Analyzing Trajectories using Uncertainty and Background Information

Bart Kuijpers, Bart Moelans, Walied Othman, and Alejandro Vaisman

Hasselt University & Transnational University of Limburg, Belgium
{bart.kuijpers,bart.moelans,
walied.othman,alejandro.vaisman}@uhasselt.be

Abstract. A key issue in clustering data, regardless the algorithm used, is the definition of a distance function. In the case of trajectory data, different distance functions have been proposed, with different degrees of complexity. All these measures assume that trajectories are error-free, which is essentially not true. Uncertainty is present in trajectory data, which is usually obtained through a series of GPS of GSM observations. Trajectories are then reconstructed, typically using linear interpolation. A first source of error in trajectory are the GPS observations themselves, since many reported points lie outside the road network. Thus, the users position must be matched to a map, leading to the problem of *map matching.* A well-known model to deal with *uncertainty* in a trajectory sample, uses the notion of *space-time prisms* (also called *beads*), to estimate the positions where the object could have been, given a maximum speed. Thus, we can replace a (reconstructed) trajectory by a necklace (intuitively, a *a chain of prisms*), connecting consecutive trajectory sample points. When it comes to clustering, the notion of uncertainty requires appropriate distance functions. The main contribution of this paper is the definition of a distance function that accounts for uncertainty, together with the proof that this function is also a metric, and therefore it can be used in clustering. We also present an algorithm that computes the distance between the chains of prisms corresponding to two trajectory samples. Finally, we discuss some preliminary results, obtained clustering a set of trajectories of cars in the center of the city of Milan, using the distance function introduced in this paper.

1 Introduction

Moving object databases The study of *Moving Object Databases* (MODs) [7,21] has been increasingly attracting the attention of the GIS (Geographic Information Systems) community. Most of the time, in this field, a moving object's trajectory is obtained from *trajectory samples*, i.e., finite sequences of time-space points. A *trajectory sample database* contains a finite number of labeled trajectory samples. There are various ways of reconstructing trajectories from trajectory samples, of which linear interpolation is the most popular one [7]. An important issue in these databases is the problem of *uncertainty*, arising from various sources (e.g., errors in measurements, interpolation). The uncertainty of

the moving object's position in between sample points has been studied using *space-time prisms* or *beads*, where it is assumed that besides the time-stamped locations of the object also some *background* knowledge is known, like, for instance, a speed limit v_{max} at a location $(\mathsf{x}_i, \mathsf{y}_i)$. Informally, the space-time prism between two consecutive sample points is defined as the collection of space-time points where the moving objects may have passed, given the speed limitation.

Clustering One of the most common data mining techniques is Clustering [9]. This technique partitions the dataset into collections of data objects, such that within each partition the objects are 'similar' to each other and 'different' from the objects contained in other partitions. In the context of moving object data, the clustering technique is aimed at identifying groups of objects that followed similar trajectories. These kinds of data present particular problems for clustering. Clustering moving object trajectories requires, for example, finding out a proper spatial granularity level, and it is not obvious to identify the best clustering algorithm among the wide corpus of work on the subject. In the presence of uncertainty, the problem of representing a trajectory of moving objects and formalizing the notion of trajectory similarity is even more involved. This issue takes us to the problem we address in this paper: studying a distance function that accounts properly for the notion of uncertainty.

1.1 Problem statement and contributions

There exist two classic approaches to trajectory clustering: one based on the notion of similarity, typically operationalized through a so-called *distance function* between trajectories. A second approach is denoted trajectory-specific, and exploits the characteristics of the data type in the clustering algorithm. In this paper we position ourselves in the first group: we study the impact over clustering of the uncertainty of trajectory samples, by means of the definition of a distance function that accounts for the uncertainty involved in a trajectory. In short, a distance function measures the similarity of two trajectories. Many different distance functions can be defined (we give a formal definition, and a review, later in the paper), ranging from the most simple ones (like, for instance, clustering trajectories with the same origin and/or destination), to very complex mathematical functions. The former are obviously more computationally efficient, probably at the expense of returning less reliable clusters.

A well-known model to deal with *uncertainty* in a trajectory sample, uses the notion of *space-time prisms* (also called *beads*), to estimate the positions where the object could have been, given a maximum speed. We therefore introduce a distance function that accounts for uncertainty, and prove that this function is a metric which can be used to cluster trajectory (and hence, uncertain) data. Given two trajectory samples $T1$ and $T2$, their uncertainty is represented by two *chains of space-time prisms* (also called *lifeline necklaces*), N_1 and N_2, respectively, that connect consecutive sample points of each trajectory. Intuitively, in our proposal, the largest the intersection of the necklaces with respect to their union, the smallest the distance between both trajectories. In other words, the more

uncertainty shared by $T1$ and $T2$, the closer they are. On the other hand, if N_1 and N_2 do not intersect, this indicates that these trajectories could not have met, given the speed limit. Then, a clustering algorithm will not group together these two trajectories. We also present an algorithm that, given the *chains of space-time prisms* of two trajectories, computes the distance between them. Finally, we present preliminary experiments, clustering a set of data corresponding to movement of cars in the center of the city of Milan, using the distance function we introduce in this paper.

In Section 2 we review related work on clustering and space-time prisms. Section 3 introduces the concepts of space-time prisms and their relation with uncertainty and background information. In Section 4 we introduce the new distance function, denoted d_u, and we show it is a metric apt for trajectory clustering. Section 5 presents an algorithm to compute d_u. Section 6 presents preliminary experimental results. We conclude in Section 7.

2 Related work

Space-time prims and uncertainty In this paper we work with *trajectory samples*, which are well-known in MODs, namely finite sequences of time-space points. A trajectory sample database contains a finite number of labeled trajectory samples. There are various ways of reconstructing trajectories from samples, of which linear interpolation is the most popular in the literature [7]. However, linear interpolation relies on the (rather unrealistic) assumption that between sample points, an object moves at constant minimal speed. It is more realistic to assume that moving objects have some physically determined speed bounds. Given such upper bounds, *an uncertainty model* has been proposed which constructs *space-time prisms* between two consecutive space-time points in a trajectory sample. Basic properties of this model were discussed a few years ago by Egenhofer et al. [3] and Pfoser et al. [17], but space-time prisms were already known in the time-geography of Hägerstrand in the 1970s [8]. In short, a *space-time prism* is the intersection of two cones in the space-time space such that all possible trajectories of the moving object between the two consecutive space-time points, given the speed bound, are located within them. Space-time prisms manage uncertainty more efficiently than other approaches based on cylinders [21]. A chain of space-time prisms connecting consecutive trajectory sample points is called a *lifeline necklace* [3].

Trajectory clustering As we mentioned above, there are mainly two approaches to the problem of trajectory clustering: one is the classic distance-based, the other is denoted trajectory-specific clustering [16]. We first comment on the latter approach.

In one of the first works on trajectory clustering, Ketterlin [10] presents an structured methodology for discovering patterns in sequences of composite objects. These objects, basically time-series data, can be considered an abstract representation of a trajectory (i.e., a composite object may be described as a

sequence of simpler data). The author studies the generalization of sequences of complex objects, and integrate this in a general-purpose clustering algorithm. This generalization-based approach, together with the limitation to time-series, could be considered as shortcomings of this first approach.

Another proposal applies to trajectories some multidimensional scaling technique for non-vectorial data. This is performed in Fastmap [5], where a data space is mapped to an Euclidean space approximately preserving the distances between objects. Then, any standard clustering algorithm for vectorial data can be applied.

Gaffney and Smyth [6] use a different approach, denoted model-based clustering. They work with continuous trajectories, grouping together objects which likely to be generated from a common core trajectory, by adding Gaussian noise. In this way, a cluster contains all objects which can be obtained by a regression function. The problem of this approach is the lack of flexibility for different application contexts.

Typically, the *distance* between two trajectories is computed by fixing two time instants and considering points within this interval. Clustering trajectory data usually produces clusters containing geographically close trajectories. A classical approach for clustering trajectories, is to adapt traditional algorithms like k-means or hierarchical clustering, to the trajectory setting, leading to the notion of *distance-based clustering.*

Nanni [14] defines a distance measure that describes the similarity of trajectories of objects across time, computed by analyzing the way the distance between the objects varies. He considers only pairs of contemporary instantiations of objects, i.e., for each time instant compares the positions of the objects at that moment, aggregating the set of distance values obtained this way. The distance between trajectories is computed as the average distance between objects:

$$D(\tau_1;\tau_2) \mid T = \frac{\int_T \ d(\tau_1(t);\tau_2(t))dt}{\mid T \mid}$$

where $d()$ is the Euclidean distance over R^2, T is the temporal interval over which trajectories τ_1 and τ_2 exist, and $\tau_i(t)$ is the position of object τ_i at time t. This is the more general expression. However, the author showed that due to the piece-wise linearity of the trajectories, the distance can be computed as a finite sum by means of $O(n1 + n2)$ Euclidean distances, where $n1$ and $n2$ are the number of observations for τ_1 and τ_2.

Other proposals compute the longest common subsequence of two series [19], and the least common sub-sequence of two series, and take these measures as the distance between trajectories [1].

A different line of work is *Density-based clustering*, which can be considered as a combination of the two approaches mentioned above. Initially proposed by Ester *et al.* [4], clusters are populated by objects which can reach each other through densely populated regions, instead of objects close to each other. In other words, these algorithms agglomerate objects in clusters based on the population within a given region. This form of clustering is used in the OPTICS algorithm,

proposed by Ankerst *et al.* [2]. It is particularly well-suited to trajectory clustering, given that trajectories of cars in urban traffic tend to agglomerate in non-convex clusters, and that many outlier trajectories should be considered as noise. Nanni and Pedreschi [15] generalize the spatial notion of distance between objects to a spatio-temporal notion of distance between trajectories, leading to a natural extension of the density-based clustering technique to trajectories. More recently, the concept of *progressive clustering* has been introduced, as a process that, using a visual analytics approach, starts from the simpler functions to complex ones, in an incremental way [18], using an iterative approach that filters clusters using simpler but efficient distance functions in the firsts steps.

3 A model for moving object data with uncertainty

A well-known model for the management of the uncertainty of the moving object's position in between sample points is the *space-time prism* model, where it is assumed that besides the time-stamped locations of the object, also some background knowledge, in particular a (e.g., physically or law imposed) speed limitation v_i at location (x_i, y_i) is known. The space-time prism between two consecutive sample points is defined as the set of time-space points where the moving objects may have passed, respecting the speed limitation. The chain of space-time prisms connecting consecutive trajectory sample points is called a *lifeline necklace* [3].

In this paper we focus on space-time prisms on *road networks*. Early adaptations of the space-time prism model to road networks were done by Miller [12,13]. We view road networks as a graph embedding in $\mathbf{R}^2$ where all edges are embedded as straight lines between vertices. All edges have a (strictly positive) speed limit as well an associated weight, called their *time span*, which is equal to the time needed to get from one end of the edge to the other when traveling at the speed limit. Also Kuijpers and Othman [11] studied the problem of space-time prisms on road networks, and introduced an algorithm for computing and visualizing space-time prisms. In Section 5 we use this algorithm to compute the surface of a space-time prism.

In the remainder of the paper we work with the speed limits of the road network, simply setting a uniform speed limit, namely v_i, on the network to construct the space-time prism between two sample times t_i and t_{i+1}. We first review basic concepts about trajectories and road networks, that we use throughout the paper.

3.1 Preliminaries: trajectories and trajectory samples

Let $\mathbf{R}$ denote the set of the real numbers and $\mathbf{R}^2$ the 2-dimensional real plane. We consider objects moving in a subset of the two-dimensional (x, y)-space $\mathbf{R}^2$ and describe this movement in the (t, x, y)-space $\mathbf{R}\times\mathbf{R}^2$, where t represents time. Moving objects, which we assume to be points, produce a special kind of curves, denoted *trajectories*. The definitions we present next, necessary to understand the remainder of the paper, are based on [11].

Definition 1. [Trajectory] Let $I \subseteq \mathbf{R}$ be an interval. A *trajectory* T is the graph of a mapping $\alpha : I \to \mathbf{R}^2 : t \mapsto \alpha(t) = (\alpha_x(t), \alpha_y(t))$, i.e., $T = \{(t, \alpha_x(t), \alpha_y(t)) \in \mathbf{R} \times \mathbf{R}^2 \mid t \in I\}$. We call I the *time domain* of T. □

In practice, trajectories are only known at discrete moments in time. This partial knowledge of trajectories is formalized by the notion of sample.

Definition 2. [Trajectory sample] A *trajectory sample* is a finite set $S = \{(t_0, x_0, y_0), (t_1, x_1, y_1), ..., (t_N, x_N, y_N)\}$ of time-space points. The order on time, $t_0 < t_1 < \cdots < t_N$, induces a natural order on the sample. □

A trajectory T, which contains a trajectory sample $S = \{(t_0, x_0, y_0), (t_1, x_1, y_1), ..., (t_N, x_N, y_N)\}$, i.e., $(t_i, \alpha_x(t_i), \alpha_y(t_i)) = (t_i, x_i, y_i)$ for $i = 0, ..., N$, is called a *geospatial lifeline* for S [3].

In this paper, we consider movement in $\mathbf{R}^2$ that is constrained to a road network. Thus, we need to formalize the notion of a road network.

Definition 3. [Road network] A *road network* RN is a graph embedding in $\mathbf{R}^2$ of a labeled graph given by a finite set of vertices $\mathsf{V} = \{(x_i, y_i) \in \mathbf{R}^2 \mid i = 1, \ldots, N\}$, and a set of edges $\mathsf{E} \subseteq \mathsf{V} \times \mathsf{V}$ that are labeled by a *speed limit* and an associated *time span*. This graph embedding satisfies the following conditions. Edges are embedded as straight line segments between vertices[1]. If an edge between (x_i, y_i) and (x_j, y_j) is labeled by the speed limit $v_{ij} > 0$, then its time span w_{ij} is $\frac{\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}}{v_{ij}}$, i.e., it is the time needed to get from one side of an edge to another when traveling at the speed limit. □

A trajectory (sample) on a road network RN is then a trajectory (sample) whose spatial projection is in RN. More formally, if T is a trajectory given by the functions α_x and α_y, then it must satisfy $(\alpha_x(t), \alpha_y(t)) \in \mathsf{RN}$ for all t in the time domain of T, and for a trajectory sample $S = \{(t_0, x_0, y_0), (t_1, x_1, y_1), ..., (t_N, x_N, y_N)\}$ we must have $(x_i, y_i) \in \mathsf{RN}$ for all $i = 0, \ldots, N$.

3.2 Space-time prisms in road networks

Given a point in a trajectory sample, if we assume that a speed limit is valid until the next point, we can use this limit to define space-time prisms which model the uncertainty of the object's location in between sample points. In this paper, we consider movement and space-time prisms in road networks in $\mathbf{R}^2$.

In general, suppose p and q are points in some space, and an object is traveling from p to q, leaving p at time t_p and arriving in q at t_q; also assume a speed limit $v_{\max}$ is given. We know that at a time t, $t_p \leq t \leq t_q$, the object's distance to p is at most $v_{\max}(t - t_p)$ and its distance to q is at most $v_{\max}(t_q - t)$. The object is therefore somewhere in the intersection of the sphere with center p and radius $v_{\max}(t - t_p)$ and the sphere with center q and radius $v_{\max}(t_q - t)$. This is illustrated in Figure 1. The geometric location of these points, for $t_p \leq t \leq t_q$, is referred to as a *space-time prism*.

[1] These edge embeddings may intersect in non-vertex points. So, we can model bridges and tunnels in our model.

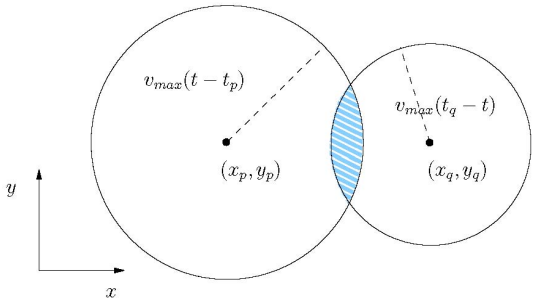

Fig. 1. A moving object's location at time t.

Although we are interested in space-time prisms on a road network, the problem does not simply amount to taking the intersection of a space-time prism representing unconstrained movement and the road network. To see this, consider the projection of the unconstrained space-time prism along the time axis onto the xy-plane. This projection is an ellipse such that its foci are the points of departure and arrival, i.e., p and q. We recall that at a time t, $t_p \leq t \leq t_q$, the object's distance to p is at most $v_{\max}(t - t_p)$ and its distance to q is at most $v_{\max}(t_q - t)$. Adding those distances gives $v_{\max}(t - t_p) + v_{\max}(t_q - t) = v_{\max}(t_q - t_p)$, which is constant. Therefore, all possible points a moving object with speed limit $v_{\max}$ could have visited must lie within this ellipse with foci p and q. Moreover, the sum of their distances to p and q is less or equal to $v_{\max}(t_q - t_p)$. Any trajectory that touches the border of the ellipse and has more than two straight line segments, is longer than $v_{\max}(t_q - t_p)$. This particular trajectory lies in the ellipse and hence in the intersection of the unconstrained space-time prism and the road network, *but it does not lie in the road network space-time prism entirely* because there are points on it which can be reached in time, but from which the destination can not be reached in time and vice versa. Figure 2 depicts such a situation. There is no path on the road network from p that reaches q in the given time interval. The intersection of the space-time prism with the road network is nonempty, whereas the road network space-time prism clearly is. Figure 3 depicts a space-time prism on a road network, and its spatial projection.

To define space-time prisms on a road network, we need to define an appropriate distance function on the network. This distance measure is derived from the *shortest path*-distance used in graph theory [20]. Consider a road network RN, given by a set of vertices V and a set of labeled edges E. Let $p = (x_p, y_p)$ and $q = (x_q, y_q)$ be two points on RN, not necessarily vertices. Also suppose that p lies on (the embedding of) the edge $((x_{p,0}, y_{p,0}), (x_{p,1}, y_{p,1}))$ and q lies on the edge $((x_{q,0}, y_{q,0}), (x_{q,1}, y_{q,1}))$. We construct a new road network RN_{pq} from RN. Its set of vertices is $\mathsf{V_{pq}} = \mathsf{V} \cup \{p, q\}$ and its set of edges is $\mathsf{E_{pq}} = \mathsf{E} \cup \{((x_{p,0}, y_{p,0}), (x_p, y_p)), ((x_p, y_p), (x_{p,1}, y_{p,1})), ((x_{q,0}, y_{q,0}), (x_q, y_q)), ((x_q, y_q), (x_{q,1}, y_{q,1}))\}$. So, we have split the edges on which p and q are located.

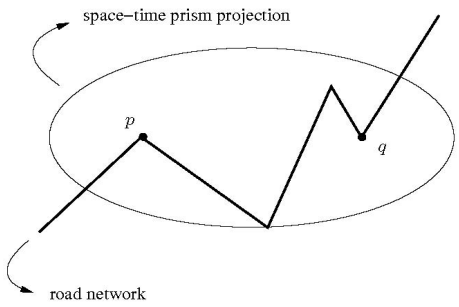

Fig. 2. Road network space-time prisms can not be easily derived from space-time prisms in $\mathbf{R}^2$.

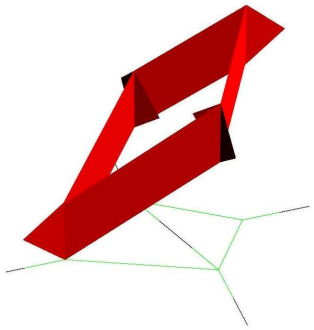

Fig. 3. Space-time prism (red) on road networks (green and black) and its spatial projection (green).

The speed limits are the ones of the original edges, and the time spans of the new edges are computed according to Definition 3. It is precisely this construction we need to define the distance along the road network RN and the space-time prism on RN. To do this we need a metric on RN.

Definition 4. [Road network time] Let RN be a road network and let $p, q \in \mathsf{RN}$. The *road network time* between p and q, denoted by $\mathsf{d}_{\mathsf{RN}}(p, q)$, is the shortest-path distance (i.e., the shortest-path distance as usual in graph theory and that can be computed by the Dijkstra's algorithm [20]) between p and q in the graph $(\mathsf{V}_{pq}, \mathsf{E}_{pq})$, with respect to the time-span labeling of the edges. □

Note that the *road network time* between p and q in the above definition has minimal total weight and returns the earliest possible time you can reach p from q and vice versa. The metric that we describe takes two points from a

road network and returns the shortest time needed to get from one to the other when traveling at the allowed maximal speed at each segment. We remark that if all edges in road network have the same speed limit $v_{\max}$, then the metric defined in Definition 4 is the shortest-path metric (up to a scaling factor $v_{\max}$) on the graph embedding RN. If, on the other hand, there are different speed limits per edge, then the metric of Definition 4 is the shortest time-span metric on the temporal projection of the spatio-temporal data. It follows that in the latter case, the shortest paths are not always the fastest paths. Figure 4 depicts a situation where neither one of the two shortest paths in the network is also the fastest path. Indeed, looking at the evolution of the prism over time, it is clear that the fastest path starts at the leftmost node, goes to the upper, then the lower node, and ends at the rightmost node.

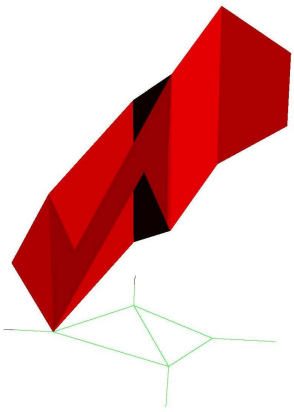

Fig. 4. Road Network space-time prism where the fastest path does not coincide with the shortest.

A space-time prism on a road network is the geometric location in $\mathbf{R} \times \mathsf{RN} \subset \mathbf{R} \times \mathbf{R}^2$ of all points a moving object could have visited when traveling, restricted to RN, from an origin p to a destination q with in a time-frame ranging from t_p to t_q, respecting the speed limits on the edges of RN.

4 An uncertainty-aware distance function

The goal of clustering algorithms is, basically, grouping together the objects which are similar to each other and keep them separated from the objects which are different. A key issue of this technique is the definition of a distance measure. To be useful for clustering, the distance function must be a metric, which means that it has to verify four well-known conditions stated in the next definition.

Definition 5. *A distance function $d()$ is a metric if:*

1. $\forall\ i:\ d(i,i) = 0$ *(identity)*
2. $\forall\ i,j : d(i,j) > 0$ *(positivity)*
3. $\forall\ i,j : d(i,j) = d(j,i)$ *(symmetry)*
4. $\forall\ i,j,k : d(i,j) + d(j,k) \geq d(i,k)$ *(triangularity).*

There are many (mainly Euclidean-based) distance functions used for clustering. In particular, for trajectory clustering, a definition of distance is aimed at considering similar two objects that followed approximately the same spatio-temporal trajectory. The main problem in this scenario is to find out which are the objects that moved together. However, depending on the application at hand, or the analysis a user needs to perform, other forms of distances could be useful. For instance, we may want to cluster together trajectories starting and ending at the same locations [18]. Nevertheless, as far as we are aware of, no distance function has been proposed to account for an intrinsic problem trajectory data have, that is, the uncertainty involved in GPS or GSM observations that originate the trajectory samples. The intuition behind this function is that the temporal projection of the intersection of the space-time prisms of two trajectories, represents the instants when the two trajectories *could have met.* Our claim is that the longer this period, the more similar the trajectories are. This notion is captured by the distance we introduce in Definition 6 below.

Definition 6 (Uncertainty-aware distance). *Let us denote A and B two necklaces corresponding to two trajectory samples τ_1 and τ_2, respectively. Let us also denote V_A and V_B the volumes defined by these necklaces. Then, the expression*

$$d_u(A,B) = 1 - \frac{V_A \cap V_B}{V_A \cup V_B} \tag{1}$$

is denoted the Uncertainty-based distance between τ_1 and τ_2.

Theorem 1 (Distance metric). *The distance of Definition 6 is a distance metric.*

Proof. Let us prove first the identity property. If we replace B by A in (1), (and, in consequence, V_B by V_A), we obtain:

$$\begin{aligned} d_u(A,B) &= 1 - \frac{V_A \cap V_B}{V_A \cup V_B} \\ &= 1 - \frac{V_A}{V_A} \\ &= 0 \end{aligned}$$

The second property is straightforward. Symmetry is proved in a way analogous to the simple proof for the first property:

$$
\begin{aligned}
d_u(B, A) &= 1 - \frac{V_B \cap V_A}{V_B \cup V_A} \\
&= d(A, B)
\end{aligned}
$$

Now, we prove the triangularity condition in Definition 5. For that, let us consider the diagram of Figure 5, where we have three sets, let us call them A, B, and C, representing three space-time prisms with these names. We replace the terms in Equation 2 with the elements in the partition of Figure 5.

$$
\begin{aligned}
d_u(A, B) &= 1 - \frac{a_2 + a_5}{a_1 + a_2 + a_3 + a_4 + a_5 + a_6} = \frac{a_1 + a_3 + a_4 + a_6}{a_1 + a_2 + a_3 + a_4 + a_5 + a_6} \\
d_u(B, C) &= \frac{a_2 + a_3 + a_4 + a_7}{a_2 + a_3 + a_4 + a_5 + a_6 + a_7} \\
d_u(A, C) &= \frac{a_1 + a_2 + a_6 + a_7}{a_1 + a_2 + a_4 + a_5 + a_6 + a_7}
\end{aligned}
$$

Then,

$$
d_u(A, B) + d_u(B, C) = \frac{a_1 + a_3 + a_4 + a_6}{a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7} + \frac{a_2 + a_3 + a_4 + a_7}{a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7}
$$

And, since $a_1, a_7 \geq 0$, we have:

$$
\begin{aligned}
d_u(A, B) + d_u(B, C) &\geq \frac{a_1 + a_3 + a_4 + a_6}{a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7} + \frac{a_2 + a_3 + a_4 + a_7}{a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7} \\
&= \frac{a_1 + a_3 + a_4 + a_6 + a_2 + a_3 + a_4 + a_7}{a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7} \\
&\geq \frac{a_1 + a_2 + a_6 + a_7 + a_3}{a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7}
\end{aligned} \tag{2}
$$

And, since "If $0 \leq a \leq b \neq 0$ and $c \geq 0$, then $\frac{a}{b} \leq \frac{a+c}{b+c}$", then we get:

$$
\geq \frac{a_1 + a_2 + a_6 + a_7}{a_1 + a_2 + a_4 + a_5 + a_6 + a_7} = d(A, C) \tag{3}
$$

5 An algorithm to compute d_u

We now describe the algorithm that computes the intersection between two chains of space-time prisms computed for two trajectories whose distance we need to calculate. In order to speed-up the computation, we pre-process information related to the road network. This pre-processed structures are presented next.

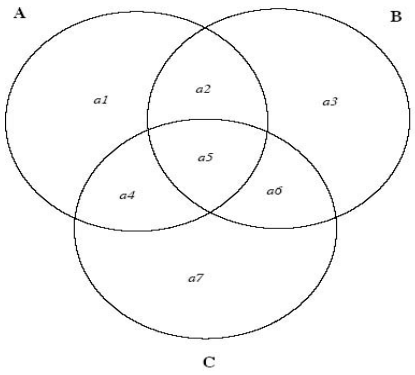

Fig. 5. A schematic description of the sets.

Data structure. The basic structure for storing the road network is a graph G, such that for each two nodes n_i, n_j, we store the weight of the edge connecting them, given by $\frac{d(n_i,n_j)}{v_{max}}$, where v_{max} is the speed limit in the road segment. We also have two lists, denoted `rowNonZeroes` and `columnNonZeroes`, defined as follows.

$\texttt{rowNonZeroes} = \{\ldots, \{.., \texttt{n}_\texttt{j}, ...\}_\texttt{i}, \ldots\}$

Thus, `rowNonZeroes` is a list of lists, such that the list in position i contains the nodes n_j such that $n_i, n_j \in G$. (i.e., the nodes reachable through just one edge from n_i). Analogously,

$\texttt{columnNonZeroes} = \{\ldots, \{.., \texttt{n}_\texttt{j}, ...\}_\texttt{i}, \ldots\}$

Thus, `columnNonZeroes` is a list of lists, such that the list in position i contains the nodes n_j such that $n_j, n_i \in G$. (i.e., the nodes that can reach n_i traversing just one edge).

Computing the distance between two chains of space-time prisms. First, we introduce an adaptation to our problem of the Dijstra's algorithm [20]. We want to find the shortest path to all nodes in the network, from a given node p, (a point in a trajectory, matched to a vertex in the network), and provided that the traveling time is less than a given threshold. For each p, we apply Algorithm 1.

Algorithm 1 *DijkstraSource*$(p, maxTime, nbrVertices)$

Output: *output*$_P$ = { *a list of nodes* $N_m = N \cup N_1$, *where* $N = (n_1, ...n_k)$ *is the set of nodes reachable from p in at most maxTime, and* $N_1 = \{n | n \notin N$ *and* $\exists\, n_i \in N, (n_i, n) \in G$, *and* $d(p, n) > maxTime\}$; *distlist* $= \{d_1, d_2, 0, d_n\}$, *a list containing the distances from p to the nodes in* N_m*; a list* O_e *of the form* $n, \langle e_1, ...e_f \rangle$ *representing the edges outgoing from the nodes in* N_m}

1: $ToProcess = \{(p, 0)\}$;

2: $distlist = \{\infty, \infty, 0, \infty, \}$, *a list of length —V—, with a zero in the position of the input node p.*
3: $Processed = \{\}$;
4: $N_m = \{\}$;
5: ***while*** $ToProcess.notEmpty()$ && $ToProcess[1].[2] < maxTime$ ***do***
6: $current := ToProcess[1]$;
7: *append* $current.[1].[1]$ *to* N_m*;*
8: *append* $(current[1].[1], \langle$`rowNonZeroes[current]`$\rangle$ *to* O_e*;*
9: *delete* $ToProcess[1]$;
10: ***for*** *each node* n_j *in* `rowNonZeroes[current]` ***do***
11: *build a pair* $(n_j, distlist[current.[1]] + d_{current.[1],j})$*;*
12: $distlist[j] := min(distlist[j], distlist[current.[1]] + d_{current.[1],j})$;
13: ***if*** $n_j \notin Processed$ ***then***
14: *append* (n_j, d_j) *to* $ToProcess$;
15: ***end if***
16: ***end for***
17: *sort ToProcess by time;*
18: *append* $current.[1]$ *to* $Processed$;
19: ***end while***

An analogous algorithm computes the shortest path from all nodes in the network, to a given node q, such that q is a point in a trajectory, matched to a vertex in the network, and provided that the traveling time is less than a given threshold. We call this algorithm `DijkstraDestination`. The main difference with Algorithm 1 is that $rowNonZeroes[current]$ is replaced by $columnNonZeroes[current]$, and that N_m contains the nodes that can reach q (i.e., instead of outgoing edges we work with incoming edges).

Algorithm 2 $\underline{DijkstraDestination(q, maxTime, nbrVertices)}$

Output: $output_Q = \{$ *a list of nodes* $N_m = N \cup N_1$, *where* $N = (n_1, ...n_k)$ *is the set of nodes that can reach* q *in at most* $maxTime$, *and* $N_1 = \{n | n \notin N$ *and* $\exists\, n_i \in N, (n, n_i) \in G$, *and* $d(n, q) > maxTime\}$; $distlist = \{d_1, d_2, 0, d_n\}$, *a list containing the distances to* q *from the nodes in* N_m*; a list* I_e *of the form* $n, \langle e_1, ...e_f \rangle$ *representing the edges incoming to the nodes in* $N_m\}$

1: $ToProcess = \{(q, 0)\}$*;*
2: $distlist = \{\infty, \infty, 0, \infty, \}$, *a list of length —V—, with a zero in the position of the input node q;*
3: $Processed = \{\}$;
4: $N_m = \{\}$;
5: ***while*** $ToProcess.notEmpty()$ && $ToProcess[1].[2] < maxTime$ ***do***
6: $current := ToProcess[1]$;
7: *append* $current.[1].[1]$ *to* N_m*;*
8: *append* $(current[1].[1], \langle$`columnNonZeroes[current]`$\rangle$ *to* O_e*;*
9: *delete* $ToProcess[1]$;
10: ***for*** *each node* n_j *in* `columnNonZeroes[current]` ***do***
11: *build a pair* $(n_j, distlist[current.[1]] + d_{current.[1],j})$*;*

12: $distlist[j] := min(distlist[j], distlist[current.[1]] + d_{current.[1],j})$;
13: ***if*** $n_j \notin Processed$ ***then***
14: $append\ (n_j, d_j)\ to\ ToProcess$;
15: ***end if***
16: ***end for***
17: *sort ToProcess by time;*
18: *append current.*[1] *to Processed*;
19: ***end while***

Now, given two points p and q in a trajectory, we compute their space-time prisms $prism_1$ and $prism_2$, using the output of Algorithms 2 and 3. From them, we compute their union and intersection, from which the distance follows straightforwardly.

Algorithm 3 $\underline{prism(p_1, t_{p_1}, q_1, t_{q_1}, p_2, t_{p_2}, q_2, t_{q_2})}$

Output: *The polygons and their surface, in the two prisms and their intersection (for the prisms defined by the two pairs of points in the input).*

1: $DijkstraSource(p_1, maxTime, nbrVertices)$;
2: $DijkstraDestination(q_1, maxTime, nbrVertices)$;
3: $DijkstraSource(p_2, maxTime, nbrVertices)$;
4: $DijkstraDestination(q_2, maxTime, nbrVertices)$;
5: ***if*** $p_1 \in output_{Q_1}[1]$ *and* $q_1 \in output_{P_1}[1]$ ***then***
6: $prism_1 = \{N_{m_{p_1}} \cap N_{m_{q_1}}, distlist_{p_1}, distlist_{q_1}, O_{e_{p_1}} \cap I_{e_{q_1}}\}$;
7: ***end if***
8: ***if*** $p_1 \in output_{Q_2}[1]$ *and* $q_1 \in output_{P_2}[1]$ ***then***
9: $prism_2 = \{\{N_{m_{p_2}} \cap N_{m_{q_2}}, distlist_{p_2}, distlist_{q_2}, O_{e_{p_2}} \cap I_{e_{q_2}}\}$;
10: ***end if***
11: ***if*** $prism[1] \cap prism_2[1] == \emptyset$ ***then***
12: *Compute only the surface of the polygons;*
13: ***else***
14: *compute polys* $= \{distlist_{p_1}, distlist_{q_1}, distlist_{p_2}, distlist_{q_2}, prism_1[1] \cup prism_2[1], prism_1[4] \cup prism_2[4]\}$;
15: ***for*** *every edge in polys* ***do***
16: *Compute the polygons of* $prism_1$, $prism_2$, *and* $prism_1 \cap prism_2$, *using the algorithm in [11];*
17: ***end for***
18: ***end if***

The final step of the algorithm computes the intersection and union of two chains of prisms. We explain this through an example, for the sake of clarity. Let us consider two trajectories:
$T_1 = \{(p_1, t_1)(p_2, t_2)(p_3, t_3)...(p_n, t_n)\}$
$T_2 = \{(p'_1, t'_1)(p'_2, t'_2)(p'_3, t'_3)...(p'_m, t'_m)\}$
We now compute the interval where the trajectories overlap, and compute the surface with Algorithm 3. For example, if the time instants are such that

$t_3 < t'_1 < t_4 < t'_2 < t'_3 < t_5$...., we first use the points $(p_3, t_3), (p_4, t_4)$ and $(p'_1, t'_1), (p'_2, t'_2)$, and compute the intersection and union of the two prisms.

The algorithm is based on a sliding window approach. Thus, we next compute the intersection of the prisms corresponding to $(p_4, t_4), (p_5, t_5)$ and $(p'_1, t'_1), (p'_2, t'_2)$ (note that there is a non-empty intersection here too); and, analogously, (p_4, t_4), (p_5, t_5) and (p'_2, t'_2), (p'_3, t'_3).

A first optimization in the computation of the distance function is that if the intersection is empty, we do not compute the surface of the polygons in the extremes (for computing the union). A second optimization we implemented in our algorithm (besides the pre-computation step already mentioned), is that we keep the results of the previous iteration when computing the intersection of the chains of prisms (for example, we keep the prism for $(p'_1, t'_1), (p'_2, t'_2)$ above).

Intersection of space-time prisms The distance defined in Section 4 requires the computation of the intersection between two space-time prisms. For this computation, in short, the algorithm described in [11] iterates over every edge that has at least one polygon in both space-time prisms, and compute the intersection as an intersection between polygons. However, since we use this intersection to compute the distance between two chains of space-time prisms, a subtle problem appears here. A prism on a road network is basically composed by the three geometries depicted in Figure 6 (a) through (c). This yields a geometry like the one in Figure 6 (d). A naive computation of the intersection would imply intersecting each component of one prism, with all the components of the other one. It is clear that this may result in an intersection larger than the union. Thus, we only intersect each component in one prism, with its analogous in the other prism. In this way, if we have two prisms B_1 and B_2, such that $B_1 = B_2$, then $B_1 \cap B_2 = B_1 \cup B_2$. Note that, in addition of allowing to use our distance function, this way of computing the intersection provides an appropriate and natural semantics to the space-time prisms. In two-way roads, and given the points r and s of Figure 6, the triangle corresponding to r represents the trajectories going from r to r. Analogously, the triangle corresponding to s represents the trajectories going from s to s. The parallelogram (Figure 6 (b)) represents the trajectories going from r to s. In one-way roads (like in the case we are studying), only the latter is considered.

6 Preliminary experimental results

We performed (very) preliminary experiments using a set of 52 trajectories obtained from cars moving in the city center of Milan, in one day, from 1PM to 2PM, that is, all in the same short period, increasing the possibility of moving together. We used k-mediods with k=2..6 as clustering algorithm. We only consider trajectories with at least 5 different points. Also, we split a trajectory if there is a gap of more then 6 minutes between two consecutive points.

Results for k=6 are shown in Figure1 7. The blue cluster represents all trajectories that, according to principle of alibi query [11], could not have met. Thus,

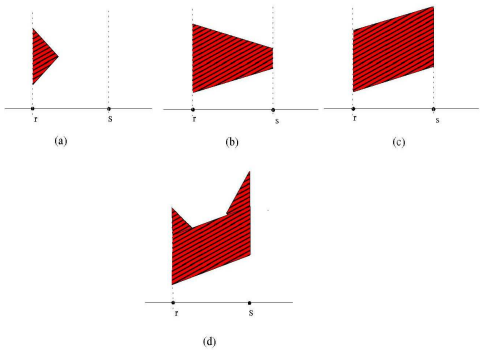

Fig. 6. Space-time prism components.

the distance between any two trajectories in the blue cluster is 1. The cluster composition is shown in the next table.

Cluster	Color	# of trajectories
0	blue	26
1	red	13
2	green	4
3	black	2
4	cyan	1
5	orange	6

Figure 8 shows the clusters for k=4, overlayed with the road network of Milan, to provide a more realistic way of displaying the results. We see that the blue clusters are similar to the ones obtained using k=6. More precisely, for all clustering runs (with k=2..6), the blue cluster contained 26 trajectories, i.e., the trajectories that could have never met each other are always together in one cluster, no matter the parameter k used.

Prism-based clustering allows to draw some additional conclusions, useful for traffic analysis. Note that the closer to the speed limits the objects move (i.e., fluent traffic), the less likely their trajectories are to get clustered together. In our case, our results suggest that cars in the sample move at speeds higher than the speed limit. This follows from observing that, initially, using the city's maximum speed limit in downtown, many empty prisms are obtained. Adjusting these limits, increasing the speed by 20%, the number of empty prisms dropped considerably, but prisms were narrow, mainly yielding empty intersections. However, we remark that this assertion always needs further insight into the dataset, since also the results are influenced by the map matching process. In our case, we use a *geometric* approach, which maps a point to the closest street segment, which sometimes could yield imprecise results.

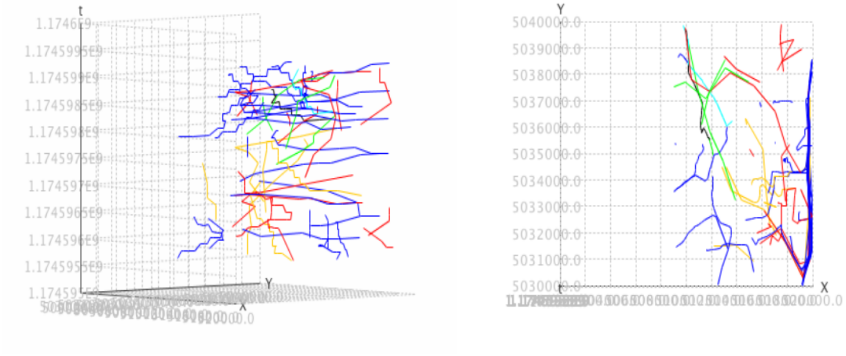

Fig. 7. Result of clustering with k=6 (left); Projection over x-y coordinates (right).

7 Conclusion and future work

We have presented a new distance function for clustering trajectory samples, that accounts for the inherent uncertainty contained in the GPS observations. For this, instead of computing the distance between the trajectories themselves (e.g., using some metric, like the Euclidean distance), we measure the relation between the union and the intersection of the space-time prisms associated to each trajectory (using the speed limit of the road segment), in a way such that if the intersection is empty, then, the two trajectories could not have met, and the distance equals '1'. To the best of our knowledge, this is the first proposal in this sense. We also provided a formal proved that this distance is actually a metric, and sketched the algorithm to compute it. Finally, we presented preliminary experimental results using real data obtained from cars moving in the Milan.

References

1. Agrawal, R., Lin, K., Sawhney, H.S., K., S.: Fast similarity search in the presence of noise, scaling, and translation in time-series databases. In: VLDB. (1995) 490–501
2. Ankerst, M., Breunig, M., Kriegel, H.P., Sander, J.: Optics: Ordering points to identify the clustering structure. In: SIGMOD Conference. (1999) 49–60
3. Egenhofer, M.: Approximation of geopatial lifelines. In: SpadaGIS, Workshop on Spatial Data and Geographic Information Systsems. (2003)
4. Ester, M., Kriegel, H.P., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: KDD. (1996) 226–231
5. Faloutsos, C., Lin, K.L.: Fastmap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In: SIGMOD Conference. (1995) 163–174
6. Gaffney, S., Smyth, P.: Trajectory clustering with mixtures of regression models. In: KDD. (1999) 63–72
7. Güting, R.H., Schneider, M.: Moving Objects Databases. Morgan Kaufmann (2005)

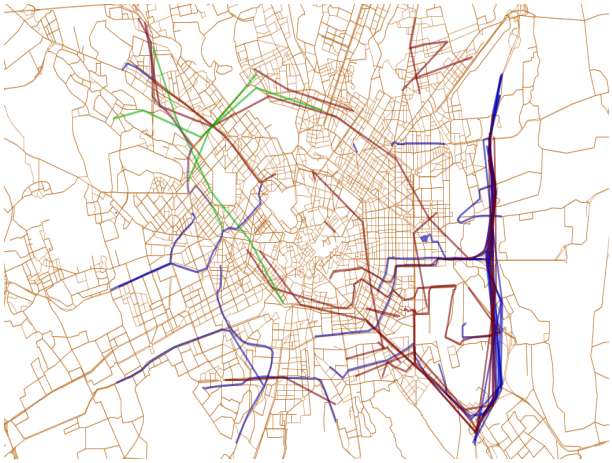

Fig. 8. Result of clustering Milan data with k=4 overlayed with the road network.

8. Hägerstrand, T.: What about people in regional science? Papers of the Regional Science Association **24** (1970) 7–21
9. Han, J., Kamber, M.: Data Mining: Concepts and Techniques. Morgan Kaufmann (2000)
10. Ketterlin, A.: Clustering sequences of complex objects. In: KDD. (1997) 215–218
11. Kuijpers, B., Othman, W.: Modelling uncertainty of moving objects on road networks via space-time prisms. International Journal of Geographical Information Science. To appear in 2009
12. Miller, H.: Modeling accessibility using space-time prism concepts within geographical information systems. International Journal of Geographical Information Systems **5** (1991) 287–301
13. Miller, H., Wu, Y.: Gis software for measuring space-time accessibility in transportation planning and analysis. GeoInformatica **4** (2000) 141–159
14. Nanni, M.: Clustering Methods for Spatio-Temporal Data. PhD thesis, Computer Science Departrment, University of Pisa (2002)
15. Nanni, M., Pedreschi, D.: Time-focused clustering of trajectories of moving objects. Journal of Intelligent Information Systems **27**(3) (2006) 267–289
16. Nanni, M., Kuijpers, B., Körner, C., May, M., Pedreschi, D.: Spatiotemporal data mining. In: Mobility, Data Mining and Privacy. (2008) 267–296
17. Pfoser, D., Jensen, C.S.: Capturing the uncertainty of moving-object representations. In: Advances in Spatial Databases (SSD'99). Lecture Notes in Computer Science (1999) 111–132
18. Rinzivillo, S., Pedreschi, D., Nanni, M., Giannotti, F., Andrienko, N., Andrienko, G.: Visually driven analysis of movement data by progressive clustering. Information Visualization **7** (2008) 225–239
19. Vlachos, M., G., D., George Kollios, G.: Discovering similar multidimensional trajectories. In: ICDE. (2002) 673–684
20. Weisstein, E.: Wolfram mathworld. (2007) http://mathworld.wolfram.com/DijkstrasAlgorithm.html.
21. Wolfson, O.: Moving objects information management: The database challenge. In: NGITS. (2002) 75–89