECG [Kel06].

In Figure 1.2, an application for finding cancercells in human tissue is used. The RNA data is extracted from the tissue and clustered accordingly. Then the found profile is matched, using similarity search techniques, against cancerprofiles to find cancer in the tissue. Since this is all done at the RNA level, it is possible to find a starting cancer much sooner by using this technique.

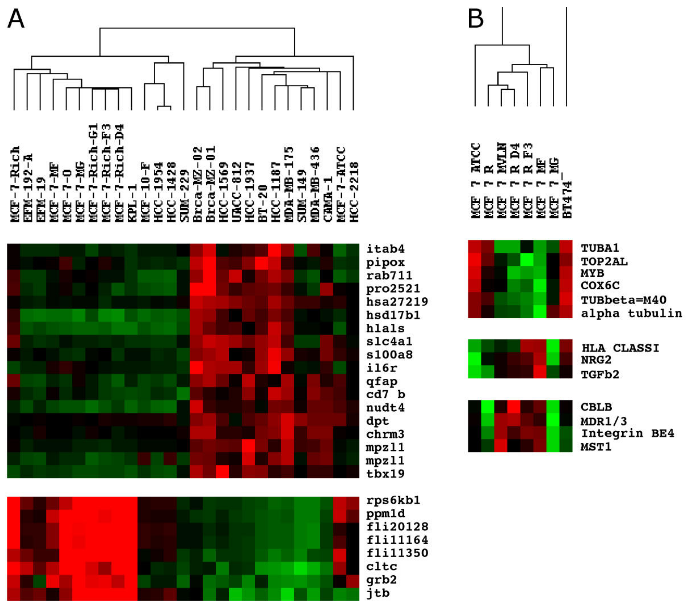
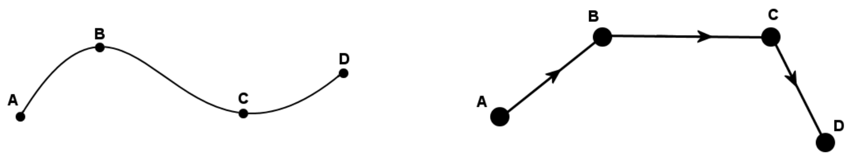


Figure 1.2: Hierarchical clustering of RNA expression profiles [NCV$^+$03].

In this work, we take a look at similarity search as it is used with the GeoPKDD project. Even more specific, we take a look at two general methods of similarity search: *quantitative methods* and *qualitative methods.* Since we concentrate on geographical data with a time dimension, we will work with methods for *trajectories* and, more specifically, *trajectories only containing straight edges*, of which an example can be seen in Figure 1.3(b).



(a) A trajectory with curved edges (b) A trajectory with only straight edges

Figure 1.3: Example trajectories.

Although we do not consider curved trajectories, like the one in Figure 1.3(a), it is not impossible to use our methods with curved ones. Every curved trajectory can be approximated by using a piecewise linear function. This approximation only contains straight edges and thus it can be used with the techniques that we will propose in this work.

Another thing we might consider is that a trajectory in the real world can look like Figure 1.3(a), but its digital representation might contain only straight edges. Consider a man walking along the trajectory in Figure 1.3(a), carrying a GPS device to record his movement.

Assume the device records it's position at certain time intervals and that the corresponding geographical points are a, b, c and d. Since there is no information known about the route between these points, they are often considered to be the shortest path between two recorded points and thus a trajectory with only straight edges is created.

For easier computation and because it solves some problems that might arise with qualitative methods, we generalize all trajectories with a *generalization principle*, discussed in Section 3.1. This principle does not only solve the mentioned problems with qualitative methods, it can also be used to find a very good piecewise linear approximation of curved trajectories, as can be seen in Figure 3.1.

But what is such a *trajectory*? A trajectory can be seen as a path on a map: a route going from one geographical place (the startvertex) to another (the endvertex). Following this path takes time and at certain moments in time, we will mark our current position (a vertex along the trajectory). Or at certain positions we mark the time. An example of a map with such a trajectory can be found in Figure 1.4.

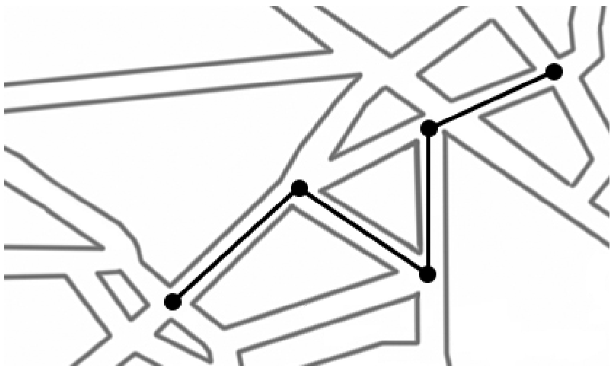


Figure 1.4: A simple illustration of a map with a trajectory projected onto it.

This example should give an excellent idea what trajectories are. A formal definition can be found in Section 2.1.

Remember the two principles mentioned before: *quantitative methods* and *qualitative methods*. In the past, research mostly focussed on quantitative methods, where similarity is calculated by using a distance measure and the smaller the distance between two objects, the more equal they are. Or the problem is solved as a decision problem: is the distance between two objects smaller than a given threshold [AG95], then they are considered similar.

Qualitative methods are different: if we take a look at the qualitative methods studied in Section 3.2, the *Double–Cross principle* [VDKD05, KMV06] and the *Twisted–Cross principle* [KM06], we see that qualitative methods return a percentile degree of similarity. Note that the second method is based on the first one. These methods are studied in Section 3.2.1 and Section 3.2.2.

These principles find a matrix representation of a trajectory, by calculating the relative positions of each segment in the trajectory to the others. A Double–Cross Matrix (DCM) can be found in Table 3.2 and in Table 3.1 we will find a Twisted–Cross Matrix (TCM). When we have a matrix representation of two trajectories, we can compare these matrices to determine their similarity, according to their similar entries. Of course, comparing a DCM with TCM makes no sense.

Now back to the quantitative methods. Two well-known and much used distance measures for quantitative methods are the Hausdorff Distance [HKR93, AHSW03] and the Fréchet Distance [AHK+06, God98, AG95, AKW01]. These are not the only two possibilities, but most other distance measures are used for specific domains or problems, like *Dynamic Time Warping* [Kel06].

The Hausdorff Distance is mostly used to find specific figures or patterns, e.g. circles, in a complex figure [HKR93]. It is a fast distance measure with good results, but it doesn't work with some special cases, as we will see later in this work.

Improvement is always possible and we have designed an algorithm for our trajectories that calculates the Hausdorff Distance between these two trajectories in asymptotic time $O(m \cdot n)$, with m and n the number of vertices in each trajectory, while the general algorithm for the Hausdorff Distance in $\mathbb{R}^2$ needs $O((m+n)\ log(m \cdot n))$ [AHSW03]. The full algorithm in pseudocode can be found in Listing 4.1.

Why is our algorithm that much faster? This is because we work with trajectories only containing straight edges and because we work with finite datasets. We know where every trajectory starts and ends and we know the total amount of vertices in each trajectory.

The Fréchet Distance is used for the same purpose as the Hausdorff Distance and gives us more precise results and does not suffer from the problems the Hausdorff Distance suffers, as we will see later in Chapter 5. And it has the same asymptotic time complexity as the Hausdorff Distance, but with enormous constants.

In fact, because of these enormous constants, the general algorithm for the Fréchet Distance is not applicable in practize [AG95]. Finding an algorithm for any metric space with dimension $n \geq 4$ is a *NP–hard* problem [God98].

Because of these problems, the Fréchet Distance itself is not really used in practical solutions, but a distance derived from the Fréchet Distance is used instead, like the *Discrete Fréchet Distance* [AHK+06], for example.

The Discrete Fréchet Distance calculates the Fréchet Distance between two polylines, but instead of using the entire polylines, only the vertices are used. With other words, the Fréchet Distance is calculated between two sets of points in space.

In this work, we followed the Fréchet idea and a great deal of time and work has been put in finding a new algorithm, designed for trajectories, based on the Fréchet Distance: *Fréchet–based Distance for trajectories.* The algorithm can be found in pseudocode in Section 4.2.2 and more specifically in Listing 4.2.

This new distance measure became the main focus of this work, because of it's excellent complexity and runtime, $O(m+n)$, of which the proof can be found in Section 4.2.5. We found that it works very well with clustering and other techniques for similarity search. The exact results of such tests can be found in Section 5.4.

We have been mentioning similarity search in the past pages, but how is it defined exactly? Similarity - in general - can be defined as some degree of symmetry or resemblance between two or more concepts or objects. For any kind of object in a metric space and for trajectories we can define many different kinds of similarity. Whatever detailed definition of similarity one uses, some kind of *degree of similarity* is always used as we will see in more detail in Chapter 5.

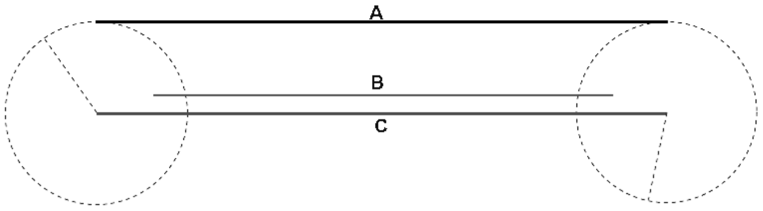


Figure 1.5: Illustration of different notions of similarity. Note that the circles are purely for easier understanding of the picture.

The exact definition of similarity greatly differs between research. If we take a look at research using quantitative methods, we see that Huttenlocher et al. [HKR93] consider objects similar when they are the same, after translation, rotation and/or scaling. A small square and a greater square, are considered similar.

While Alt et al. [AKW01] are a bit stricter: objects are only considered similar, when they are the same after translating, but they do not consider rotation or scaling.

If we take a look at Figure 1.5, for these two notions of similarity, we see that Huttenlocher would consider all three lines very similar, while Alt would consider line A en line C to be more equal than line A en line B or line B and line C.

If we would use a very strict notion of similarity, where two object can only be similar if they are exactly the same, without any transformations. Then line A in Figure 1.5 would be considered more similar to line B than to line C. This last notion of similarity is not really used in reality: it serves here as an example.

Chapter 2

Trajectories

In the previous chapter, we gave an informal definition of a trajectory as a route on a map, with markings at certain points. Logically, a point on such a trajectory contains two coordinates for the position of the specific point and a timestamp, a position in time. But these simple point and timevalue, give us a lot of information.

Recall the trajectory in Figure 1.4. What do we know of a person walking along this trajectory?

- The position of the person at a given time.
- The direction of his movement.
- The average speed at which he is walking.
- Some of the accelerations along the trajectory.
- The travelled distance.
- The time needed to walk the entire trajectory.
- Start– and endtimes.

This list illustrates the fact that by just adding a single time value for every vertex in a trajectory, the amount of information that can be derived from the trajectory increases greatly. And this makes trajectories very interesting objects to research.

This time value and the extra information it holds, made us take a look at the Fréchet Distance, which uses timeparametrizations to find a distance between two curves and it is this time value that gave us the idea for our new distance measure, discussed in Section 4.2.

After the informal definition mentioned before, let us now give a more formal and precise definition of these useful trajectories.

2.1 Definition

Let $\mathbb{R}$ denote the set of real numbers, $\mathbb{R}^2$ the real plane and $\mathbb{T}$ a set of *timestamps.*

Definition 1 (Trajectory). *A trajectory $T \subset \mathbb{R}^2 \times \mathbb{T}$ is a piecewise linear curve that is given by its vertices ordered by the timestamp $\in \mathbb{T}$, i.e. $T = \langle (x_0, y_0, t_0), (x_1, y_1, t_1) \dots (x_n, y_n, t_n) \rangle$ where $\forall\, i \in [0, n] : t_i < t_{i+1}$*

Let $T = \langle (x_0, y_0, t_0), (x_1, y_1, t_1) \dots (x_n, y_n, t_n) \rangle$ be a trajectory. The vertices (x_0, y_0, t_0) and (x_n, y_n, t_n) are respectively the start and end vertex of T.

We denote the line segment between (x_i, y_i, t_i) and $(x_{i+1}, y_{i+1}, t_{i+1})$ by $L_i(T)$, its length by $\ell(L_i(T))$ and its time duration by $td(L_i(T))$

We call $\ell(T) := \ell(L_0(T)) + \ell(L_1(T)) + \dots + \ell(L_{n-1}(T))$ the length of T.

If T is clear from the context, we will omit (T) from the above notations.

The semantics of the trajectory $T = \langle (x_0, y_0, t_0), (x_1, y_1, t_1) \dots (x_n, y_n, t_n) \rangle$ is the subset of $\mathbb{R}^2 \times \mathbb{T}$ consisting of all line segments between consecutive vertices of T, and we write:
$sem(T) := \bigcup\limits_{1 \le i < n} L_i(T)$

Often, when confusion is not possible we will just talk about T when we mean sem(T).

2.2 Set of timestamps $\mathbb{T}$

As $\mathbb{T}$ is defined as "a set of timestamps" as seen in Section 2.1, we can actually use anything to represent these timestamps as long as we use something that has an order defined on it. Most of the time, we will just use $\mathbb{R}$ for our timestamps, but there will always be an order on the used values: after all, we cannot go back in time.

In the remainder of this work, all timestamps will be represented with elements of $\mathbb{R}$. If any other notion for the timestamps would be used, it is possible to map every value used in that notion to a value in $\mathbb{R}$.

One well-known, but very important definition remains: that of a *metric.* This work contains different distance measures for trajectories and if we like to use and compare them, it is interesting to know if the measure in question is a metric or not. The reason why can be found in the next section.

2.3 Distance metrics

Metrics [Wei] and *pseudometrics* [Bar] are well known in modern geometry. We will first define the term metric and then use this definition to define the term pseudometric.

Definition 2 (Metric)**.** *A metric is a nonnegative function $g(x, y)$ describing the distance between different points for a given set that satisfies following conditions:*

1. *Triangle inequality:* $g(x, y) \leq g(x, z) + g(z, y)$
2. *Symmetry:* $g(x, y) = g(y, x)$
3. *Identity:* $g(x, y) = 0 \Leftrightarrow x = y$
4. *Non–negativity:* $g(x, y) \geq 0$

Definition 3 (Pseudometric)**.** *A pseudometric is a metric that allows the distance between two different points to be zero.*

Knowing of a distance measure is a metric is very interesting, because similarity search with quantitative methods, as we will see in Chapter 5, is done by measuring the distance between two objects and considering two objects more equal as their distance decreases. Two objects with a distance '0' are considered the same.

This is not possible without *non–negativity* and *identity*, after all, when two objects are the same but their distance is greater than '0', they would be considered not equal or even, if the distance is large enough, dissimilar [God98, HKR93, AHSW03].

The *triangle inequality* and *symmetry* are also interesting properties. If the symmetry doesn't hold, this means that it is possible for one object to be very simmilar to another object that is very dissimilar to the first.

Distance measures like this are sometimes used, e.g. *asymmetric Hausdorff Distance*, as described in Definition 7. This problem is then solved by computing both distances and using the smaller one, as can be seen in the same Definition 7.

The triangle inequality can be used to find bounds for uncalculated distances, by using calculated distances and thus it can speed up calculations [BFM$^+$96b]. Sometime similarity methods are based on the triangle inequality [BFM$^+$96a] and only a metric can be used with these techniques.

With some knowledge of trajectories in $\mathbb{R}^2 \times \mathbb{T}$ and an idea what *metrics* and *pseudometrics* are, we can now study some interesting distance measures for these trajectories: qualitative and quantitative methods.

We will first discuss the generalization principle and the qualitative methods this principle was originally designed for in Chapter 3: the Double–Cross and the Twisted–Cross.

Chapter 4 will start with the Hausdorff Distance in Section 4.1, which will not be discussed in great detail, but what will serve as an introduction before we move on to Fréchet Distance in Section 4.2, based on which we found a new distance measure for trajectories: the Fréchet–based Distance, as given in Section 4.2.2.

Chapter 3

The generalization principle and qualitative methods

If we like to study these trajectories defined in the previous chapter and, more specifically, if we like to calculate some kind of similarity between different trajectories, some problems might arise, that can be solved with a generalization.

- Some distance functions used with the qualitative Double–cross or Twisted–cross representations of trajectories, require that both trajectories to be compared have the same number of vertices. Examples of such distance functions can be found among matrix–based distances. Generalizing the trajectories can solve this problem.

- This concept makes it possible to represent a curve with a polyline only containing straight edges, this can speed up calculations or can make it possible to use methods that only work with such polylines. An example of this can be found in Figure 3.1.

- Since qualitative methods work with relative positions, as we will see in Section 3.2, the two figures in Figure 3.2 would be considered the same. But when we first generalize these two trajectories with the method described in Section 3.1, this problem does not occur.

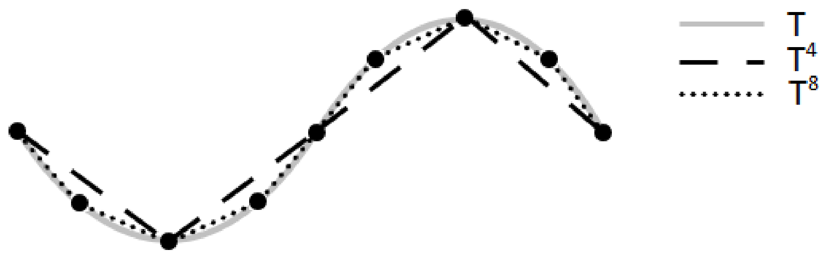


Figure 3.1: The curved line T is approximated by T^4 and T^8. It is clear that the higher the n in T^{2^n}, the better the approximation.

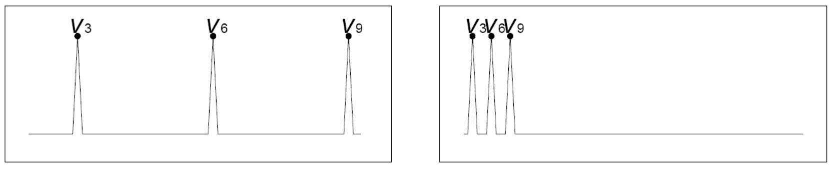


Figure 3.2: A problem with a boundary–based approach [VDKD05].

3.1 Generalization principle

Now we have given a summary of the advantages of the principle, let's continue with the formal definition of the generalization principle [VDKD05].

3.1.1 Definition

Definition 4 (Generalized trajectory)**.**
Let $T = \langle (x_0, y_0, t_0), (x_1, y_1, t_1) \ldots (x_n, y_n, t_n) \rangle$ *be a trajectory. We define the generalized trajectory of T, denoted* T^2, T^4, T^8, *... ,* T^{2^n}, *... as follows.*
$T^{2^n} = \langle (u_0, v_0, w_0), (u_1, v_1, w_1) \ldots (u_{2^n}, v_{2^n}, w_{2^n}) \rangle$ *with* (u_i, v_i, w_i) *the unique point on sem(T) where* $w_i = t_0 + \frac{i}{2^n} \cdot (t_n - t_0)$.

This generalization concept can be used for polylines containing only straight edges as well as for polylines also containing curved edges, because the distance along a curved line can be measured [VDKD05].

Of course, this is not the only existing generalization, but it is the one that will be used in this work, so will just call it *generalized trajectory*, instead of using any specific name.

3.1.2 Algorithm generalize trajectory

The algorithm to calculate a *generalized trajectory* is fairly simple and straightforward. It is given in pseudocode in Listing 3.1 and can be found in [KM06]. The concept of this specific generalization was proposed in [VDKD05].

Listing 3.1: Algorithm generalize trajectory

```
Function generalize_trajectory
Input: trajectory T; threshold $0 < \varepsilon \leq 1$

Set: n := 1
compute $T^2$;

while $|\ell\left(T^{2^n}\right) - \ell\left(T\right)| \geq \varepsilon$
    n := n + 1
    compute $T^{2^n}$ from $T^{2^{n-1}}$ and $T$;
end;

return $T^{2^n}$;

end function;
```

Although the algorithm was given in [KM06], the proof that the algorithm would stop on any input contained an error: case 1(b) of the proof, illustrated in Figure 3.3, was not included. We corrected this proof so we would be able to use this principle in the remainder of this work.

Proposition 1. *The algorithm* GENERALIZE_TRAJECTORY *in Listing 3.1 terminates on any input T and $0 < \varepsilon \leq 1$.*

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

If we construct $T^{2^{n+1}}$ from T^{2^n} and T there are two possible cases:
Case (1): if each vertex of $T^{2^{n+1}}$ is also an element of sem(T^{2^n}), then there are two subcases: (a) sem(T^{2^n}) = sem(T) or (b) sem(T^{2^n}) $\neq$ sem(T). [1]

[1] If we take a look at an example of a trajectory where this case arises in Figure 3.3, we can clearly see that, $sem(T^4)$ and $sem(T^8)$ are equal, but neither one of them is equal to $sem(T)$. As we can see on the picture, this is because we have unhandled vertices. If $sem(T^{2^n}) = sem(T^{2^{n+1}}) \neq sem(T)$, it will always be because there is some part of T that is not (yet) generalized.

Case 1 (a): If we construct $T^{2^{n+1}}$ from T^{2^n} and T and $sem\left(T^{2^n}\right)$ and $sem\left(T\right)$ are equal, then they contain the same vertices and the same line-segments between these vertices (See Definition 1). Thus we can conclude that $\ell\left(T^{2^n}\right) = \ell\left(T^{2^{n+1}}\right) = \ell\left(T\right)$, the stopcondition is satisfied and the algorithm will stop.

Case 1 (b): $\text{sem}\left(T^{2^n}\right) \neq \text{sem}(T)$. $\Rightarrow \exists\, m > n : T^{2^m}$ has at least one vertex $\text{w} : \text{w} \notin sem\left(T^{2^n}\right)$. Because every $T^{2^{n+1}}$ is constructed from T^{2^n}, we know that $\ell\left(T^{2^{n+1}}\right) \leq \ell\left(T^{2^n}\right)$. Of course, $\forall\, n \geq 1 : \ell\left(T^{2^n}\right) \leq \ell\left(T\right)$. If we now use the triangle inequality, we can conclude that $\ell(T^{2^n}) < \ell(T^{2^m}) \leq \ell(T)$.

Case (2): there is at least one vertex v of $T^{2^{n+1}}$ that is not an element of $sem(T^{2^n})$. We can use the same deduction as (1b): because of the triangle inequality and by construction, we know that $\ell(T^{2^n}) < \ell(T^{2^{n+1}}) \leq \ell(T)$. And thus we can conclude that $\forall\, m \in \mathbb{R}\ with\ m > n : T^{2^m}$ wil converge towards T.

Or in other words: there is at least one vertex of $sem(T^{2^{n+1}})$ that is not an element of $sem(T^{2^n})$. This means that $\ell(T^{2^n}) < \ell(T^{2^{n+1}})$ and, by construction, we know that $\ell(T^{2^{n+1}})$ is always smaller or equals than $\ell(T)$. And thus we can conclude that $\forall\, m \in \mathbb{R}\ with\ m > n : T^{2^m}$ wil converge towards T.

From (1) and (2) we can deduce that $\lim_{n\to\infty} \ell(T^{2^n}) = \ell(T)$ and therefore there exists a $n \geq 1$ such that $\left|\ell(T) - \ell(T^{2^n})\right| < \varepsilon$ □

We remark that there are some cases where $sem\left(T\right)$ and $sem\left(T^{2^n}\right)$, will never be exactly the same [KMV06], no matter the value for n. But eventually we will approximate the original trajectory T close enough.

An example can be found in Figure 3.4, where every step in the generalization process will create a T^{2^n} that gets closer to points a and b, but will never actually reach them because $\nexists\, x\ : x/2 = 1/3$. The algorithm will however, reach a point where a n is found, such that $|\ell\left(T^{2^n}\right) - \ell\left(T\right)| \geq \varepsilon$ [KMV06].

In the beginning of this chapter, we gave a few reasons why the generalization principle is important. Two of these reasons where because some qualitative methods for similarity search for trajectories have some special requirements. Since the generalization principle has originally been developed to tackle these problems, we will give a short introduction on qualitative methods.

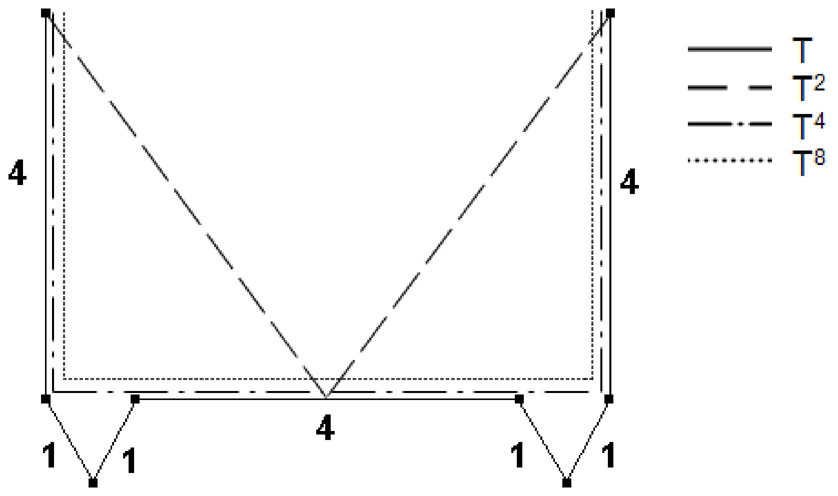


Figure 3.3: Illustration of case 1 (b) of the proof. The numbers on the figure are the length of the nearest segment. As can be seen, $sem(T^4) = sem(T^8) \neq sem(T)$, but $sem(T^{16}) = sem(T)$.

Qualitative methods for similarity search [VDKD05, KMV06, KM06] are rather new in comparison to quantitative methods [God98, HKR93] and there are not many publications yet. We will give here a short introduction to two of these methods: the *Double–Cross method* and the *Twisted–Cross method.*

3.2 Qualitative methods

If we take a look at qualitative methods in general, we will see that they do not use the exact position of vertices or the length of trajectories or segments: all calculations are based on the relative position of one segment to another.

This gives a greater importance to the general shape and orientation of the trajectory rather than the exact position and the coordinates involved. An advantage of this, is that we can use these methods to compare trajectories no matter their position, rotation or even scale.

A disadvantage is that it is nearly impossible to redraw a shape from it's qualitative representation, but this still needs more research. For the European GeoPKDD project, this can also be seen as an advantage: if it is very difficult to redraw a trajectory, then it is also nearly impossible to find the original trajectory and the person where the data comes from. In other words: it increases the privacy.

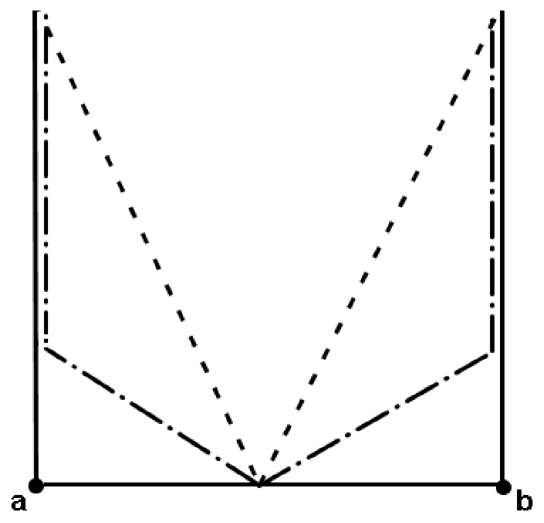


Figure 3.4: Illustrations of a trajectory T, drawn in full lines, T^2 in dashed lines and T^4 in dashed-dotted lines. The length of each side of T is 1.

In this section, we will not give an algorithm for the Double–Cross or the Twisted–Cross, but we will explain the principles behind it and give examples to illustrate how the results of these methods are calculated.

3.2.1 The Double–Cross method

The Double–Cross principle [VDKD05, KMV06] is a way of qualitatively representing two vectors with a tuple that expresses the orientation of both of them with respect to eachother. Or in other words, we represent two vertices with a representation that expresses the relative position of the two vertices.

For each two vectors $\vec{u}$ and $\vec{v}$ in the trajectory, we calculate a code that is a representation of their position, denoted $DC\left(\vec{u}, \vec{v}\right)$. All these codes are put into a table and can be interpreted.

How does it work? The main principle is actually quite simple: for each two line segments in a trajectory, we take the two tailpoints of both segments as reference points and place a double cross, like the one in Figure 3.5(a) on the segments. Note that the two vertical lines are perpendicular to the horizontal one.

The points on the figure with coordinates (x_i, y_i) and (x_j, y_j) are the reference points. When the cross is in place, we assign a code (a four–tuple) to the pair of segments, depending on their direction. A full list of all the codes can be found in Appendix A.

We will now give a formal definition of the Double–Cross. For the definition, we will refer to the vector between (x_i, y_i) and (x_j, y_j) as $\overrightarrow{u}$ as shown on Figure 3.5(b). In the definition, we will calculate $DC\left(\overrightarrow{\ell_i}, \overrightarrow{\ell_j}\right)$. This definition is taken from [KMV06].

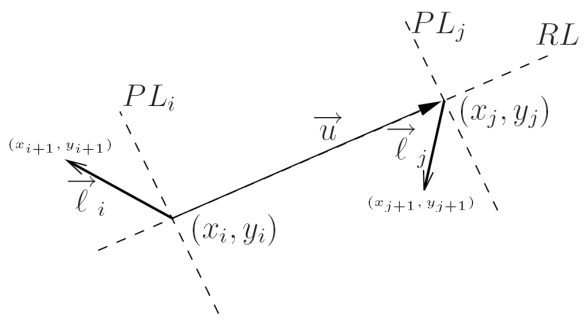
Definition 5 (Double–Cross)**.** *For $\overrightarrow{\ell_i} = \overrightarrow{\ell_j}$, we define, for reasons of continuity, $DC\left(\overrightarrow{\ell_i}, \overrightarrow{\ell_j}\right) = (C_1\, C_2\, C_3\, C_4) = (0\,0\,0\,0)$. For $\overrightarrow{\ell_i} \neq \overrightarrow{\ell_j}$, we define the 4–tuple $DC\left(\overrightarrow{\ell_i}, \overrightarrow{\ell_j}\right) = (C_1\, C_2\, C_3\, C_4)$ as follows:*

$$\begin{array}{lll}
C_1 = - & iff & (x_{i+1}, y_{i+1}) \textit{ lies on the same side of } PL_i \\
 & & \textit{as } (x_j, y_j) \textit{ and } (x_{i+1}, y_{i+1}) \notin PL_i \\
C_1 = 0 & iff & (x_{i+1}, y_{i+1}) \in PL_i \\
C_1 = + & iff & \textit{else} \\
C_2 = - & iff & (x_{i+1}, y_{i+1}) \textit{ lies on the same side of } PL_j \\
 & & \textit{as } (x_j, y_j) \textit{ and } (x_{i+1}, y_{i+1}) \notin PL_j \\
C_2 = 0 & iff & (x_{i+1}, y_{i+1}) \in PL_j \\
C_2 = + & iff & \textit{else} \\
C_3 = - & iff & (x_{j+1}, y_{j+1}) \textit{ lies on the left of } \overrightarrow{u} \\
 & & \textit{as } (x_i, y_i) \textit{ and } (x_{j+1}, y_{j+1}) \notin RL_3 \\
C_3 = 0 & iff & (x_{j+1}, y_{j+1}) \in RL \\
C_3 = + & iff & \textit{else} \\
C_4 = - & iff & (x_{j+1}, y_{j+1}) \textit{ lies on the right of } \overrightarrow{u} \\
C_4 = 0 & iff & (x_{j+1}, y_{j+1}) \in RL \\
C_4 = + & iff & \textit{else}
\end{array}$$

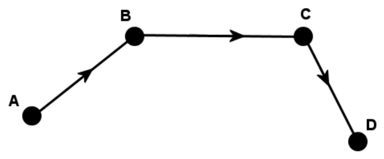
Let's continue with an example to make things more clear. Consider the trajectory in Figure 3.5(b). After applying the Double–Cross method, we found Table 3.1(a) with the results. This table is called the *Double–Cross Matrix*. These results can easily be verified with the cross in Figure 3.5(a) and the codes in Appendix A.

As proposed in [VDKD05], this table can be easily simplified: the code for the pairing a line with itself is always 0000 and if $DC\left(\overrightarrow{u}, \overrightarrow{v}\right) = (a\, b\, c\, d)$ than $DC\left(\overrightarrow{v}, \overrightarrow{u}\right) = (b\, a\, d\, c)$.

We no longer need to fill the entire table: calculating the upper part of it will give us all the information we need. This leads to a smaller matrix and less runtime for calculating the codes. The reduced matrix of Table 3.1(a) can be found in Table 3.1(b).



(a) Visualization of the Double–Cross [KMV06].



(b) Example trajectory.

Figure 3.5: Illustrations for the Double–Cross method.

Table 3.1: Double–Cross codes for the trajectory in Figure 3.5(b).

(a) Shape Matrix

	AB	BC	CD
AB	0000	$-+0-$	$-+--$
BC	$+--0$	0000	$-+0-$
CD	$+---$	$+--0$	0000

(b) Reduced Shape Matrix

	BC	CD
AB	$-+0-$	$-+--$
BC		$-+0-$

An important problem with the Double-Cross – and with the Twisted–Cross – is that it does not use the timestamp in the trajectories and that it's very difficult to draw a trajectory from a given Shape Matrix.

But this last one is also an advantage: not being able to redraw a trajectory, gives us higher security: since multiple trajectories can have the same matrix, it is nearly impossible to find the "real" corresponding trajectory. This improves security and increases the privacy of the source of the data.

Both methods assume that the time is a continuous function or it considers the time to be qualitatively included by the generalization principle, seen in Section 3.1.

The real strength of this principle lies in similarity search, which is discussed in Chapter 5, with the specific details in Section 5.1, where we will compare this method with our new quantitative method.

We will now take a look at an even more experimental qualitative method: the *Twisted–Cross*.

3.2.2 The Twisted–Cross method

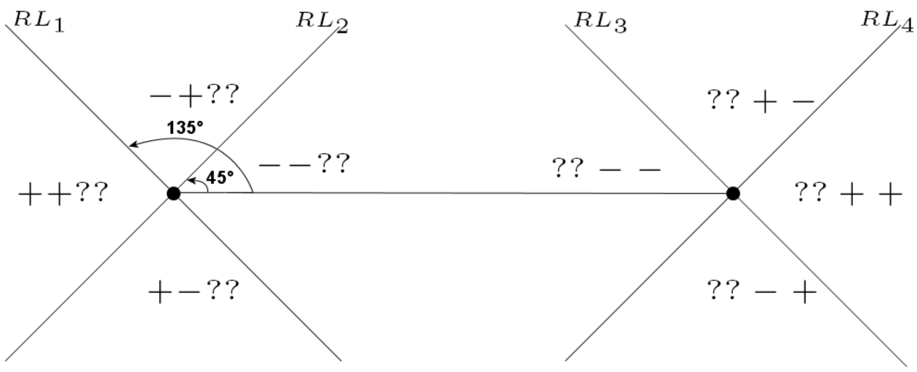


Figure 3.6: Visualization of the Twisted–Cross [KM06].

The Twisted–Cross is derived from the Double–Cross [KM06]. Research on this method is still very new and little has been published on this formalism. Due to these circumstances, we will only give a short introduction to this method.

Just like the Double–Cross, the Twisted–Cross computes a matrix for a trajectory by assigning a code (a four–tuple) to every pairing of two vertices $\vec{u}$ and $\vec{v}$ in the trajectory, denoted $TC\left(\vec{u},\vec{v}\right)$. But, as the name suggests, it uses a different kind of cross. An illustration of this cross can be found in Figure 3.6. The two reference points are again (x_{i-1}, y_{i-1}) and (x_{j-1}, y_{j-1}).

We will now give the formal definition of the Twisted–Cross, taken from [KM06] and afterwards we will give an example to illustrate the working of the method. The exact construction of the cross can be found in Figure 3.6, but for the definition we will be referencing to the vectors on Figure 3.7. For the definition, we will refer to the vector from (x_i, y_i) to (x_j, y_j) as $\vec{u}$ as shown on Figure 3.7.

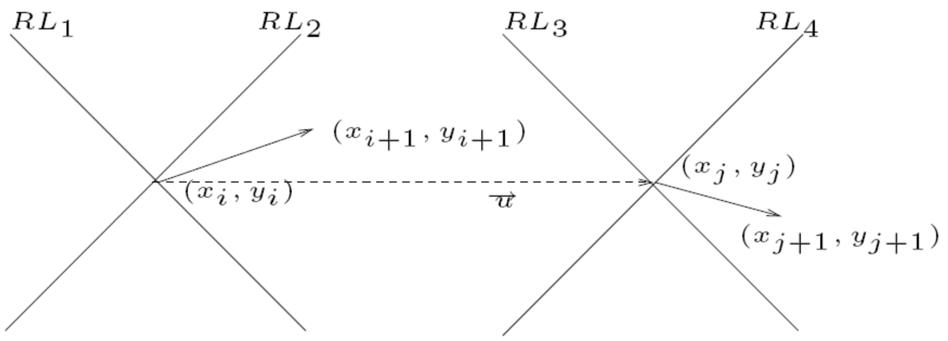


Figure 3.7: Illustration for the definition [KM06].

Definition 6 (Twisted–Cross)**.** *For $\overrightarrow{\ell_i}=\overrightarrow{\ell_j}$, we define, for reasons of continuity, TC* $\left(\overrightarrow{\ell_i}, \overrightarrow{\ell_j}\right) = (C_1\, C_2\, C_3\, C_4) = (0\,0\,0\,0)$*. For $\overrightarrow{\ell_i}\neq\overrightarrow{\ell_j}$, we define the 4–tuple TC* $\left(\overrightarrow{\ell_i}, \overrightarrow{\ell_j}\right) = (C_1\, C_2\, C_3\, C_4)$ *as follows:*

$C_1 = -$	*iff*	(x_{i+1}, y_{i+1}) *lies on the same side of* RL_1 *as* (x_j, y_j) *and* $(x_{i+1}, y_{i+1}) \notin RL_1$
$C_1 = 0$	*iff*	$(x_{i+1}, y_{i+1}) \in RL_1$
$C_1 = +$	*iff*	*else*
$C_2 = -$	*iff*	(x_{i+1}, y_{i+1}) *lies on the same side of* RL_2 *as* (x_j, y_j) *and* $(x_{i+1}, y_{i+1}) \notin RL_2$
$C_2 = 0$	*iff*	$(x_{i+1}, y_{i+1}) \in RL_2$
$C_2 = +$	*iff*	*else*
$C_3 = -$	*iff*	(x_{j+1}, y_{j+1}) *lies on the same side of* RL_3 *as* (x_i, y_i) *and* $(x_{j+1}, y_{j+1}) \notin RL_3$
$C_3 = 0$	*iff*	$(x_{j+1}, y_{j+1}) \in RL_3$
$C_3 = +$	*iff*	*else*
$C_4 = -$	*iff*	(x_{j+1}, y_{j+1}) *lies on the same side of* RL_4 *as* (x_i, y_i) *and* $(x_{j+1}, y_{j+1}) \notin RL_4$
$C_4 = 0$	*iff*	$(x_{j+1}, y_{j+1}) \in RL_4$
$C_4 = +$	*iff*	*else*

By placing this cross over a trajectory, one can easily see how it works and codes can be found very easily by hand. If a vertex should be exactly on one of the lines of one of the two crosses, the corresponding value in de 4–tuple will be 0.

When all values are calculated, we get a matrix representing the trajectory, called the *Twisted–Cross Matrix*. A small example of the Twisted–Cross can be found in Table 3.2(a).

Just like with the Double–Cross, this matrix can be reduced to a smaller matrix with the following rule, proposed in the same work that contained the definition: If $TC\left(\vec{u}, \vec{v}\right) = (a\,b\,c\,d)$ than $TC\left(\vec{v}, \vec{u}\right) = (c\,d\,a\,b)$. The reduces matrix of Table 3.2(a) can be found in Table 3.2(b).

Table 3.2: Twisted–Cross matrix for the trajectory in Figure 3.5(b).

(a) Twisted–Cross Matrix

	AB	BC	CD
AB	0000	$00-+$	$---+$
BC	$-+00$	0000	$00-+$
CD	$-+--$	$+-00$	0000

(b) Reduced Twisted–Cross Matrix

	BC	CD
AB	$00-+$	$---+$
BC		$00-+$

Although qualitative methods are studied in this work, much work remains. The focus of this work lies on the new Fréchet–based Distance and thus these qualitative methods are not studied in great detail. Now we have a notion on these methods and the relationship between these methods and the generalization principle, can we continue with the second principle used in similarity search: *quantitative methods.*

Chapter 4

Quantitative methods

As mentioned in Chapter 1, quantitative methods for similarity search, return a real number, that can be interpreted as a degree of dissimilarity or distance between two objects. To compute this number, a distance measure is used. In this chapter we will discuss two widely used distance measures for this kind of similarity search: *Hausdorff Distance* and *Fréchet Distance*.

For quantitative methods, the emphasis will be on the Fréchet Distance and – more specifically – a new distance measure based on the Fréchet Distance.

4.1 Hausdorff Distance

The Hausdorff Distance [Bla] is used to measure the distance between two non-empty sets. More specific, we can – and will – use this metric to measure the distance between two *trajectories* in Euclidean space.

4.1.1 General Definition

The Hausdorff Distance was discovered by Felix Hausdorff (° 1868 – † 1942), a german mathematician who became famous for his work on topological and metric spaces and set theory. The Hausdorff Distance was published in 1914 in his book *"Grundzüge der Mengenlehre"*, which builds on work by Fréchet and others.

We will now continue with the general definition for the Hausdorff Distance. Afterwards, we will complete our study of the Hausdorff Distance with a definition for the Hausdorff Distance for trajectories.

Definition 7 (Hausdorff Distance)**.** *Let X be a metric space and δ_X its metric. Given a point $x \in X$ and a non empty set $A \subseteq X$, let us first define the distance of x to A by $\delta_h(x, A) := \inf\limits_{a \in A} \delta_X(x, a)$.*
Then the ***Hausdorff Distance between A and B*** *is defined for any non empty sets $A, B \subseteq X$ as $\delta_H(A, B) := \max(\delta_{asym}(A, B), \delta_{asym}(B, A))$, where $\delta_{asym}(A, B) := \sup\limits_{a \in A} \delta_H(a, B)$ denotes the so called asymmetric Hausdorff Distance from A to B.*

Although *asymmetric Hausdorff Distance* is used in Definition 7, the Hausdorff Distance itself is symmetric and thus also called the *symmetric Hausdorff Distance.*

We notice that the Hausdorff Distance is a metric [Hen99]: $\delta_H(A, B) = \delta_H(B, A)$, $\delta_H(A, A) = 0$, $\delta_H(A, B) \leq \delta_H(A, C) + \delta_H(C, B)$ and $\delta_H(A, B) = 0 \Rightarrow A = B$ for a *closed and bounded* space. For a *non-closed* space, the Hausdorff Distance is a *pseudometric.*

To prove that the distance between two different objects is not always greater than 0 for a non-closed space, we just need to find a single example. If we take a look at Figure 4.1, we see that Figure 4.1(a) is an *open* disk in $\mathbb{R}^2$ and Figure 4.1(b) is a *closed* disk in $\mathbb{R}^2$. Both disks have the same radius.

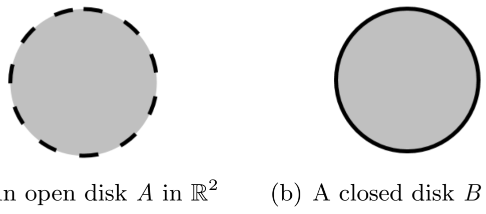

(a) An open disk A in $\mathbb{R}^2$ (b) A closed disk B in $\mathbb{R}^2$

Figure 4.1: Illustration that the Hausdorff Distance is not a metric for non–closed spaces.

If we translate both disks to have the same centers, then their Hausdorff Distance will be equal to 0, although both disks are not the same. Thus $\delta_H(A, B) = 0$ but $A \neq B$.

In the following explanation, we will use trajectories as an example, but they can be replaced by any two non-empty sets in any metric space.

An algorithm for the Hausdorff Distance will first calculate the distance between the startvertex of A and every vertex of B, the smallest of these distances will be remembered: this is the Hausdorff Distance between the startvertex and trajectory B. This will be repeated for every vertex in A and in the end we will take the biggest distance of all the remembered distances: this is the *Hausdorff Distance* between trajectory A and trajectory B.

This is a general definition for the Hausdorff Distance. If we can define a notion of distance (e.g. Euclidean Distance in $\mathbb{R}^2$), we can easily adapt this definition for trajectories in $\mathbb{R}^2 \times \mathbb{T}$. As we have noted earlier in Section 2.2, we use $\mathbb{R}$ for $\mathbb{T}$ (with a few limitations as seen in Section 2.2). We can then use the standard distance (Euclidean distance) in $\mathbb{R}^3$ to calculate the Hausdorff Distance for trajectories.

Now we will use this idea to give an algorithm for Hausdorff for trajectories and afterwards we will give a definition for trajectories in $\mathbb{R}^2 \times \mathbb{T}$.

4.1.2 Algorithm Hausdorff Distance for trajectories

This algorithm, proposed here in Listing 4.1, calculates the *asymmetric* Hausdorff Distance from one trajectory to another. To get the *symmetric* distance, the algorithm needs to run twice, once for each trajectory, and the maximum distance of the two runs will be the symmetric distance.

Because the focus in this work lies on the Fréchet–based Distance, we have not searched for a better algorithm for the Hausdorff Distance for trajectories. For the same reason, there has been no intensive testing of the algorithm.

The algorithm uses a simple datastructure to store the trajectories. This datastructure contains a list of vertices and appropriate functions. The ones needed for the algorithm in Listing 4.1, are listed here.

- *getStartVertex( )* Returns the first vertex of the trajectory.
- *getEndVertex( )* Returns the last vertex of the trajectory.
- *getNextVertex( )* Returns the next vertex of the trajectory, the first vertex following the last vertex of the trajectory that has been requested. If there are no more vertices in the trajectory, this function returns a null-value.

A few other functions are also necessarily for the algorithm:

- *getTime()* Returns the value of the timestamp of the vertex.
- *add(double num)* Add *num* to a set of values
- *maximum()* Returns the maximum value in a set.
- *EuclideanDistance(Vertex v_1, v_2)* Returns the Euclidean Distance in $\mathbb{R}^3$ between vertices v_1 and v_2.

Listing 4.1: Algorithm Hausdorff Distance for trajectories

```
Function Hausdorff_distance_for_trajectories

Input: trajectories $T, U \subset \mathbb{R}^2 \times \mathbb{T}$

Set $v_t$ = T.getStartVertex();
Set $v_u$ = U.getStartVertex();
Set $measured$ = $\emptyset$;

while( $v_t \neq$ T.getEndVertex() )
{
    Set $distance = +\infty$;

    while( $v_u \neq$ U.getEndVertex() )
    {
        set d = EuclideanDistance( $v_t$, $v_u$ );

        if( d < distance )
            distance = d;

        $v_u$ = U.getNextVertex();
    }end;

    $measured$.add( distance );

    $v_t$ = T.getNextVertex();

}end;

return $measured$.maximum();
```

Our algorithm in Listing 4.1 is very simple and straightforward: for every vertex of one trajectory, find the closest vertex of the other trajectory and remember their distance. Store all these found distances and in the end, return the maximum of the set of found distances.

As noted earlier, this algorithm computes the *asymmetric* Hausdorff Distance between two trajectories T and U. If we take a look at Figure 4.2, we can see the distance from T to U, calculated by the algorithm, represented by the gray line labelled *"A"* and the distance from U to T labelled *"B"*.

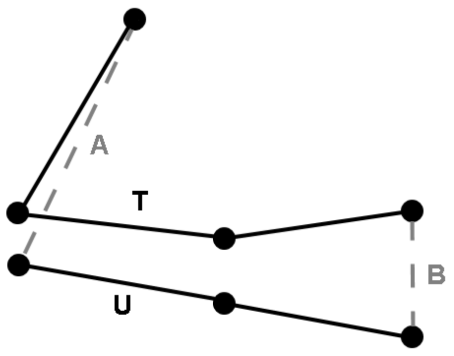


Figure 4.2: Assymetric Hausdorff Distance

Although the Hausdorff Distance and thus also the algorithm, does not have a special way of dealing with a timefactor in the trajectory, we use the 3D definition of the Euclidean distance is this algorithm. We remember that we mapped the timestamp in every vertex to a value in $\mathbb{R}$ and by calculating the distance with the 3–dimensional distance, we make sure not to forget an entire dimension of information.

4.1.3 Definition Hausdorff Distance for trajectories

Definition 8 (Hausdorff Distance for trajectories)**.** *Let A, B be trajectories in $\mathbb{R}^2 \times \mathbb{T}$ and D the Euclidean distance used in $\mathbb{R}^3$.*

$$\delta_H(A, B) := \max\{D_H(A, B), D_H(B, A)\}$$

$$\textit{with } D_H(A, B) := \sup_{a \in A}\left(\inf_{b \in B}(D(a, b))\right)$$

is the Hausdorff Distance from A to B in $\mathbb{R}^2 \times \mathbb{T}$.

Since distance measure D in the definiton is a metric, the Hausdorff Distance for trajectories in $\mathbb{R}^2 \times \mathbb{T}$ is also a metric [Hen99].

4.1.4 A note on Hausdorff Distance in this work

Hausdorff Distance is a widely known and used [Hen99, HKR93, AHSW03] distance metric for similarity search. As has been told in the beginning of this chapter, some basic research on this topic has been done for this work, but the focus for this work will be on Fréchet Distance and not on Hausdorff Distance. Thus, although the basic concepts of similarity search with Hausdorff Distance can be found in Section 5.2, they are far from complete.

4.2 Fréchet Distance

The Fréchet Distance was proposed by Maurice Fréchet (° 1878 – † 1973), a French mathematician, in his doctoral dissertation: *"Sur quelques points du calcul fonctionnel"*. He made major contributions to the topology of point sets and defined and founded the theory of abstract spaces.

The Fréchet Distance [God98] is, like the Hausdorff Distance, used as a measurement of the distance between two non-empty sets. The main difference between the two, is that the Fréchet Distance is designed to find a time parametrization for any object in a specified metric space.

The use of this parametrization, is to make sure that the direction of the trajectory is brought into account. But if a time parametrization already exists, we might be able to speed calculations up, if it is an usable parametrization. More about this idea can be found in Section 4.2.2.

To find a definition for the Fréchet Distance for trajectories in $\mathbb{R}^2 \times \mathbb{T}$, we start with a general definition and we will deduce a more appropriate definition for trajectories.

4.2.1 General definition

Definition 9 (Fréchet Distance). *Let (X, δ_X) be a fixed metric space and $d \in \mathbb{N}$ a fixed constant. This will be the dimension of the geometric objects we consider. The abbreviation $\sigma : A \overset{\sim}{\rightarrow} B$ stands for the fact that $\sigma : A \rightarrow B$ is an orientation preserving homeomorphism. For two objects $f : A \rightarrow X$ and $g : B \rightarrow X$ the Fréchet Distance is defined by*

$$\delta_F (A, B) := \inf_{\sigma : A \overset{\sim}{\rightarrow} B} \left(\sup_{x \in A} \left(\delta_X \left(f \left(x \right), g \left(\sigma \left(x \right) \right) \right) \right) \right)$$

or $+\infty$ if no such σ exists.

This distance is usually [God98, Ewi85, AHK$^+$06] explained with a man who is walking his dog. Each of them walks on their respective curves and they can each control there own speed, but they cannot go back. The Fréchet Distance of these two curves is the minimal length of a leash necessary for the dog and his handler to move from the starting points of the two curves to their respective endpoints.

More mathematically, this distance searches for a time parametrization that minimizes the distance between two non-empty sets in a metric space.

In this entire work, we will only consider trajectories in $\mathbb{R}^2 \times \mathbb{T}$. The trajectories already have time values and we could try to find a definition for the Fréchet Distance for trajectories that uses this timestamp. Using a version of the Fréchet Distance without these values would would ignore an entire dimension of information: after all, we know where the man and his dog are at a given time. The value of this timedimension has been illustrated in Section 2.

Using this timestamp could possible make it easier to compute the Fréchet Distance between two trajectories. Although an algorithm that calculates this distance in ***O(nm log(nm))*** time, with n the number of vertices of one trajectory and m the number of vertices of the other, is known, it is not a feasible algorithm since it involves enormous constants.

> "*It should be mentioned here that Algorithm 3, although it has low asymptotic complexity, is* ***not really applicable in practice****. In fact, Cole's parametric searching technique makes use of the Ajtai-Komlos-Szemeredi (AKS) sorting network wich involves* ***enormous constants****.*" [AG95]

Note that, because we used a quote to illustrate this important problem with the Fréchet Distance, it contains a reference to an "Algorithm 3" that cannot be found in this work.

This is the main reason why sometimes the less correct Hausdorff Distance is used: it's is easier to implement and even if we use the algorithms with the same asymptotic runtime ***O(nm log(nm))***, the Hausdorff Distance will be calculated a lot faster, because the constants are a lot smaller [HKR93], as we will see in Section 5.2 and Section 5.3.

4.2.2 Algorithm for Fréchet–based Distance

As said before, by adding a time parametrization to a trajectory, we bring the direction of the trajectory into account. But if such a time parametrization already exists, we might be able to speed calculations up, if it is an usable parametrization, because we do not need to calculate one.

We have developed our own algorithm, inspired by the original principle of the Fréchet Distance to find a distance measure for trajectories that is useful in similarity search and that has a better runtime than the general Fréchet definition, as we will see in Section 4.2.5.

We will use this algorithm to find a suitable definition for this new distance measure for trajectories in $\mathbb{R}^2 \times \mathbb{T}$. The algorithm is given in pseudocode.

We will use the same datastructure for trajectories as used in Section 4.1.2, with an extra functionality.

- *getVertexBefore(Vertex v)* Given a vertex v, this returns the last vertex of the trajectory before v.

The extra functions are also the same as the ones described in Section 4.1.2, plus one extra function.

- *shortestDistance(Vertex v, Vertex u)* Returns the shortest Euclidean distance between vertex v and the segment $(getVertexBefore(u), u)$ (Details of this function can be found in Listing 4.3).

Note that we don't need the two functions for a set of numbers:
add(double num) and *maximum()*.

Listing 4.2: Algorithm Fréchet–based Distance

```
Function Frechet_based_distance
Input: trajectories $T, U \in \mathbb{R}^2 \times \mathbb{T}$

Set distance = 0;
Set $v_t$ = T.getStartVertex();
Set $v_u$ = U.getStartVertex();

while $v_t \neq$ T.getEndVertex() || $v_u \neq$ U.getEndVertex()
{
    if( $v_t$.getTime() == $v_u$.getTime() )
    {
        aidDistance = EuclideanDistance( $v_t$, $v_u$ );

        $v_t$ = T.getNextVertex();
        $v_u$ = U.getNextVertex();
    }
    else if( $v_u$.getTime() < $v_t$.getTime() )
    {
        aidDistance =
            shortestDistance( $v_u$, $v_t$ );

        if( $v_u$ == U.getEndVertex() )
            $v_t$ = T.getNextVertex();
        else
            $v_u$ = U.getNextVertex();
    }
    else
    {
        aidDistance =
            shortestDistance( $v_t$, $v_u$ );

        if($v_t$ == T.getEndVertex())
            $v_u$ = U.getNextVertex();
        else
            $v_t$ = T.getNextVertex();
    }

    if( aidDistance > distance )
        distance = aidDistance;

}end;

return distance;

end function;
```

The algorithm uses the timestamps of the vertices to calculate a distance measure, instead of searching for the optimal parametrization.

To make a study of this algorithm possible, we need more detailed explanations of the functions used in this algorithm. All the functions returning vertices - *getNextVertex(), getVertexBefore(), getStartVertex(), getEndVertex()* - can be implemented very efficiently depending on the used data structure. For our tests, we have used a data structure based on a vector, which already contains many of these functions.

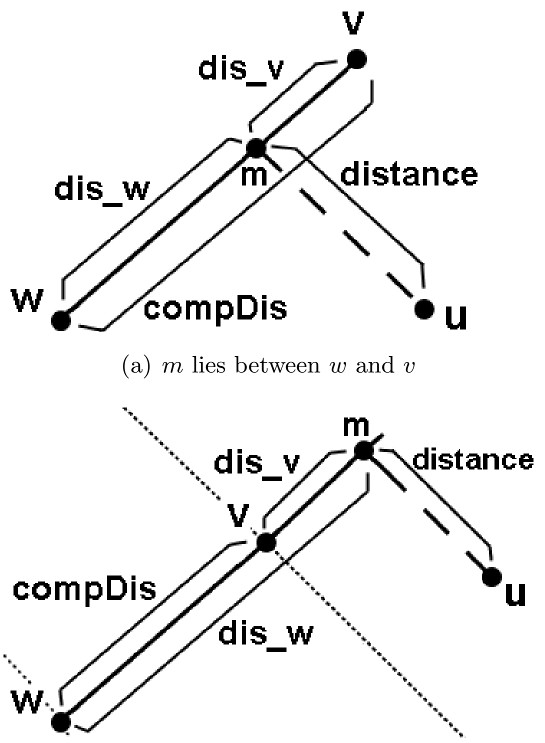
The function *getTime()* is trivial. This leaves us with one last function to explain: *shortestDistance(Vertex, Vertex).* To explain this function, we will first give the algorithm in pseudocode and explain it afterwards.

Listing 4.3: Function shortestDistance

```
1  Function shortest_distance
2  Input: Vertex $u$, Vertex $v$
3
4  Set $w = getVertexBefore(v)$;
5  Set $distance = \frac{|(w-v) \times (v-u)|}{|v-w|}$;
6
7  Set $dis\_v = (u.EuclideanDistance(v))^2 - (distance)^2$;
8  Set $dis\_w = (u.EuclideanDistance(w))^2 - (distance)^2$;
9
10 Set $sqRoot = \sqrt{dis\_v} + \sqrt{dis\_w}$;
11
12 Set $compDis = v.EuclideanDistance(w)$;
13
14 if( $sqRoot \neq compDis$ )
15 {
16     $distance = u.EuclideanDistance(v)$;
17     Set $aidDistance = u.EuclideanDistance(w)$;
18
19     if( $aidDistance < distance$ )
20         $distance = aidDistance$;
21 }
22
23 return distance;
24
25 end function;
```

The algorithm shortestDistance in Listing 4.3, first calculates the distance between vertex u and the segment between vertices v and w (Line 5). Of course this is the distance from u to the *straight line* determined by v and w, not the distance from u to the *segment* between w and v. But we can use this distance to find the real shortest distance from u to the segment between w and v.

We will explain how we find this distance using Figure 4.3. First, let m be the intersection of the line wv and mu the perpendicular line through u on wv. Let $d(a,b)$ be the Euclidean distance in $\mathbb{R}^3$ from a to b with $a, b \in \mathbb{R}^3$. We already calculated $d(u,m) = distance$ and $d(u,v)$ and $d(u,w)$ can be easily computed.



(a) m lies between w and v

(b) m does not lie between w and v

Figure 4.3: Illustrations of the shortestDistance algorithm

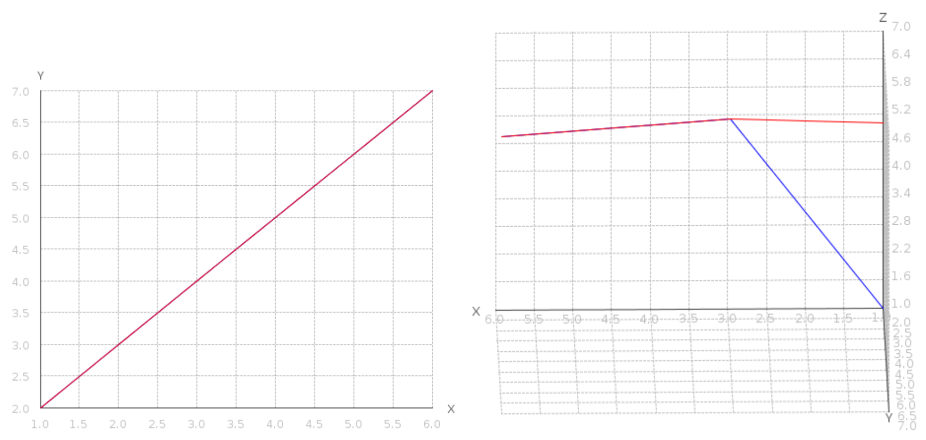
With these newly calculated distance and the Pythagorean Theorem, we can calculate $d(w, m)$ and $d(v, m)$. If we take the sum of these two and compare it with $d(v, w)$, we can check if u lies between these two vertices (as can be seen on Figure 4.3(a)) or not (as can be seen on Figure 4.3(b)). If it does, the shortest distance equals the distance from u to m on the line determined by v and w. If it doesn't, the shortest distance is $d(u, v)$ or $d(u, w)$, depending on which one is closer to u.

4.2.3 Note on 2- and 3-dimensional distance

In the algorithm and in the proofs afterwards, we will use the Euclidean Distance in $\mathbb{R}^3$ and not the 2–dimensional variant. We can use either of them, but depending on the choice, we are looking for a different kind of similarity between the trajectories.

The 3–dimensional representation of a trajectory is an *interpretation*, to be able to visualize and work with the timevalue in each vertex. If we would like to compare the general shape of two trajectories, a 2–dimensional approach would be better, because the 3–dimensional adds the timevalues as an extra spatial dimension to our 2–dimensional trajectory. Depending on the choosen idea, we place a stronger or less strong emphasis on the timevalues for similarity search.

For example, take a look at the following two trajectories:
$T = \langle (1,2,1), (3,4,6), (6,7,7) \rangle$ and $U = \langle (1,2,5), (3,4,6), (6,7,7) \rangle$. These two trajectories are equal, except for their timevalues. We can see that U travels along exactly the same path as T, but much faster. This example is plotted in 2D and 3D in Figure 4.4.



(a) 2D plot of the example (b) 3D plot of the example

Figure 4.4: Illustration of the use of 2D and 3D distance

In general: if two vertices exist with the same timestamp, they will have a distance 0 because they are completely the same. If we have a vertex from one trajectory with a timestamp that has no corresponding vertex with the same timestamp from the other trajectory, we would search for the shortest 2–dimenstional distance between that vertex and a line containng that vertex, which is obviously 0.

With these details of the algorithm for the Fréchet–based Distance, we can and will prove that it stops on any input. Afterwards, we will have a closer look at the complexity of the algorithm.

4.2.4 Note on curved trajectories

This algorithm is designed to work with trajectories in $\mathbb{R}^2 \times \mathbb{T}$ *only containing straight edges.* If we allow curved edges, the results will be useless. Remember that we already adressed this problem in Section 2.1. The *generalization principle* can easily solve this problem, as shown in Figure 3.1.

The problems and possibilities of curved trajectories in $\mathbb{R}^2 \times \mathbb{T}$, will not be discussed in this work, but we wanted to point this problem out, so it can be examined in future work.

4.2.5 Properties of the algorithm

Proposition 2. *The algorithm in Listing 4.2 terminates on any input T and U.*

Proof. To prove this proposition, we must prove that $v_t \neq$ *T.getEndVertex()* && $v_u \neq$ *U.getEndVertex()* will eventually evaluate to *false* for every $T, U \in \mathbb{R}^2 \times \mathbb{T}$

It is easy to see that in every run, the algorithm gets the next vertex from either T or U or both. If the final vertex of one of the trajectories is reached, the algorithm will continue with fetching the vertices of the other trajectory until the endvertices of both trajectories are reached. Since both trajectories consist of a finite amount of vertices, v_t and v_u will always converge to the corresponding endvertex of their respective trajectories and the algorithm will stop. □

Proposition 3. *The algorithm in Listing 4.2 has time complexity $O(n + m)$ with n and m the number of vertices in T and U, respectively.*

Proof. The algorithm contains no recursion and only a single loop. If we take a look at all the functions used in this loop, we see that all of them can be calculated in constant time, thus with complexity: $O(1)$. The loop itself will run at most $n + m$ times, with n the number of vertices of trajectory T and m the number of vertices of trajectory U.

Since k is a constant, the dominant factor is $O(n + m)$ and this is the time complexity of the algorithm. □

Although we have proven that the algorithm stops and we know it's time complexity, a very important remains unanswered: is the algorithm optimal? We need to check and prove that no faster algorithm exists.

Proposition 4. *The algorithm for the Fréchet–based Distance given in Listing 4.2 has optimal time complexity.*

Proof. To find the correct distance, we need to handle each vertex of each trajectory at least once. If we have n vertices in the first trajectory and m vertices in the second trajectory, we get a worst case complexity of $O(n+m)$, when, for example, all the vertices of the first trajectory have a smaller timestamp than any vertex in the second trajectory. $\square$

4.2.6 Definition of the Fréchet–based Distance

After we examined the algorithm in Listing 4.2, we can now give a definition for this new distance measure.

Definition 10 (Fréchet–based Distance)**.** $\forall T, U \subset \mathbb{R}^2 \times \mathbb{T} : \delta_{\varphi_b}(T,U)$, *the* ***Fréchet–based Distance*** *between T and U is defined as:*

$$\delta_{\varphi_b}(T,U) = \max\left\{\max_{t\in T}\left(D_\varphi(t,U)\right), \max_{u\in U}\left(D_\varphi(u,T)\right)\right\}$$

with:

$D_\varphi(t,U) = D(t,u)$ *for $u \in U$ with D the Euclidean Distance in $\mathbb{R}^3$ if u and t have the same timevalue.*

otherwise:

$D_\varphi(t,U) = D_S(t,u_1u_2)$ *with u_1 the vertex in U with the largest timestamp smaller than the timestamp of t, u_2 the vertex in U with the smallest timestamp greater than the timestamp of t and $D_S(a,AB)$ the distance between point a and line segment AB in $\mathbb{R}^3$.*

otherwise:

$D_\varphi(t,U) = D(t,u)$ *for $u \in U$ with D the Euclidean Distance in $\mathbb{R}^3$ with u having the timevalue closest to the timevalue of t.*

Note that we use the Euclidean Distance in $\mathbb{R}^3$ in our definition. If we remember Section 4.2.3, we see that we can replace this by the Euclidean Distance in $\mathbb{R}^2$, depending on our notion of similarity.

Although both definitions of the Euclidean Distance are metrics with the same properties, it does not matter for the proofs of Proposition 5, Proposition 6 and Proposition 7. For both metrics, the proof remains the same.

4.2.7 Properties of the Fréchet–based Distance

Proposition 5. $\delta_{\varphi_b}(T,U) = \delta_{\varphi_b}(U,T)$ *or the Fréchet–based Distance for trajectories in* $\mathbb{R}^2 \times \mathbb{T}$ *is symmetric.*

Proof. Trivial, following directly from the definition. □

Proposition 6. $\delta_{\varphi_b}(T,T) = 0$ *or the Fréchet–based Distance between a trajectory and itself is* 0 *(Identity).*

Proof. Let $S \subset \mathbb{R}^2 \times \mathbb{T}$. Run the algorithm for Fréchetbased Distance with $T = S$ and $U = S$. Since $T = S = U$: the startvertices of T and U are the same and the algorithm begins in the first if-test and $v_t = v_u \Rightarrow EuclideanDistance(v_t, v_u) = 0$.

We then take the next vertex of T and U, wich are also the same and thus also are at Euclidean distance 0. This repeats itself for every vertex in T and U. In the end, the algorithm outputs the greatest of the found distances, but since they are all 0 this will also be 0 and thus $\delta_{\varphi_b}(T,U) = 0$. □

Proposition 7. *If T and U have the same number of vertices, then* $\delta_{\varphi_b}(T,U) = 0 \Rightarrow T = U$ *or if two trajectories have a Fréchet–based Distance 0, they are the same (non–negativity).*

Proof. Let T, $U \subset \mathbb{R}^2 \times \mathbb{T}$ and $\delta_{\varphi_b}(T,U) = 0$ Since $\delta_{\varphi_b}(T,U) = 0$, we have two possibilities:

1. $\forall\, v_t \in T, \exists\, v_u \in U : v_t.getTime() = v_u.getTime() \wedge D(v_t, v_u) = 0 \Rightarrow T = U$

2. $\exists\, v_t \in T, \exists\, v_{u_1}, v_{u_2} \in U : v_{u_1}$ and v_{u_2} are consecutive vertices $\wedge v_{u_1}.getTime() < v_t.getTime() \wedge v_t.getTime() < v_{u_2}.getTime() \wedge shortestDistance(v_t, v_{u_2}) = 0$. This means that v_t lies between v_{u_1} and v_{u_2} and on the line between these two vertices and thus: $v_t \in U$.

□

Since we work with *generalized trajectories*, as seen in Chapter 3, we can always make sure that two trajectories have the same number of vertices.

An example of the reason why we need trajectories with the same number of vertices, can be found in Figure 4.5. As we can see in this example, both trajectories have a different amount of vertices, but they do take up the same amount of time and space in $\mathbb{R}^2 \times \mathbb{T}$.

So, although both trajectories are not exactly the same mathematically speaking, they are equal in time and space, thus they can be considered *similar* and the distance measure can be used for similarity search, as we will see in Chapter 5.

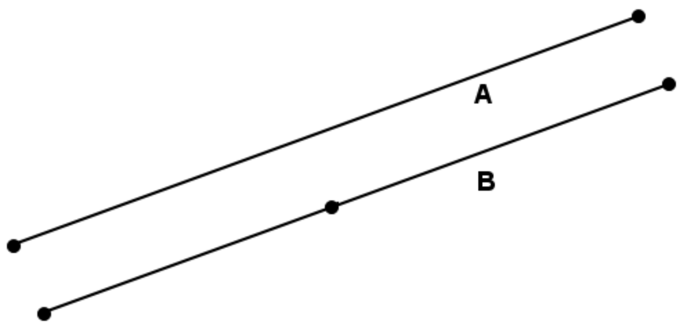


Figure 4.5: Example why the same number of vertices is needed for the non–negativity property. Both trajectories are translated for better illustration.

Proposition 8. $\delta_{\varphi_b}(T,U) = 0 \Rightarrow sem(T) = sem(U)$

Proof. Let T, $U \subset \mathbb{R}^2 \times \mathbb{T}$, D_φ as defined in Definition 4.2.6, $n_t \in \mathbb{N}$ the number of vertices of T and $n_u \in \mathbb{N}$ the number of vertices of U with $n_t > n_u$ and $\delta_{\varphi_b}(T,U) = 0$. We will now prove Proposition 8:

$\delta_{\varphi_b}(T,U) = 0 \Rightarrow sem(T) = sem(U)$

Since $n_t > n_u : \exists\, t \in T : t \notin U$, but because $\delta_{\varphi_b}(T,U) = 0$, we know that $D_\varphi(t,U) = 0$ and we can conclude that there exists a segment $S \in sem(U)$ which contains t and $S = sem(t_1, t, t_2)$ with t_1 the vertex directly preceding t and t_2 the vertex directly following t.

We can use the same idea for every $t \in T : t \notin U$ and thus we can conclude that: $sem(T) = sem(U)$.

□

Only one condition is missing to conclude that the Fréchet–based Distance for trajectories in $\mathbb{R}^2 \times \mathbb{T}$ is a metric: the triangle inequality. Unfortenately, this condition does not hold for our new distance measure, if we use the Euclidean Distance in $\mathbb{R}^2$, as can be seen in Figure 4.6. It is visually obvious that $\delta_{\varphi_b}(B,C) \geq \delta_{\varphi_b}(B,A) + \delta_{\varphi_b}(A,C)$.

Apparently, our new distance measure is not a metric if we allow the use of the Euclidean Distance in $\mathbb{R}^2$, but one could think that, if we add some restrictions on the trajectories, it is a metric. If we take a closer look at the problem, illustrated in Figure 4.6, we notice that the problem arises because of the timestamp. How can we solve this problem?

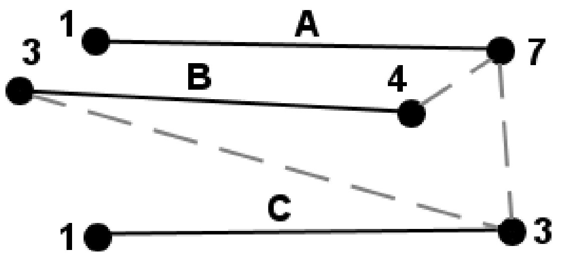


Figure 4.6: Example that the triangle inequality doesn't hold for the Fréchet–based Distance, usind the distance in $\mathbb{R}^2$. The grey lines are the found distances between the trajectories.

Translating the trajectories in time, does not solve the problem. Even when all the trajectories start at the same moment, the problem can still arise, as shown in Figure 4.7

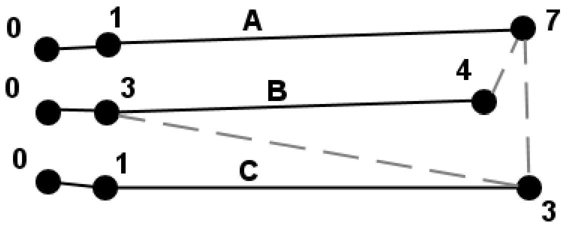


Figure 4.7: Example of a trajectory where the triangle inequality doesn't hold either, even with equal begintimes. The grey lines are the found distances between the trajectories.

By using the timevalue to match vertices, we create some kind of distortion in the spatial data, but maybe if we match the temporal data with a spatial dimension, we might be able to avoid this distortion. Since already use values from $\mathbb{R}$ for $\mathbb{T}$, as seen in Section 2.2, we can map every $t - value$ to a spatial value.

But unfortunately, this idea also doesn't work. Let's take a look at following trajectories A, B, $C \subset \mathbb{R}^2 \times \mathbb{T}$. $A = \langle (0,0,0), (10,20,3) (40,20,4) \rangle$, $B = \langle (0,0,0), (40,20,1) (70,10,7) \rangle$, $C = \langle (0,0,0), (20,30,2) (80,40,3) \rangle$. We have mapped all the t-values to spatial values, so we can use the distance in $\mathbb{R}^3$.

Note that all three trajectories start with the same vertex $(0,0,0)$. This is done to show that translating in any direction does not help to solve the problem.

Using the algorithm with the Euclidean Distance in $\mathbb{R}^3$, we find following values: $\delta_{\varphi_b}(A, C) = \sqrt{5300} \approx 72,80$, $\delta_{\varphi_b}(A, B) = \sqrt{1003} \approx 31,67$, $\delta_{\varphi_b}(B, C) = \sqrt{1004} \approx 31,68$. As we can see $\delta_{\varphi_b}(A, B) + \delta_{\varphi_b}(B, C) < \delta_{\varphi_b}(A, C)$ and thus the triangle inequality also does not hold when we use 3–Dimensional Distance instead.

Chapter 5

Similarity Search

We already explained what similarity search is in Chapter 1. Now we will take a look how it is done. We will start with a short note on similarity search with qualitative methods and we will try to link results from these methods with our results from the Fréchet–based Distance for trajectories.

Then, we will give a short explanation on similarity search with the Hausdorff and Fréchet Distance. Afterwards, we will see how we can use our new Fréchet–based Distance for similarity search. We will run different tests with this new measure and compair the results. For these tests, we have designed a new program *Algorithmtester*. Details of the program can be found in Section 5.4.1 and Appendix B.2.

5.1 Similarity search with the Double–Cross

For the qualitative methods, we will discuss similarity search with the Double–Cross method. For the Twisted–Cross, the principle is the same, only the calculation of the matrix differs.

If we use this method, we first calculate the Shape Matrix for every trajectory we like to compair. If we want to use a matrix–based distance, we need to make sure both matrices have the same dimension. This is not a problem if we remember this while generalizing the trajectories with the generalization principle seen in Section 3.1 and make sure both matrices have the same dimension.

An advantage of this method is that it's not influenced by scaling, rotation and translation, so there is no need of first translating or rotating the two objects we are going to compair to match as good as possible.

When we have the two matrices of the two figures, we need to use some measure of distance between the two matrices. Δ_H has been proposed in

[KMV06] and we will use the same method here.

Given two Double–Cross matrices M_1 and M_2, $N \times N$ matrices, construct for both vectors $\gamma(M_1), \gamma(M_2) \in \mathbb{N}^{65}$ that counts for each of the 65 realizable codes, the number of times they occur in M_1 and M_2. With this, we can use following formula:

$$\Delta_H(M_1, M_2) = \frac{1}{2}\sum_{i=0}^{65} |\gamma(M_1)[i] - \gamma(M_2)[i]|$$

$\Delta_H \in [0,1]$ and this number is interpreted as a degree of *dissimilarity.* To get a degree of *similarity*, we simple use $1 - \Delta_H(M_1, M_2)$ and multiplying this number with 100 gives us a percentile degree of similarity between two trajectories.

5.2 Similarity search using the Hausdorff Distance

Hausdorff Distance is mostly used to find specific figures, e.g. circles, in a complex figure. You take a basic startfigure, here a circle, and you calculate the Hausdorff Distance between this circle and any part of the larger image.

Note that it is allowed to translate or rotate one of the figures as needed to minimize the distance [HKR93]. In this case, we see similarity without the positions of the objects, we are only checking how much the general shape of two objects are alike.

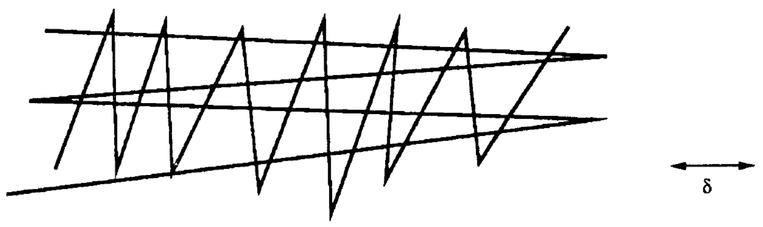


Figure 5.1: Two trajectories with Hausdorff Distance δ, although they are far from similar, they have a very little Hausdorff Distance [AG95].

Similarity search with this metric, is defined as followed: Given the Hausdorff Distance, determine a translation, scaling or rotation – or a combination of any of these – that minimizes the Hausdorff Distance between the two objects [AHSW03].

If the resulting distance is zero or smaller than a given threshold, the two figures are considered equal. If not, a smaller distance can be interpreted as a greater similarity, but the following example in Figure 5.1 will show that this is not necessarily the case.

The trajectories in Figure 5.1 are a excellent example why the Fréchet Distance is often used, despite it's long runtime, as seen in Section 4.2.1.

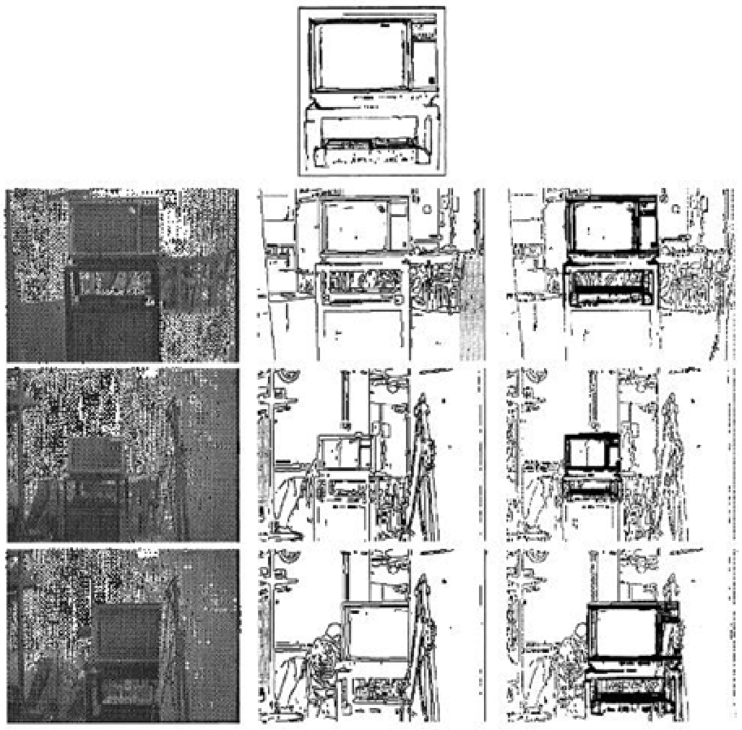


Figure 5.2: Example of the use of Hausdorff Distance in similarity search. At the top we see the image we are looking for, left a photograph, in the middle a picture derived from the photograph and on the right the drawn picture with a match [HKR93].

If we take a look at the definition of the Hausdorff Distance, seen in Section 4.1.3, we can easily find the reason of this error: we look for the *smallest* distance between two trajectories, with no regard to the orientation of the trajectories. To find a solution for this problem, the Fréchet Distance, studied in Section 5.3 is often used instead of the Hausdorff Distance.

As mentioned before in Section 4.1.4, we will only take a look at this basic concept of similarity search for the Hausdorff Metric. Other concepts of similarity can be examined in a similar way.

We will end our examination of the Hausdorff Distance with an example, found in Figure 5.2. The source here are three camera images that are digitalized and the Hausdorff Distance is used for edge detection: to find a television set on the picture. In this example, the Hausdorff Distance was used succesfully.

5.3 Similarity search using the Fréchet Distance

Similarity search with the Fréchet Distance is very much like similarity search with the Hausdorff Distance. In general, similarity search with this distance measure is considered a decision problem: is the Fréchet Distance between two objects smaller than a given threshold or not? [AKW01]

The idea behind similarity search, using the Fréchet Distance is quite simple: first a translation is sought to make the two objects fit as good as possible. After this, the Fréchet Distance between the two is computed and if it is smaller than a given threshold, they are considered equal. This is called the *decision problem for the Fréchet Distance.*

Sounds simple, but there is one large problem: the decision problem for the Fréchet Distance is *NP–Hard* for any dimension n with $n \geq 4$ [God98]. If it is also NP–Complete, remains a question. However, it should be possibile to find a specific algorithm for a specific metric space, as has been done in [BBW06].

Similarity search with the Fréchet Distance heavily depends on these case specific distance measures and on distance measures based on the Fréchet Distance, like *Discrete Fréchet Distance* [AHK$^+$06] or *Fréchet–based Distance for trajectories*, as seen in Section 4.2.6.

5.4 Similarity search with the Fréchet–based Distance

5.4.1 Testsoftware

For the tests in the following section, we have designed a new program named *Algorithmtester*. This program accepts trajectories as input and can then be used to visualise these, calculate the Fréchet–based Distance or Hausdorff Distance between these trajectories or run a k–means clustering [HK01] on this dataset, using a distance measure of your choice. It also measures the time to perform a requested operation and all output can be saved to a file. More details can be found in Appendix B.2

For every test, we have printed the time needed to perform the test, as measured by the tool, but the measured durations are only an indication since runtime greatly depends on the system that is used and the circumstances of the moment. The run time results we mention are with respect to a system with an Intel© Centrino© Duo 1.86 GHz and 2GB RAM running Microsoft Windows™XP Professional as operating system.

All graphics in the following part, are screenshots taken from the program Algorithmtester.

5.4.2 Testresults for clustering

If we like to use the Fréchet–based Distance for trajectories for similarity search, we need to test this distance measure. First we will use this distance measure with a k–means clustering algorithm and see how it works. Afterwards we will have a look at how it handles specific trajectories, like stepfunctions.

We are mostly interested in using this algorithm – and thus the new distance measure – for *similarity search*, as can be seen in Chapter 1. First we tried to see what results might come up if we used the distance measure in a clustering algorithm to cluster trajectories. We started with a dataset containing 250 trajectories, visualized in Figure 5.3. Note that colouring is done for no other reason than easier recognition of the individual trajectories.

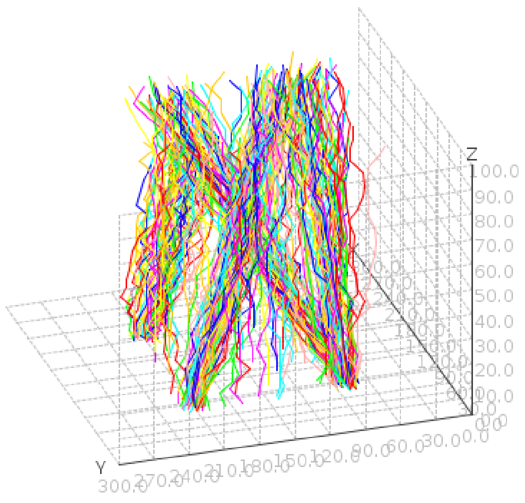


Figure 5.3: Visualisation of the clusterdata.

We used a classical implementation of a k–means clustering algorithm [HK01] for this process. Since k–means clustering is a well–known and well–used algorithm, no pseudocode is included in this work, but we will give a short explanation on the exact implementation used in our tool.

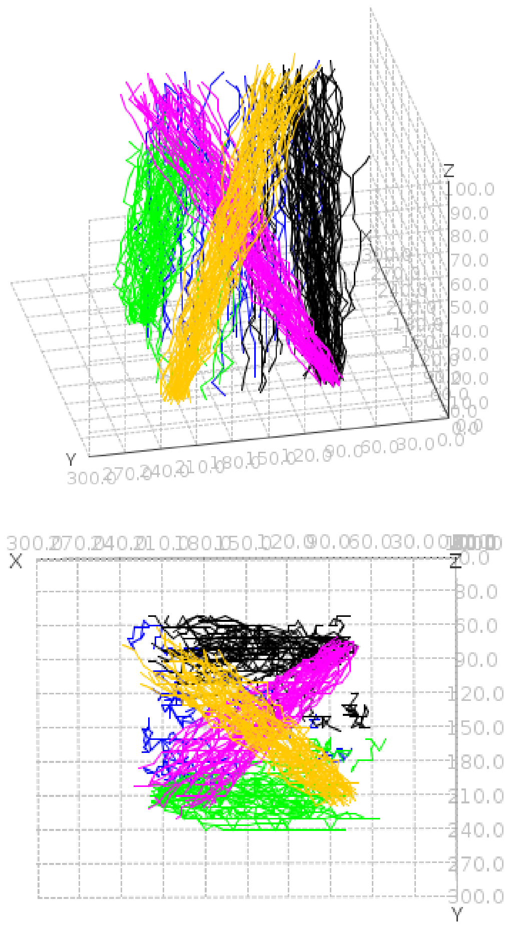


Figure 5.4: Result of the k–means clustering algorithm for 5 clusters.

The user inputs the trajectories he wants to cluster and the number of clusters k he wants to be calculated. Before the clustering algorithm starts, the program first calculates all the Fréchet–based Distances between all trajectories to save time and the need for recalculation.

The algorithm starts with assigning k random trajectories as clusterscentroids. Then for every trajectory, the algorithm will search for the centroid closest to the specific trajectory, using the Fréchet–based Distance for trajectories.

Every trajectory will then be classified in the cluster, corresponding to the centroid it is closest to. The algorithm then recomputes the centroids: in every cluster, it searches for the trajectory that has the shortest distance to all the other trajectories in the cluster. And all trajectories are reassigned to a cluster, using these new centroids. This process is repeated as long as changes occur.

For a first test, we used the dataset mentioned before and shown in Figure 5.3 and asked the program to divide the data in 5 clusters. The result took an average of 375ms to calculate and is shown in Figure 5.4. We can clearly see 4 visible clusters and, if we take a closer look, see that the fifth cluster are the trajectories that can be considered noise.

We then asked the program to divide the dataset in 6 clusters. The result took an average of 406.5ms to calculate and is shown in Figure 5.4. We can see the same 4 visible clusters in Figure 5.5 and notice that the "noise"–cluster is divided in two.

On this testdata, the algorithm seems to return excellent results if we use it to find clusters in a group of trajectories, with an excellent runtime lower than half a second.

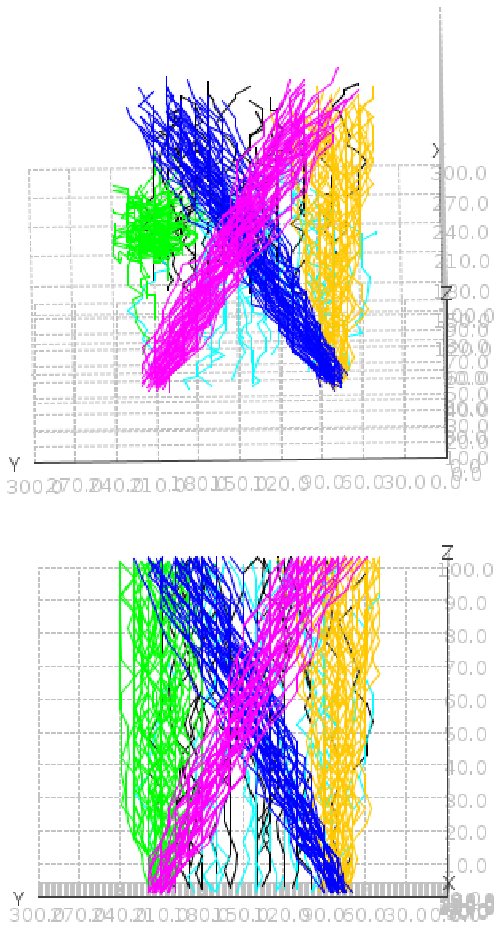


Figure 5.5: Result of the k–means clustering algorithm for 6 clusters.

5.4.3 Testresults with special trajectories

We have seen in Section 5.1 that similarity search using the Double–Cross — or Twisted–Cross – is invariant under translation, rotation and scaling. We will now run a small series of tests to see if this is also the case with our new Fréchet–based Distance.

First take a look at Figure 5.6, a plot with three trajectories. They are exactly the same, except that one of them is 1/10 and one is 1/5 the size of the largest one. Note that the trajectories on the plot in Figure 5.6, are translated for visualization purposes.

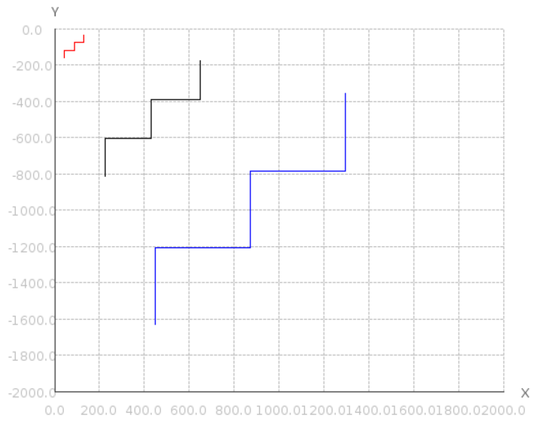
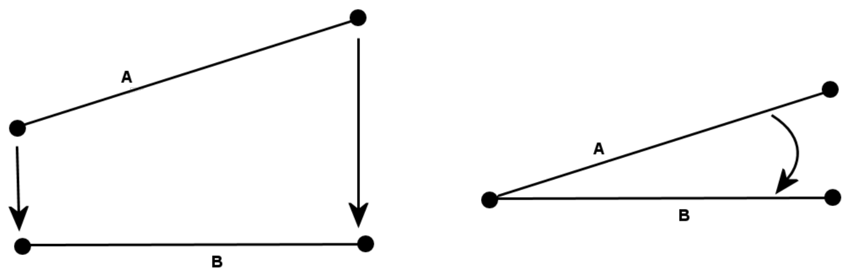


Figure 5.6: A trajectory and two smaller versions of the same trajectory at scale 1/10 and 1/5.

If we calculate the Fréchet–based Distance for these three trajectories and we calculate the Fréchet–based Distance between each trajectory and the line connecting it's begin– and endvertex, we will see that each trajectory is more similar to that line than to any of the other two trajectories.

Of course, your notion of similarity is very important here: does size matter or not? If we want to find similarity no matter the scale of our objects, than the Double–Cross has the advantage. We can add this functionality to our algorithm by scaling our trajectories to make them match as good as possible. In this case, we will see that after scaling, the three trajectories are considered exactly the same.

Analogue concepts can be used for translating and rotating trajectories to make them match as good as possible. But what is the perfect match? In our code, we have choosen to bring the startvertices of trajectory A and B together, shown in Figure 5.7(a), and than rotate one of the two trajectories,



(a) First the trajectories are translated to make the startvertices match.

(b) Then they are rotated to bring the endvertices as close to eachother as possible.

Figure 5.7: Illustration of the used techniques to bring two trajectories together.

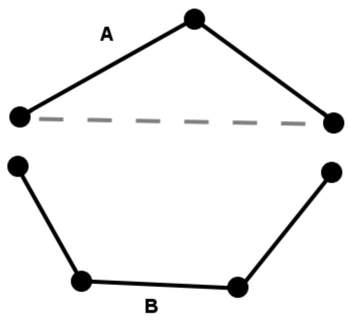
A, to make the endvertex of A lie on the straight line between the begin– and endvertex of B, shown in Figure 5.7(b).

Our technique is very fast, since only two transformations are necessary: a translation and a rotation. But this does not give us the ideal match. It is very possible that two trajectories would have a smaller distance with another transformation. This can be seen in Figure 5.8.

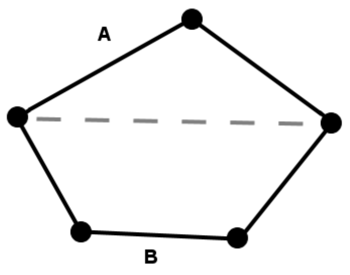
Figure 5.8(a) shows the two trajectories, A and B, that we would like to match. Using our own proposition, we find the match shown in Figure 5.8(b). But if we take a look at Figure 5.8(c), we see that this third image contains a matching that gives us a smaller Fréchet–based Distance than the second one.

Finding the ideal transformation to minimize the distance between two trajectories or polylines is not an easy task. In [AKW01] a technique is proposed to find this ideal transformation, but it only works in a fixed direction and it only works with transformations (rotations and scaling is not possible).

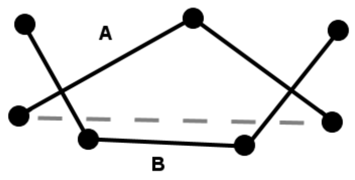
The run time of this algorithm is $O\left((mn)^3\,(m+n)^2\right)$, while our proposal is a lot faster: we perform a single translation and – if necessary – a single rotation. Essentially, our idea is less correct but much faster.



(a) Two trajectories we wish to match as good as possible.



(b) The two trajectories matched with our algorithm.



(c) A match that results in a smaller Fréchet–based Distance between the two trajectories.

Figure 5.8: Illustration of the main disadvantage of our proposed technique.

Chapter 6

Conclusions

Here we will end our study of qualitative and quantitative methods for similarity search. Originally, there would have been a greater emphasis on the differences and similarities between these two classes of methods, but our newfound distance measure, the Fréchet–based Distance, appeared to be very interesting and thus the focus of this thesis moved towards a study of this new method.

I would like to complete my work with a reflection on our findings and a look at the direction future research in this method might go to.

6.1 General conclusions

We have taken a look at the generalization principle and the qualitative methods it was originally designed for. We have seen that it is a necessity for the qualitative methods, but that it also can be used for our new distance measure to compare curved trajectories and that is solves the problems that might arise with a boundary based approach.

We have seen the Hausdorff Distance and the Fréchet Distance and we followed the classical idea for the Fréchet Distance to find a new distance measure for our own type of data: create a measure based on the Fréchet idea especially designed for our datatype.

Although our proposed distance measure is not a metric, it does work very well with clustering techniques and it is very fast. We have found times smaller than half a second for clustering with excellent results.

An other advantage is that the new measure can be used for trajectories in an space: since we use the Euclidean Distance. We can use this algorithm for trajectories in $\mathbb{R}^3 \times \mathbb{T}$ as well. An example of this kind of data could be the route travelled by a helicopter.

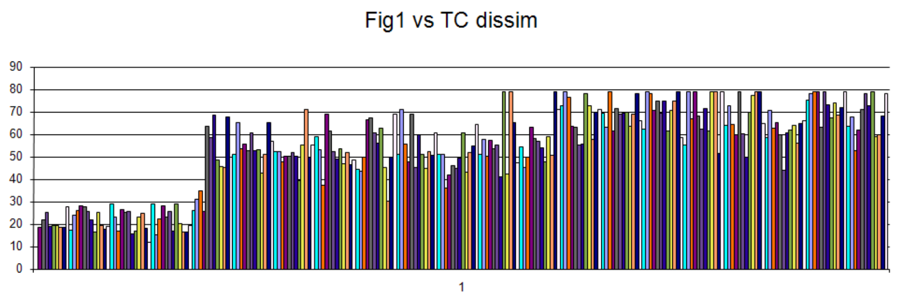
6.2 Further research

Not all secrets of the Fréchet–based Distance have been discovered and many questions remain. The original intention of this work was to link qualitative and quantitative distance measures. We already explained why the goal of this work changed, but the original question is not less interesting because of our findings, rather the opposite.

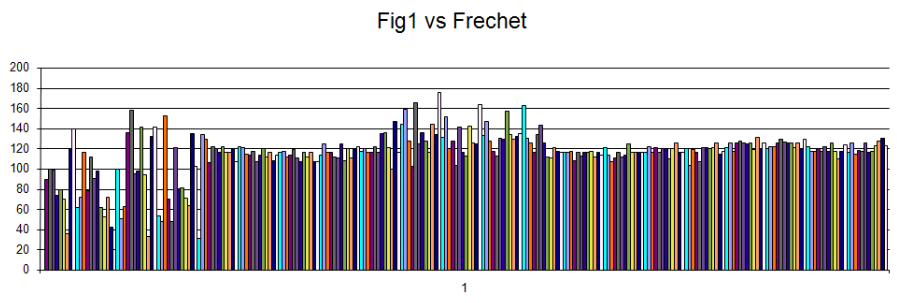
Since we now use exactly the same datatype – generalized trajectories – the search for similarities might become easier and more interesting. Is there a relation between the two techniques? Which one is better? When is it better and why? This are a few examples of questions that can be answered in the future.

We do have a feeling that there is a connection between the new Fréchet–based Distance and the Twisted–Cross method. As we take a look at Figure 6.1, we see that all trajectories with a Fréchet–based Distance smaller than 100 are in contained in the set of all the trajectories with Twisted–Cross dissimilarity 30, with other words: if the Fréchet–based principle finds it very similar, the Twisted–Cross does the same.

This needs to be researched in more detail, to check if this is a coincidence or something that is true. If this always holds, the Twisted–Cross can be used as a preprocessing step for the Fréchet–based distance.



(a) Graphical representation of Twisted–Cross results



(b) Graphical representation of Fréchet–based results

Figure 6.1: Illustration of a relation between the Twisted–Cross and the Fŕechet–based Distance

Bibliography

[AG95] Helmut Alt and Michael Godau. Computing the fréchet distance between two polygonal curves. *Int. J. Comput. Geometry Appl*, 5:75–91, 1995.

[AHK$^+$06] Boris Aronov, Sariel Har–Peled, Christian Knauer, Yusu Wang, and Carola Wenk. Fréchet distance for curves, revisited. In Yossi Azar and Thomas Erlebach, editors, *ESA*, volume 4168 of *Lecture Notes in Computer Science*, pages 52–63. Springer, 2006.

[AHSW03] Pankaj K. Agarwal, Sariel Har–Peled, Micha Sharir, and Yusu Wang. Hausdorff distance under translation for points and balls. In *Symposium on Computational Geometry*, pages 282–291. ACM, 2003.

[AKW01] Helmut Alt, Christian Knauer, and Carola Wenk. Matching polygonal curves with respect to the fréchet distance. In Afonso Ferreira and Horst Reichel, editors, *STACS*, volume 2010 of *Lecture Notes in Computer Science*, pages 63–74. Springer, 2001.

[Bar] Margherita Barile. Pseudometric. `http://mathworld.wolfram.com/Pseudometric.html` (Last visited: 08–05–2007).

[BBW06] Kevin Buchin, Maike Buchin, and Carola Wenk. Computing the fréchet distance between simple polygons in polynomial time. In Nina Amenta and Otfried Cheong, editors, *Symposium on Computational Geometry*, pages 80–87. ACM, 2006.

[BFM$^+$96a] J. E. Barros, J. C. French, W. N. Martin, P. M. Kelly, and T. M. Cannon. Using the triangle inequality to reduce the number of comparisons required for similarity-based retrieval. In I. K. Sethi and R. C. Jain, editors, *Proc. SPIE Vol. 2670, p. 392-403, Storage and Retrieval for Still Image and Video Databases IV, Ishwar K. Sethi; Ramesh C. Jain; Eds.*, volume 2670 of *Presented at the Society of Photo-Optical Instrumentation Engineers (SPIE) Conference*, pages 392–403, March 1996.

[BFM$^+$96b] Julio Barros, James French, Worthy Martin, Patrick Kelly, and Mike Cannon. Using the triangle inequality to reduce the number of comparisons required for similarity-based retrieval, 1996.

[Bla] Paul E. Black. Hausdorff distance. `http://www.nist.gov/dads/HTML/hausdorffdst.html` (Last visited: 08–05–2007).

[BPSW05] Sotiris Brakatsoulas, Dieter Pfoser, Randall Salas, and Carola Wenk. On map-matching vehicle tracking data. In Klemens Böhm, Christian S. Jensen, Laura M. Haas, Martin L. Kersten, Per-Åke Larson, and Beng Chin Ooi, editors, *VLDB*, pages 853–864. ACM, 2005.

[Ewi85] George M. Ewing. *Calculus of variations with applications.* Dover Publications, 1985.

[geo] Geographic privacy–aware knowledge discovery and delivery. `http://www.geopkdd.eu/` (Last visited: 28–05–2007).

[God98] Michael Godau. *On the complexity of measuring the similarity between geometric objects in higher dimensions*, chapter 3, pages 19–20, 33–59. 1998.

[Hen99] Jeff Henrikson. Completeness and total boundedness of the hausdorff metric. *Massachusetts Institute of Technology Undergraduate Journal*, pages 70–72, 1999.

[HK01] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques*, chapter 8, pages 349–351. Academic Press, 2001.

[HKR93] Daniel P. Huttenlocher, Gregory A. Klanderman, and William Rucklidge. Comparing images using the hausdorff distance. *IEEE Trans. Pattern Anal. Mach. Intell*, 15(9):850–863, 1993.

[Hoo06] Jelle Van Hoof. Masterstage - simple shapetool: verslag. Technical report, Hasselt University, 2006.

[Kel06] Tom Kellens. Database search van tijdreeksen, met toepassing in de firma medtronic. Master's thesis, Hasselt University, 2006.

[KM06] Bart Kuijpers and Bart Moelans. Similarity between trajectories by using a qualitative feature extraction approach. Technical report, Hasselt University, 2006.

[KMV06] Bart Kuijpers, Bart Moelans, and Nico Van de Weghe. Qualitative polyline similarity testing with applications to query-by-sketch, indexing and classification. In Rolf A. de By and Silvia Nittel, editors, *GIS*, pages 11–18. ACM, 2006.

[KYM+05] Kikuya Kato, Riu Yamashita, Ryo Matoba, Morito Monden, Shinzaburo Noguchi, Toshihisa Takagi, and Kenta Nakai. Cancer gene expression database (CGED): a database for gene expression profiling with accompanying clinical information of human cancer tissues. *Nucleic Acids Research*, 33(Database-Issue):533–536, 2005.

[NCV+03] Melanie Nugoli, Paul Chuchana, Julie Vendrell, Beatrice Orsetti, Lisa Ursule, Catherine Nguyen, Daniel Birnbaum, Emmanuel Douzery, Pascale Cohen, and Charles Theillet. Genetic variability in mcf-7 sublines: evidence of rapid genomic and rna expression profile modifications. *BMC Cancer*, 3(1):13, 2003.

[VDKD05] Nico Van de Weghe, Guy De Tré, Bart Kuijpers, and Philippe De Maeyer. The double-cross and the generalization concept as a basis for representing and comparing shapes of polylines. In *OTM Workshops*, volume 3762 of *Lecture Notes in Computer Science*, pages 1087–1096. Springer, 2005.

[Wei] Eric W. Weisstein. Metric. `http://mathworld.wolfram.com/Metric.html` (Last visited: 08–05–2007).

Appendix A

Double–Cross codes

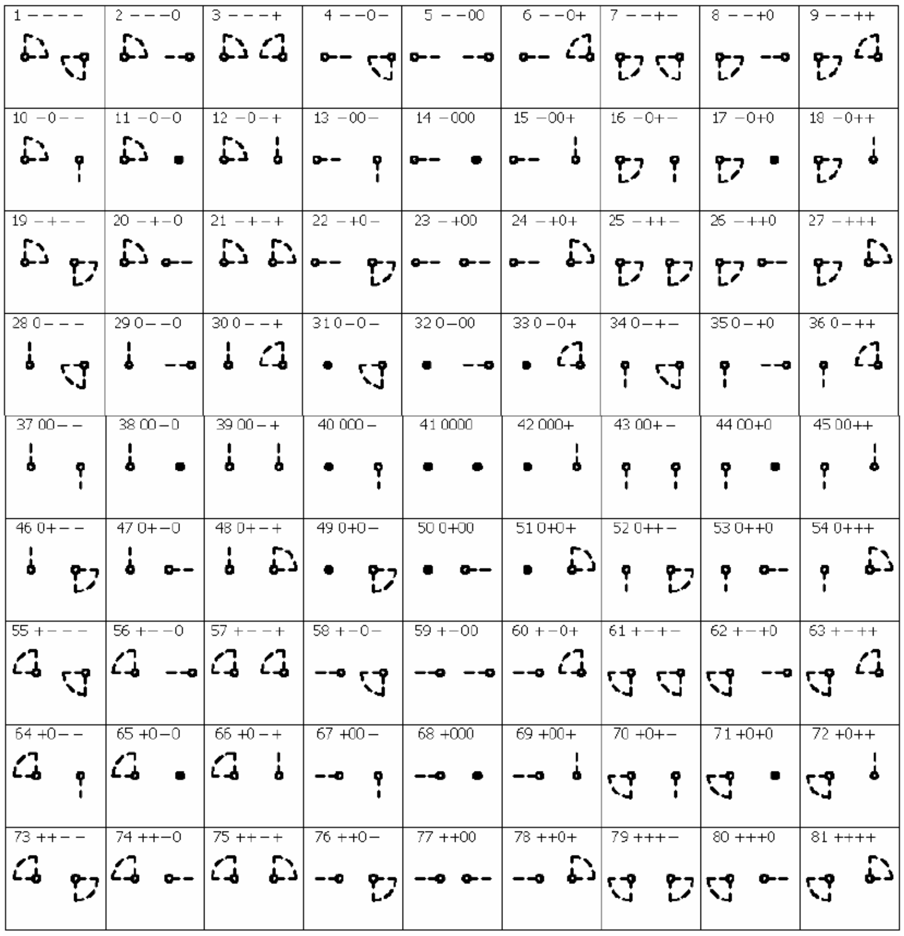


Figure A.1: List of all possible codes for the Double–Cross. Note that only 65 of these codes are possible in reality. [VDKD05]

Appendix B

Used tools

I have developed two tools have been used in this work: one that has been developed during an internship at Hasselt University and one during the creation of this work. The first one is called *Simple Shape Tool (SST)*, discussed in Section B.1 and the other one *Algorithmtester*, which can be found in Section B.2.

B.1 Simple Shape Tool

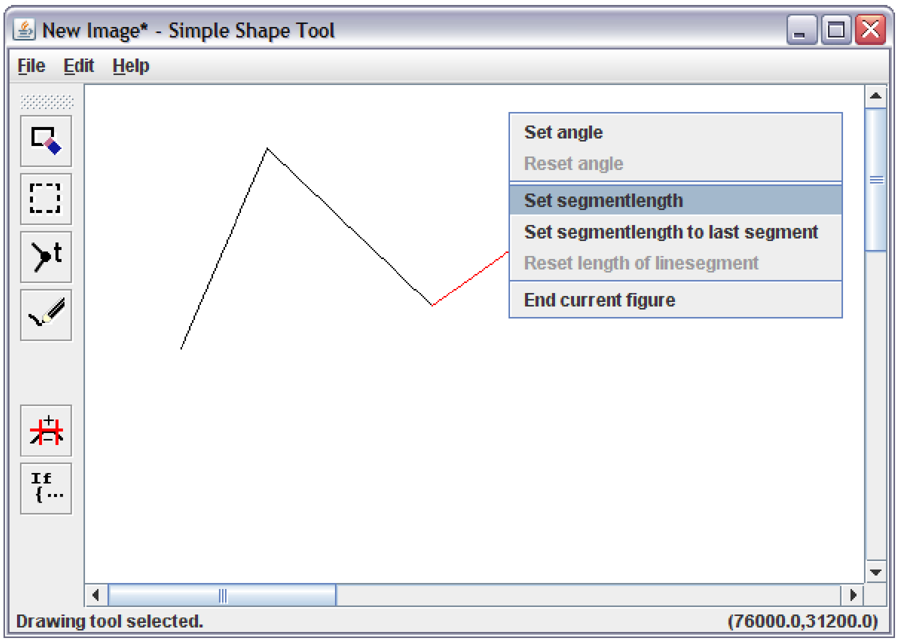


Figure B.1: Screenshot of the Simple Shape Tool

The Simple Shape Tool has been developed for researchproject R–0920, a project from Hasselt University and Ghent University. A screenshot can be found in Figure B.1.

The first idea behind the program was to create a very easy to use, platform independant, tool for drawing polylines, polygons and trajectories, with the possibility to draw these figures with fixed angles or fixed segmentlenghts. The program is written in Java.

The second idea was to create a tool that could be used to test different algorithms with a newly drawn figure. The program contains a small editor to make it possible for the user to insert an algorithm in the program that can be used.

For example, a user could write the algorithm for calculating the simple cross codes and then immediatly use this algorithm to find the corresponding codes for a figure. Inserted algorithms are saved in a special folder so they can be reused. More details on this program can be found in [Hoo06].

B.2 Algorithmtester

Algorithmtester is a small program, written in Java, for vizualisation – plotting – of trajectories in either 2D or 3D and for testing the different distance measures encountered in this work. Screenshots can be found in Figure B.2 and Figure B.3.

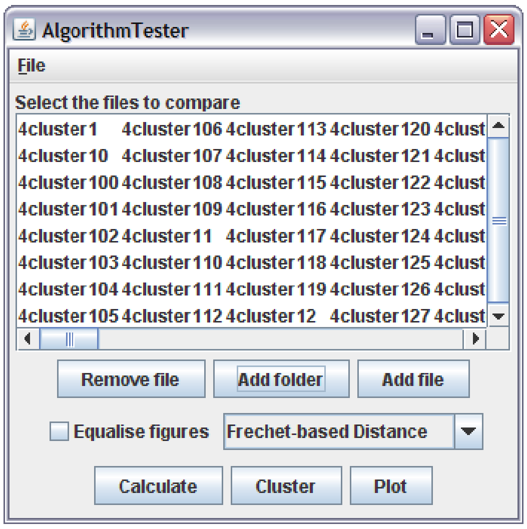
Figure B.2 shows the very simple main interface of the program, the purpose of the buttons and such is very obvious and thus not explained. Except for the option "Equalise figures": when a user checks this box, the program will use translations, rotations or scaling to make every pair of trajectories match as good as possible, before using the selected distance measure.

The program accepts files in the format used by the Simple Shape Tool as inputtrajectories.

In Figure B.3, we see the resulting distance between each pair of trajectories, after calculations. Since the Fréchet–based Distance is symmetric, we only calculate the distances between every pair once. These distances can be saved in a file with comma-separated values, which can be opened with any popular program for spreadsheets. The time to calculate the values is also given, for test purposes.

Examples of the plots generated by the program can be found in Figure 5.3, Figure 5.4 and Figure 5.5. To generate these plots, we have used a simple opensource extension for Java: JMathPlot[1].

In the end, some extra functionality was added: the possibility to cluster a group of trajectories, using a distance measure available in the program. For this option, we used a k–means algorithm, described in Section 5.4.2.

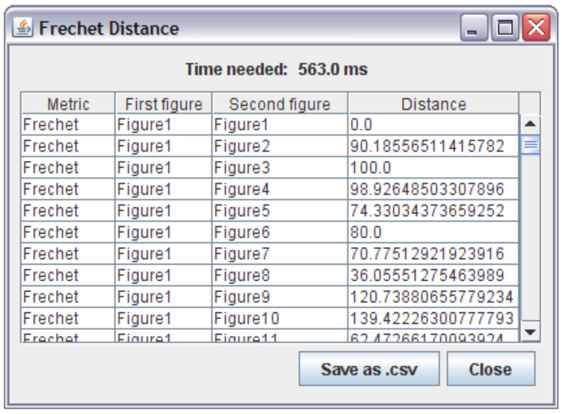


Select the files to compare

4cluster1	4cluster106	4cluster113	4cluster120	4clust
4cluster10	4cluster107	4cluster114	4cluster121	4clust
4cluster100	4cluster108	4cluster115	4cluster122	4clust
4cluster101	4cluster109	4cluster116	4cluster123	4clust
4cluster102	4cluster11	4cluster117	4cluster124	4clust
4cluster103	4cluster110	4cluster118	4cluster125	4clust
4cluster104	4cluster111	4cluster119	4cluster126	4clust
4cluster105	4cluster112	4cluster12	4cluster127	4clust

☐ Equalise figures — Frechet-based Distance

Figure B.2: Main interface of the program



Time needed: 563.0 ms

Metric	First figure	Second figure	Distance
Frechet	Figure1	Figure1	0.0
Frechet	Figure1	Figure2	90.18556511415782
Frechet	Figure1	Figure3	100.0
Frechet	Figure1	Figure4	98.92648503307896
Frechet	Figure1	Figure5	74.33034373659252
Frechet	Figure1	Figure6	80.0
Frechet	Figure1	Figure7	70.77512921923916
Frechet	Figure1	Figure8	36.05551275463989
Frechet	Figure1	Figure9	120.73880655779234
Frechet	Figure1	Figure10	139.42226300777793
Frechet	Figure1	Figure11	62.47266170093924

Figure B.3: Resultwindow of the program

[1]`http://jmathtools.sourceforge.net/`