TRACK TRADE

Creating a Data Mart for Floating Car Data

COOP-CT-2006-032823
Co-operative Research Projects, FP6

Deliverable 2.1
Data Management I

Deliverable lead contractor: CTI

Sotiris Brakatsoulas, CTI mprakats@cti.gr
Dieter Pfoser, CTI pfoser@cti.gr
Carola Wenk, CTI

Due data: 31-08-2007
Actual submission date: 09-10-2007

Abstract

Sampling vehicular movement using GPS is affected by error sources. Given the resulting inaccuracy, the vehicle tracking data can only be related to the underlying road network by using map-matching algorithms. This deliverable presents the map-matching algorithms developed during this project.

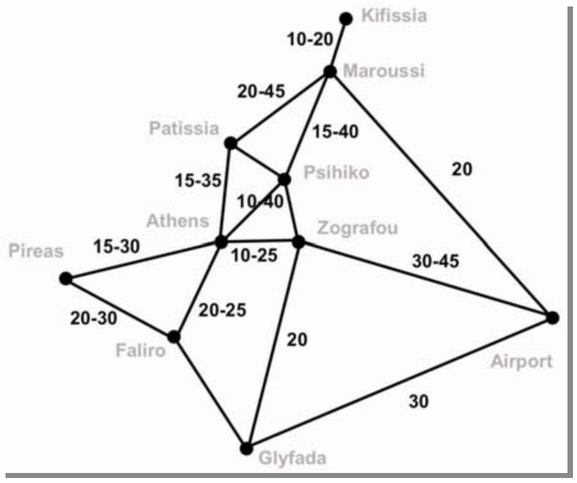
Table of Contents

1. Introduction

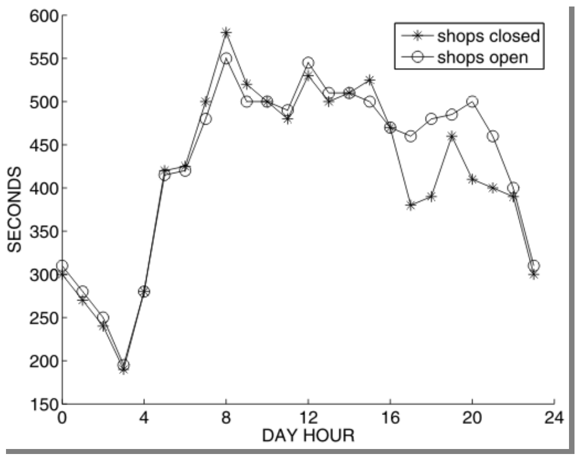
With the availability of cheap positioning technology and the penetration of asset tracking applications such as fleet management applications, vehicle tracking data, as a component of floating car data (FCD), becomes an important tool for traffic assessment and prediction. The whole motivation of the TRACK&TRADE project is derived from this fact.

Map-matching the tracking data produces travel time data related to a specific edge in the road network. Collection of large amounts of tracking data and, thus, travel times produce travel time profiles for a road network (cf. Figure 1).

Daily courses of speed profiles and current travel times derived from historic and current FCD, respectively provide dynamic link-based speed types, i.e., dynamic impedance in a road network. This is in contrast to static weights base on speed limits and road categories, which are currently available from map producers.



Travel time fluctuations in a road network



Daily course of speed

Figure 1: Varying travel times

With the tracking data typically being a byproduct of another application, e.g., fleet management, this has serious implications for the data quality, e.g., infrequent position samples. Still being able to utilize such data for traffic assessment affords sophisticated map-matching algorithms to accomplish the task of matching this inaccurate tracking data to a road network.

Of critical importance for traffic assessment is the amount of available data and its timeliness. To provide real-time map-matching, fast map-matching algorithms are needed. This deliverable describes the map-matching algorithm developed in TRACK&TRADE.

Figure 2 relates this deliverable to the rest of core technical deliverables. D1.1 provided for the data collection technology. D2.2 will provide for the glue that links data collection, map-matching and travel time aggregation and computation together. A significant portion of the work on D2.2 has been completed since it has been essential for the testing of the map-matching algorithms.

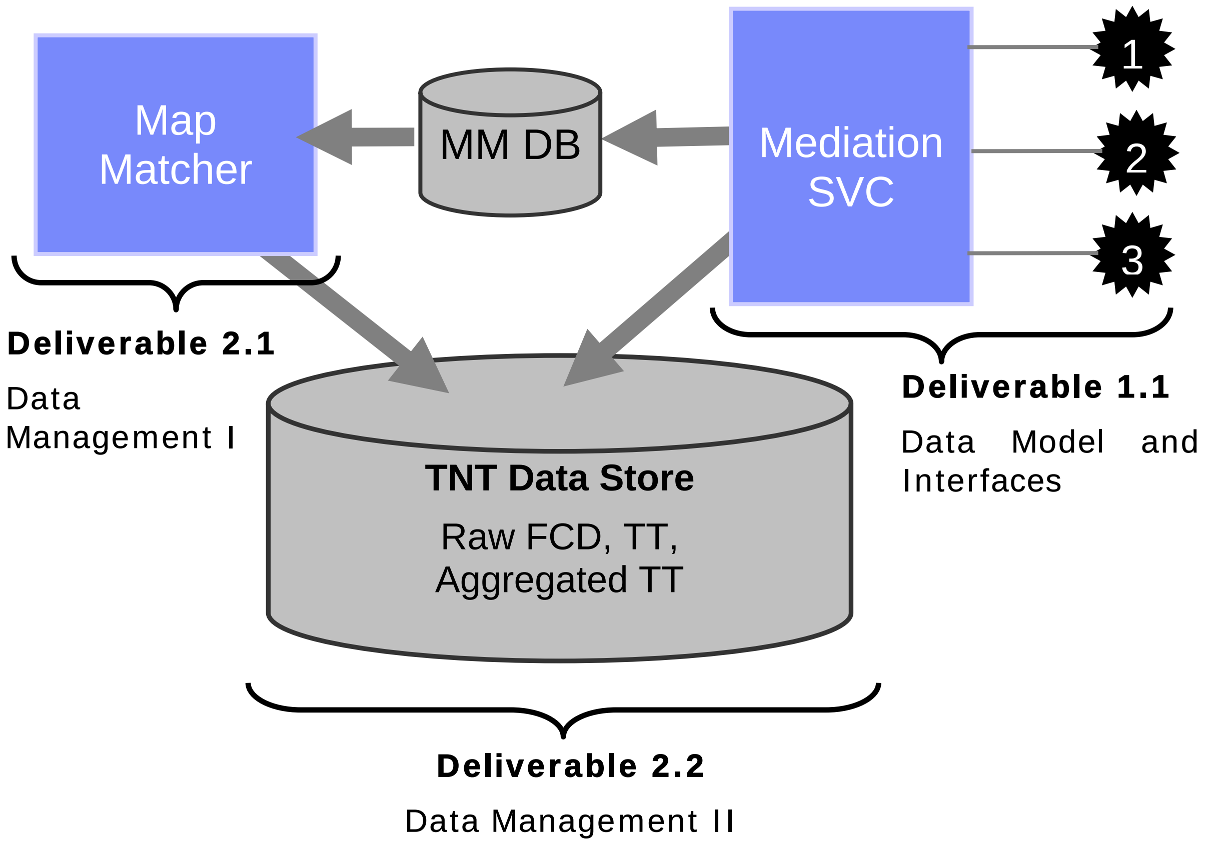


Figure 2: Core technical deliverables overview and integration

The outline of this deliverable is as follows. Section 2 will introduce the nature of the data we are dealing with and showcases the problem when tracking data needs to be matched to a road network. Section 3 introduces the inner workings behind our map-matching algorithm. Section 4 shows the evaluation of the algorithm. Section 5 describes the software framework that has been developed, essentially wrapping the map-matching algorithm in a set of software components and respective interfaces.

2. On Tracking Data

The objective of this work is to develop map-matching algorithms for vehicle tracking data that are used as a sensor data source to assess and predict the traffic condition in related applications. This section overall describes the properties of the tracking data with a focus on its accuracy to define requirements for map-matching algorithms.

2.1 Imprecise Tracking Data

The tracking data can be modeled in terms of a trajectory, which is obtained by interpolating the position samples. Typically, linear interpolation is used as opposed to other methods such as polynomial splines. The sampled positions then become the endpoints of line segments of polylines, and the movement of an object is represented by an entire polyline in 3D space [6].

The positioning technology of choice for vehicle tracking is typically GPS. Its associated *measurement error* in connection with the *sampling rate* require us to match position samples to the road network.

2.1.1 Measurement Error

Two assumptions are generally made about the accuracy of GPS. First, the error distribution, i.e., the error in each of the three dimensions and the error in time, is assumed to be *Gaussian*. Second, we can assume that the horizontal error distribution, i.e., the distribution in the x-y plane, is circular.

The error in a positional GPS measurement can be described by the probability function following a bivariate normal distribution [4]. The probability function is composed of two normal distributions in the two respective spatial dimensions. The mean of the distribution is the origin of the coordinate system.

Although the GPS error can be substantial in certain situations (shadowed and reflected signals), with the use of augmenting the GPS signal (WAAS, EGNOS) the typical error is in the range of 8m to 2m.

The following section discusses the sampling error, which in the case of vehicle tracking data is by two magnitudes larger than the measurement error and consequently has a larger impact on the algorithmic design of a map-matching algorithm.

2.1.2 Sampling Error

The uncertainty of the representation of an object’s movement is affected by the frequency with which position samples are taken, the *sampling rate*. To illustrate the impact of the sampling rate on the imprecision of the interpolated trajectory data, consider sampling the position of a vehicle every 30*s*, which is a typical sampling rate used in vehicle tracking applications. At a speed of 50*km/h*, the traveled distance between position samples is as much as 417*m*!

Not measuring the positions in-between two consecutive position samples, the best we can do is to *limit the possibilities of where the moving object could have been*. The trajectory can be constrained by what we know about the object’s actual movement.

The sampling error between two positions, P_1 and P_2 in the time interval **Error!** and a given *maximum speed*, v_m, for a time t_x with $t_1<t_x<t_2$ is bound by a lens-shaped probability distribution. Computing the sampling error for various instances of t_x shows that a possible trajectory between P_1 and P_2 is overall bound by an error ellipse (cf. Figure 3). The foci of the ellipse are the sampled positions P_1 and P_2 and the eccentricity 2c is the Euclidean distance between P_1 and P_2. The length of the semi-major axis, 2a, is the maximum distance the object can travel. If the sampling rate r is given in seconds and the velocity is v*km/h* then $2a=5/18 \cdot vr$. The “thickness” of the ellipse, 2b, is determined by the equation $b^2=a^2-c^2=25/36^2 \cdot v^2r^2-c^2$. In simple words, the faster the object travels and the closer its path to that of a linear trajectory, the “thinner” the ellipse. In extreme cases, the ellipse degrades to a *line*, or even to a *point* in case the object stopped.

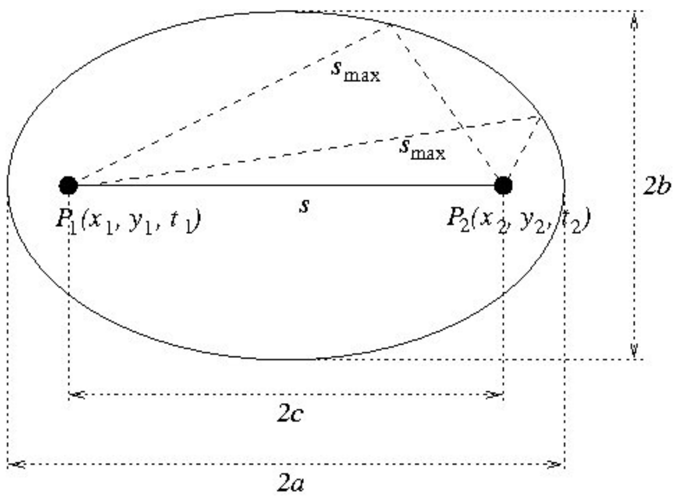


Figure 3: Sampling error ellipse

Considering again the initial example, Figure 4 gives its sampling error scenario. On a road network (gray lines), a vehicle travels along the dotted road network edge from P_1 to P_2 at a typical speed of $50km/h$. Since it has to stop at the intersection, its average speed is $25km/h$. Between P_1 and P_2, the travelled distance along the road network is $208m$ (length of dotted path) and assuming P_1 is $140m$ and P_2 $68m$ from the intersection, the Euclidean distance is $156m$. The resulting error ellipse has a major axis of $2a=5/18\cdot 50km/h\cdot 30s=417m$ and an eccentricity of $2c=156m$. The thickness of the ellipse is $2b=387m$. Translated to a map-matching task, one would have to consider all the network edges contained in this error ellipse. Additional knowledge such as the number of intersections on a path reduce the possibilities, but several possible alternatives for mapping the trajectory onto the road network remain.

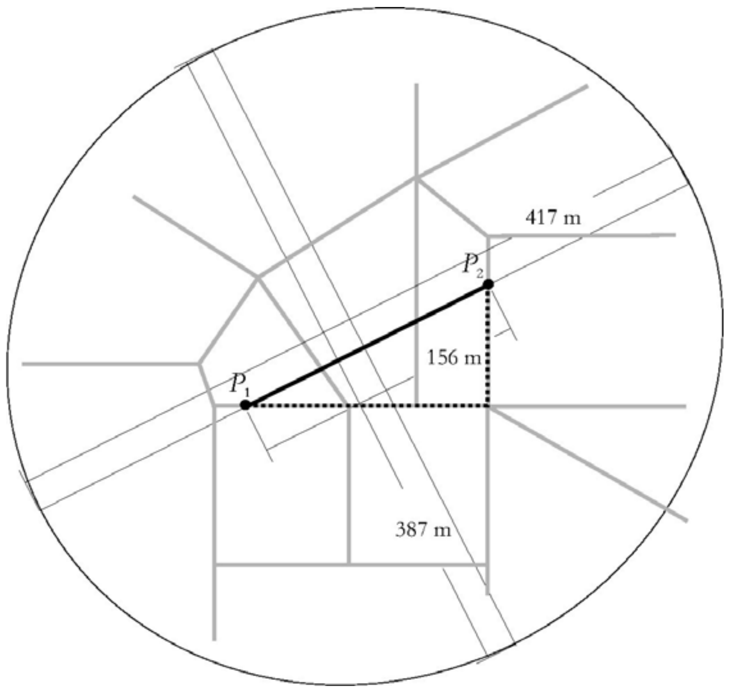


Figure 4: Sampling error ellipse example

Figure 5 gives an example of a vehicle trajectory composed of GPS measurements (asterixes connected by line segments). Typically the GPS measurements do not lie on the road network (measurement error) and neither can the connecting line segments easily be matched to edges of the road network (sampling error).

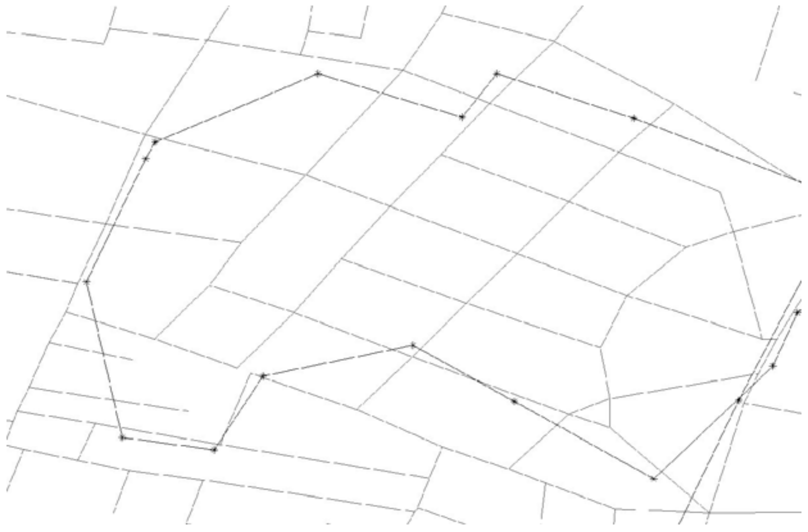


Figure 5: Measurement and sampling error example

3. Map-Matching Algorithm

Algorithms for map-matching vehicle-tracking data comprise the categories of *incremental algorithms* and *global algorithms*.

Incremental algorithms follow a greedy strategy of sequentially extending a solution from an already matched edge. Essentially, such algorithms try to match one GPS point after the other to the closest and most similar road network edge. Incremental algorithms are usually very fast, however due to their heuristic nature they restrict the trajectories in the road network that are considered as a valid match. A incremental map-matching algorithm is described in [3]. We will use this algorithm in Section 4 as a benchmark to assess the performance of the global algorithm that we utilize in TRACK&TRADE.

Global algorithms consider all possible trajectories in the road network to find amongst them the trajectory that is most similar to the vehicle trajectory. Alt et al. [1] and Brakatsoulas et al. [3] utilize the strong and the weak Fréchet distance measures to find a path in the road network that closely resembles the trajectory. Although at first sight considering all possible paths seems inefficient, the existing algorithms run in polynomial time. Moreover, such algorithms are based on a notion of similarity between two curves, which has the advantage that the computed result trajectory comes with a quality guarantee, since it is a curve which minimizes the distance to the vehicle trajectory among all possible curves in the road network.

In terms of quality and speed, global algorithms generally produce high-quality matching results but are slow compared to incremental algorithm.

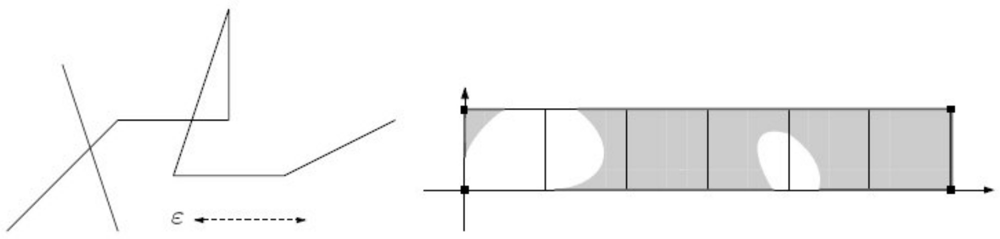
3.1 Fréchet Distances

The algorithm we utilize in TRACK&TRADE is based on the global map-matching algorithms of [1] and [3], which employ the Fréchet distance measure for curves.

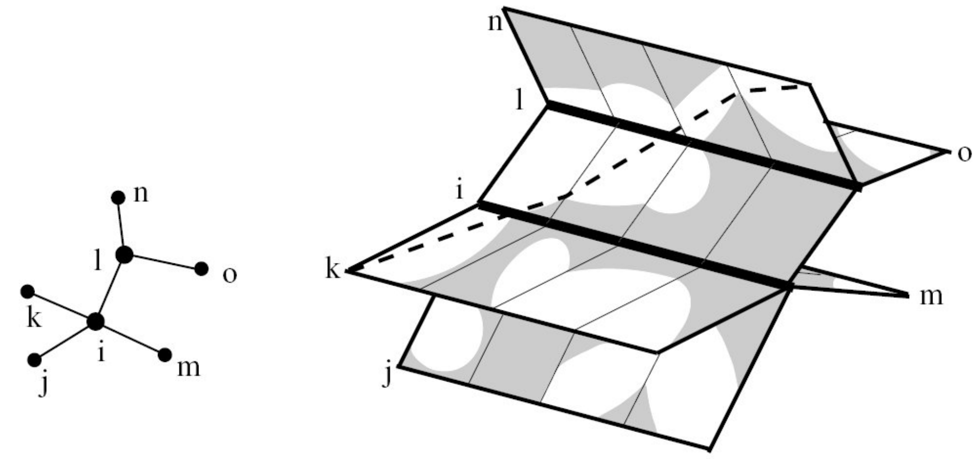
This section will give relevant definitions and results that will be needed in the remainder of this work. For more details we refer the reader to [1],[2] and [3].

The (strong) Fréchet distance for two curves has been proposed by Fréchet. A popular illustration of the Fréchet distance is the following: Suppose a person is walking his dog, the person is walking on the one curve and the dog on the other. Both are allowed to control their speed but they are not allowed to go backwards. Then the *(strong) Fréchet distance* of the curves is the minimal length of a leash that is necessary for both to walk the curves from beginning to end. If both are allowed to go backwards then one obtains the *weak Fréchet distance*

Most algorithms that compute the (weak or strong) Fréchet distance between two curves or which employ those distances in global map-matching algorithms first solve their decision variant: For a fixed $\varepsilon>0$ decide whether the distance is at most ε or not. Afterwards the minimization problem is solved by applying parametric search or binary search. The algorithms solving these decision problems for a fixed $\varepsilon>0$ are all based on the notions of the *free space diagram* or the *free space surface* [1].



(a) Free space



(b) Free space surface

Figure 6: Free spaces: (a) A free space diagram. (b) A road network (left) and corresponding free space surface (right).

The free space diagram of a line segment (edge in the road network graph) and a curve (trajectory) encodes which pair of points, one on the line segment and the other on the curve, are at distance at most ε. The axes of a coordinate system are identified with the parameterizations of the curve and the line segment. A white point in the free space diagram encodes a pair of points at distance at most ε, and a black point a distance greater than ε. See Figure 6(a) for an example.

Both the free space diagram of two polygonal curves as well as the *free space surface* of the road network and the trajectory are composed of these segment-curve free space diagrams; for every two incident line segments (in the road network, or in another curve) their individual free space diagrams with the curve are glued together according to the incidence information. Figure 3(b) gives an example road network (left) and its corresponding free space surface for a vehicle trajectory consisting of five position samples (right). The vehicle trajectory is not shown explicitly but implicitly by the white free space area. An example path in the free space from lower left to upper right is drawn dashed.

Deciding whether the weak or strong Fréchet distances are at most ε amounts to finding a path in the white free space area from a lower left corner to an upper right corner. For the strong Fréchet distance the path has to be monotone, for the weak Fréchet distance it can be any path.

3.2 Using the Fréchet Distance

The decision problem for the Fréchet distance between two curves can be solved by computing a monotone curve from the lower left corner to the upper right corner in the free space. This can be done using a dynamic programming approach in $O(mn)$ time [2].

Alt et al. [1] generalized this approach to finding a monotone path in the free space surface from a lower left corner of some individual edge-trajectory free space diagram to an upper right corner of some other individual edge-trajectory free space diagram. The algorithm conceptually sweeps a line from left to right (in direction of the trajectory) over all free spaces at the same time while maintaining the points on the sweepline that are reachable by some monotone path in the free space from some lower left corner. It then updates this reachability information Dijkstra-style while advancing the sweepline. Interestingly, the algorithm runs in $O(mn\log mn)$ time, which is only a log-factor slower than the algorithm of [2], although it accomplishes the seemingly more complicated task of comparing the trajectory to all possible curves in the road network.

3.2.1 Using the Weak Fréchet Distance

The decision problem for the weak Fréchet distance between two curves can be solved by testing if there exists any path in the free space of the two curves from the lower left corner to the upper right corner. This can be done using any graph traversal algorithm such as depth-first search in $O(mn)$ time.

We generalize this approach to the global map-matching problem by applying depth first search to the free space surface. We initialize the search with all white lower left corners of individual edge-trajectory free spaces, and stop the search if we found some upper right white corner. Since the free space surface consists of mn edge-segment cells, this algorithm runs in $O(mn)$ time, which is a log-factor faster than the algorithm based on the Fréchet distance. Applying parametric search for optimization, in the same way as in [1],[2] adds an additional log-factor to the runtime for a total of $O(mn\log mn)$ to solve the optimization problem.

3.3 Algorithmic Optimizations

Additional metadata information related to the tracking data, such as tracking data error, can be exploited to refine the modeling of the map-matching task. In this section, we introduce the *error-aware map-matching task* and the Adaptive

Clipping algorithm. The algorithm is a way to localize the global weak Fréchet map-matching algorithm. At the same time it is the first map-matching algorithm which provably solves this well-defined map-matching task.

3.3.1 Error-Aware Map-Matching

The tracking data is obtained by sampling positions, typically using GPS, to produce data that in database terms is commonly referred to as trajectories. Unfortunately this data is not precise due to the *measurement error* caused by the limited GPS accuracy, and the *sampling error* caused by the sampling rate (cf. Section 2.1). Figure 7 visualizes a worst-case estimate superimposing measurement and sampling error.

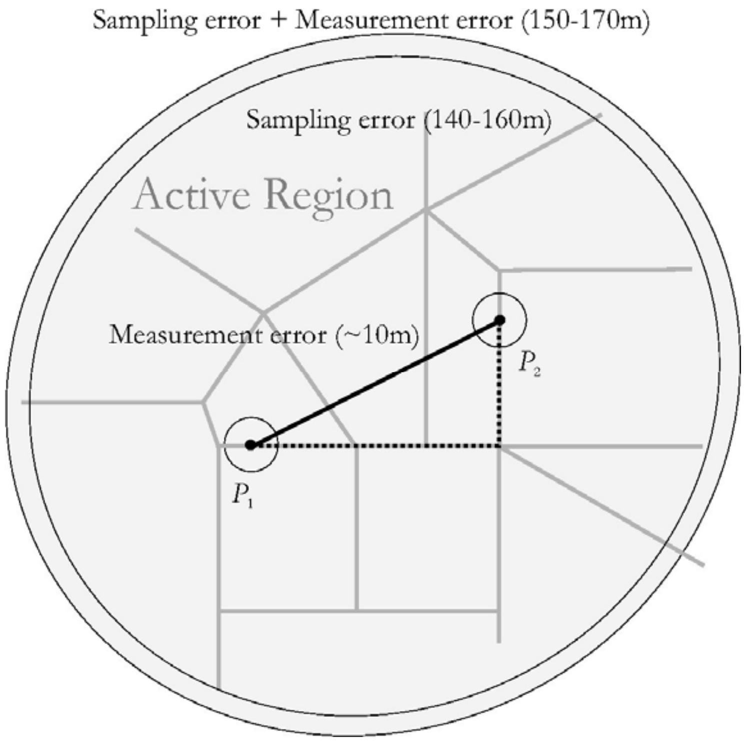

Sampling error + Measurement error (150-170m)
Sampling error (140-160m)
Active Region
Measurement error (~10m)
P_2
P_1

Figure 7: Measurement error, sampling error, and active region.

Usually the vehicle trajectory is modeled as a polygonal curve consisting of a sequence of vertices that are connected by straight-line edges, with the vertices $p_{1,\ldots,}p_n$ being the sampled GPS positions. Given the two error measures, the positions of the vertices are only accurate up to the measurement error, and the straight-line edges are only accurate up to the sampling error. A more appropriate error-aware representation of the vehicle trajectory is to represent every GPS position with a disk that reflects the measurement error and every two consecutive GPS positions with an error-ellipse that reflects the sampling error (cf. Figure 7). This is a “fuzzy” representation of the trajectory, which describes a region in the plane that contains all possible trajectories where the vehicle could have been, and therefore conveys the true content of the data, as opposed to the polygonal-curve representation, which conveys a false sense of preciseness.

For any two consecutive positions p_i, p_{i+1} let us call $\overline{p_i p_{i+1}}$ an *edge* of the vehicle trajectory. Let the velocity *v*, the sampling rate *r*, and the measurement error μ be given. For every p_i we define its *active region* $A(p_i)$ as the disk centered at p_i

with radius μ. For every edge $\overline{p_i p_{i+1}}$ we define its *active region* $A\left(\overline{p_i p_{i+1}}\right)$ as the Minkowski sum of the error ellipse (defined by *e*, *v*, and *r*) with the disk of radius μ (centered at the origin).[1] Intuitively $A\left(\overline{p_i p_{i+1}}\right)$ is a “thickened” ellipse whose boundary has been thickened by a disk of radius μ (gray-shaded area in Figure 7). $A(p_i)$ contains all positions in the plane which by the measurement error could correspond to p_i, and $A\left(\overline{p_i p_{i+1}}\right)$ is a combination of measurement and sampling errors and contains all positions in the plane which could correspond to any point in between p_i and p_{i+1}.

The sequence of all active regions

$$A(p_1), A\left(\overline{p_1 p_2}\right), A(p_1), \ldots, A(p_{n-1}), A\left(\overline{p_{n-1} p_n}\right), A(p_n)$$

is an *error-aware* representation of the vehicle trajectory. See Figure 8 for an example of active regions. Notice that metadata such as known speed on trajectory edges could be used to refine the active regions.

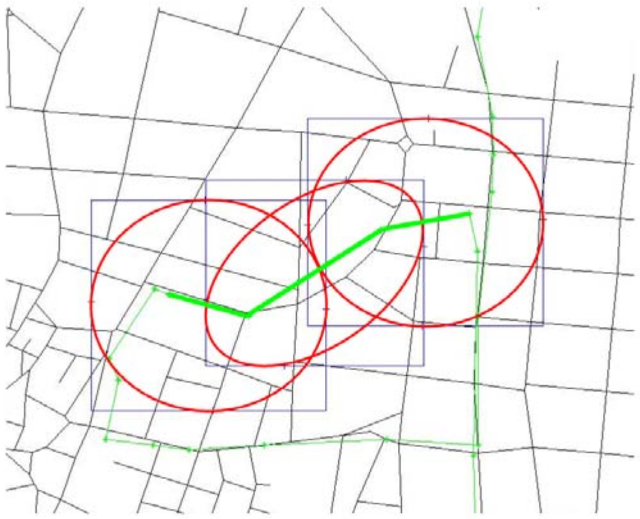


Figure 8: An excerpt of the road network, the trajectory, and three active regions with their bounding boxes.

The following Lemma shows which properties those trajectories in the road network have that could have led to the observed vehicle trajectory.

Lemma 1: *The true trajectory of vertices* $q_1, \ldots, q_l$ *in the road network that has lead to the vehicle trajectory* $p_1, \ldots, p_n$ *has the following properties:*

(i) ***Start condition:*** $q_1 \in A(p_1)$

(ii) ***End condition:*** $q_l \in A(p_n)$

[1]The Minkowski sum (also called vector sum) of two point sets **Error!** is defined as $C \oplus D = \{c + d \mid c \in C, d \in D\}$.

(iii) ***Intersection property:*** *The polygonal curve* $q_1,\ldots,q_l$ *intersects* $A(p_i)$ *for every* $1 \le i \le n$.

(iv)***Containment property:*** *If* $q_i \in A(p_j)$ *and* $q_{i+k} \in A(p_{j+1})$ *for* $k \ge 0$ *then the polygonal curve* $q_i,\ldots,q_{i+k}$ *has to lie in* $A(\overline{p_j p_{j+1}})$.

The respective proof can be found in [7]. We define the error-aware map-matching task as follows:

- **Given**: A road network G and an error-aware representation $A(p_1), A(\overline{p_1p_2}), A(p_1),\ldots, A(p_{n-1}), A(\overline{p_{n-1}p_n}), A(p_n)$ of the vehicle trajectory.
- **Task**: Find a curve $q_1,\ldots,q_l$ in G that fulfills the properties of Lemma 1.

This formulation of the map-matching task models all known information about the map-matching problem. In the following Section we describe the *Adaptive Clipping algorithm* which solves the error-aware map-matching task.

3.3.2 Adaptive Clipping

The *Adaptive Clipping* algorithm is a way to localize the global weak Fréchet map-matching algorithm. In Theorem 2 we show that in addition it provably solves the error-aware map-matching task as defined in the previous Section.

The *Adaptive Clipping* algorithm follows an incremental clipping approach: Our output-sensitive weak Fréchet algorithm computes

- the free space graph, identifies start and end vertices and then
- runs Dijkstra’s algorithm to find a shortest path from a start to an end vertex.

We modify this algorithm to run in stages, each stage corresponding to one trajectory edge. Let the trajectory be $p_1,\ldots,p_n$, and denote by $\overline{p_ip_{i+1}}$ the trajectory edge between p_i and p_{i+1}.

Stage 1

Dijkstra's algorithm is seeded with the start free space graph vertices (p_1, e), where e is any road network edge (or a part of it) in $A(p_1)$. Then, Dijkstra's algorithm is executed on the part of the free space graph induced by the free space graph vertices (p_1, e), $(v, \overline{p_1p_2})$, (p_2, e'), where v is any vertex and e any edge in $A(\overline{p_1p_2})$, and e' is any edge in $A(p_2)$.

This computes shortest path values for shortest paths from a start free space graph vertex (p_1, e) to any vertex (p_2, e').

Stage *i for $2 \le i \le n-1$.*

Dijkstra's algorithm is seeded with the shortest path values that have been computed in stage $(i-1)$ for all free space graph vertices (p_i, e'). Then, Dijkstra's algorithm is executed on the part of the free space graph induced by the free space graph vertices (p_i, e), $(v, \overline{p_i p_{i+1}})$, (p_{i+1}, e'), where v is any vertex and e any edge in $A(\overline{p_i p_{i+1}})$, and e' is any edge in $A(p_{i+1})$. This computes shortest path values for shortest paths from a start free space graph vertex (p_1, e) to any vertex (p_{i+1}, e').

Traceback

All stages store predecessor trees. In the end one shortest path is constructed by tracing back through the predecessor trees of all stages. See Figure 8 for an illustration of the active regions in the road network. Clearly, this algorithm computes a shortest path in a restricted free space graph. We claim that it also solves the error-aware map-matching task.

Theorem 2 *The Adaptive Clipping algorithm solves the error-aware map-matching task as defined in Section 4.1. Moreover, amongst all curves in the road network that fulfil the properties of Lemma 1, the Adaptive Clipping algorithm finds a curve with minimum weak Fréchet distance.*

The Proof can be found in [7].

3.3.3 Running Time

As for the running time, making some simplifying assumptions, the running time of the algorithm is $O(n\log n)$, with n being the size of the vehicle trajectory in terms of sampled GPS positions. This running time represents a considerable improvement over the original measure of $O(mn\log mn)$, with m corresponding to the size of the road network!

4. Performance and Quality Assessment

An important aspect in developing a map-matching algorithm is to assess its performance, both, in terms of running time and quality. The later argument is important in that we do not want random map-matching results that will result in useless travel times.

The evaluation of the map-matching algorithm was conducted in two steps. One, from a *micro-level perspective* a small, but significant set of trajectory curves was used and the map-matching result was examined with respect to its quality. This step can be seen in the context of low-level debugging of the algorithm.

Second, from a *macro-level perspective*, a larger number of trajectories was map-matched and the overall result was inspected as well as the time it took to compute it was evaluated from a qualitative point of view.

4.1 Micro-level Experimentation

4.1.1 Experimental setup

The tracking data used in the experiments was obtained by sampling vehicle movements at a rate of 30 seconds. The dataset consists of 27 vehicle trajectories consisting of a total of 15k position samples. The smallest and the largest trajectory consist of 207 and 1003 edges, respectively. The tracking data was collected in the municipal area of Athens (40x40km). The respective road network consists of 150000 edges. The implementation platform for the various map-matching algorithms is Java 1.5 in connection with an Oracle database. The Adaptive Clipping algorithm assumes a maximum speed of $v = 80km/h$ and a measurement error of $8m$.

In the experiments, the performance of our global map-matching algorithm is compared to an incremental algorithm as described in Section 3 (cf. also [3]).

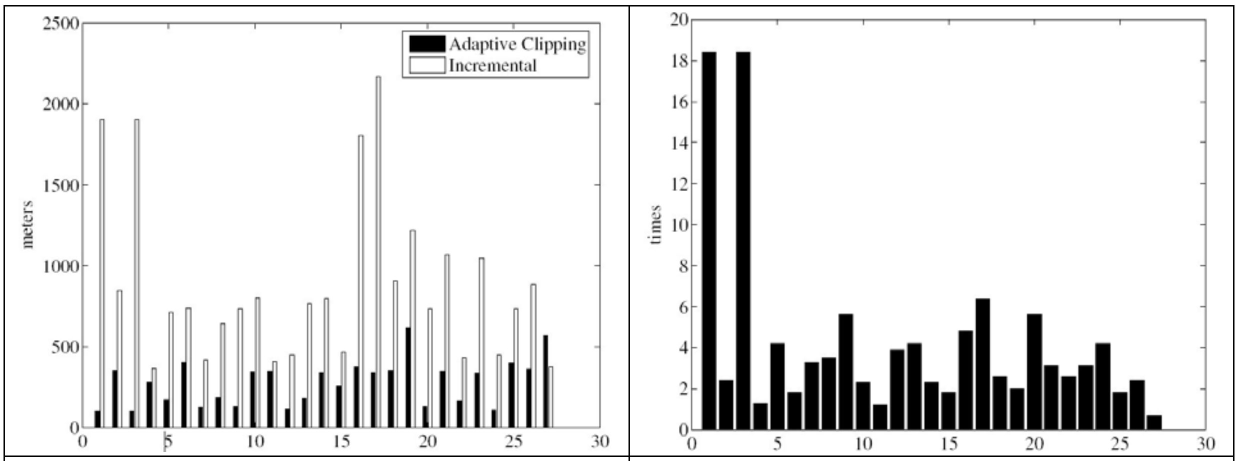
In all charts that follow below, the sorting order of the trajectories is according to the running time of the Adaptive Clipping algorithm for the respective matching result (cf. Figure Figure 9).

4.1.2 Accuracy

To assess the map-matching algorithm, map-matching results for the tracking data were evaluated using the (i) *weak Fréchet distance* and (ii) the *average Fréchet distance* with sampling distance 2*m* (cf. [3]).

Using the *weak Fréchet distance* to evaluate the matching result shows the distance between the original GPS-based vehicle trajectory and the path that the map-matching algorithm found in the road network most closely matching the vehicle trajectory.

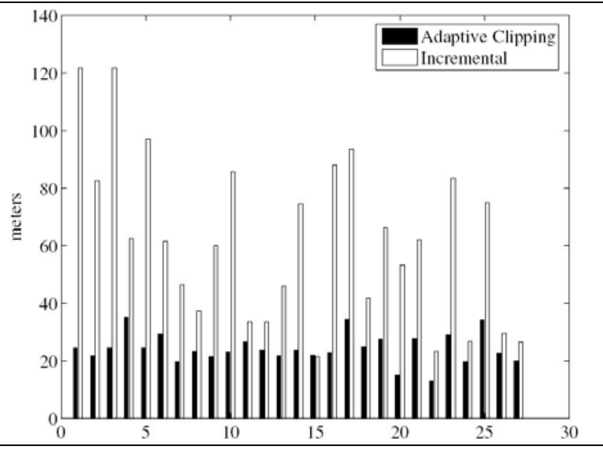
The *average Fréchet distance* computes an average of the distance between matched points. This is in contrast to the Fréchet distance itself which compute the maximum.



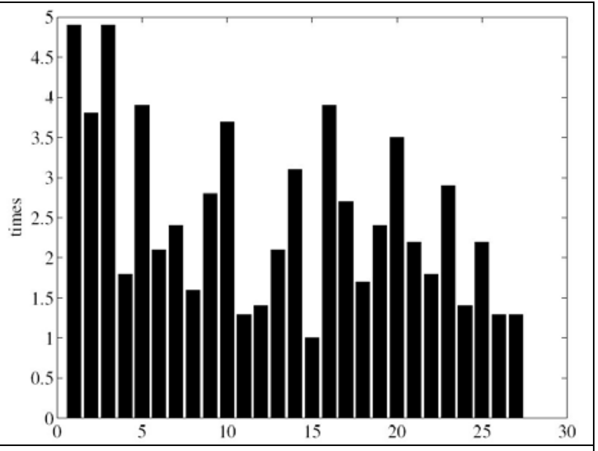
(a)	(b)

Figure 9: Weak Fréchet distance quality measure: (a) absolute and (b) relative

Figure 9 and Figure 10 determine *the quality of the matching result* by means of the weak Fréchet and the average Fréchet distance, respectively. Figures Figure 9(a) and Figure 10(a) give the absolute distances, whereas the relative quality is shown in Figures Figure 9(b) and Figure 10(b). For the relative distance, the Adaptive Clipping algorithm is the benchmark. What can be readily observed is that the Adaptive Clipping algorithm produces matching results of superior quality. The measured weak Fréchet distance for the Incremental algorithm is an average of 4 times larger (worst case 18 times). Using the average Fréchet distance, it is still on average 1.5 times larger (worst case 5 times). The absolute distances indicate that the trajectory and the matched curve exhibit (i) a worst case distance (weak Fréchet) of up to 2000*m* and 600*m* and (ii) an average distance (average Fréchet) of up to 120*m* and 35*m* for the Incremental and the Adaptive Clipping algorithm, respectively. This rather large distances for the Incremental algorithm are the result of a poor match for the initial trajectory points.



(a)

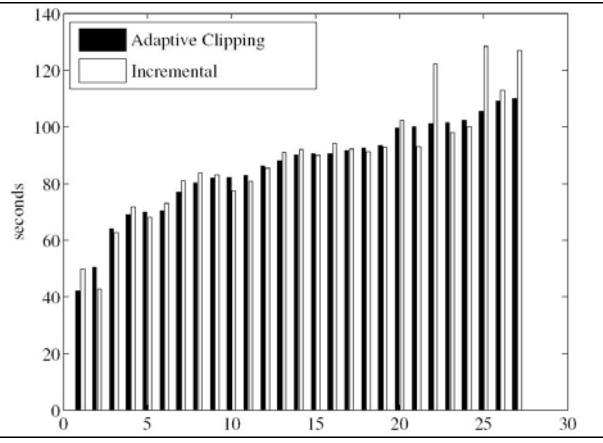


(b)

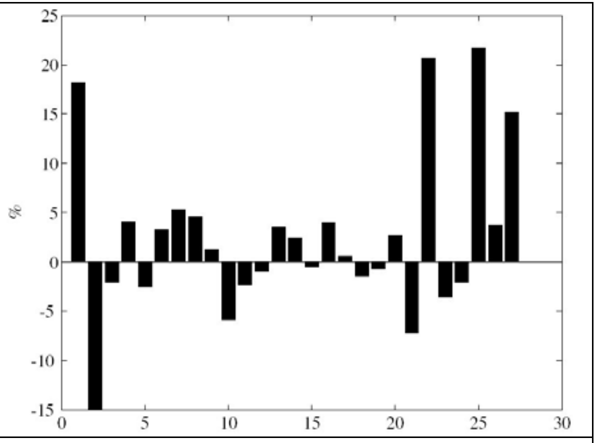
Figure 10: Average Fréchet distance quality measure: (a) absolute and (b) relative

4.1.3 Running Time

Figure 11 gives the overall running times comparing the two algorithms. Again, absolute and relative times (percentages) are shown, with the Adaptive Clipping algorithm serving as a benchmark for the Incremental algorithm.



(a) absolute



(b) relative

Figure 11: Total running times of the map-matching algorithms

An important objective for this work was to use it for real-time map-matching of tracking data. Figure 11(a) shows that matching the smallest trajectory, which consists of 207 position samples takes 40*s* (50*s*). With a sampling rate of 30*s*, this data was collected over a period of 6210*s* (1h 40min). Thus, the sampling rate could be as low as 0.2*s* (0.25*s*) (for 207 samples, the collection period would then be equal to the running times of the map-matching algorithms) for the Adaptive Clipping (Incremental algorithm) still to be able to keep up with the incoming tracking data stream.

In comparing the two algorithms, surprisingly in almost all cases, Adaptive Clipping runs faster than the Incremental algorithm (up to 20%). According to the asymptotic running times of the algorithms, Adaptive Clipping - *O*(*n*log*n*) and Incremental algorithm - *O*(*n*) [3], with *n* being the number of trajectory edges, the opposite was expected. However, the *O*(*n*) time of the Incremental algorithm absorbs the local look-ahead cost (constant factor), which in practice significantly affects the running time.

Overall, the experiments show that Adaptive Clipping with respect to the Incremental algorithm (i) produces better matching results by (ii) in most cases having a lower running time. Both algorithms are further well suited to perform real-time map-matching for tracking data collected at a typical sampling rate of 30s. *The sampling rate could be as low as 0.2s to still fulfill the real-time requirement!*

4.2 Macro-level Experimentation

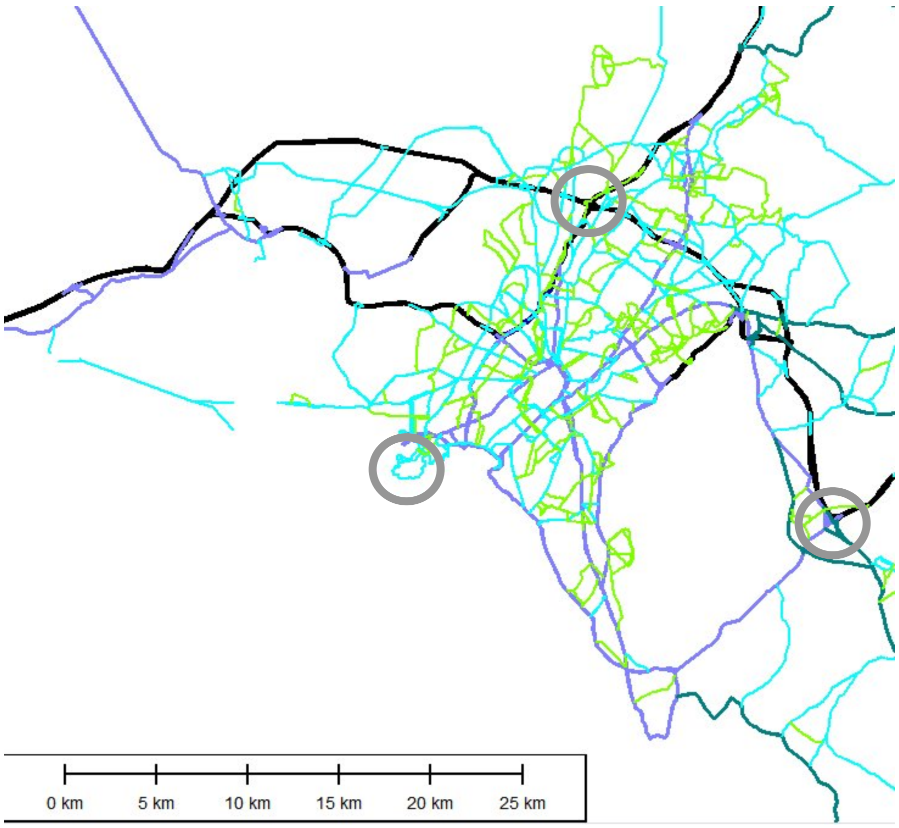
This experiment aims at map-matching large amounts of historic vehicle tracking data to the road network and assessing the result. A dataset containing 2.5M position samples covering the greater Athens, Greece area is used. This data is the equivalent of roughly 60 vehicles collecting data over a period of six months (first half of 2007). The GPS sampling rate was again 30s.

Figure 12 shows the entire Athens road network and the number of traversals in the road network. Figure 12(a) shows a road network roughly containing 250k edges (links between junctions) and 500k road network edges describing the exact geometry of the road network. Different colors mark the various road categories. Highways are shown in black and generally, the thicker a line, the higher the road category it represents. In the specific image only the higher category roads (up to category 6 out of 9 in total, with 9 being neighborhood roads) are shown.

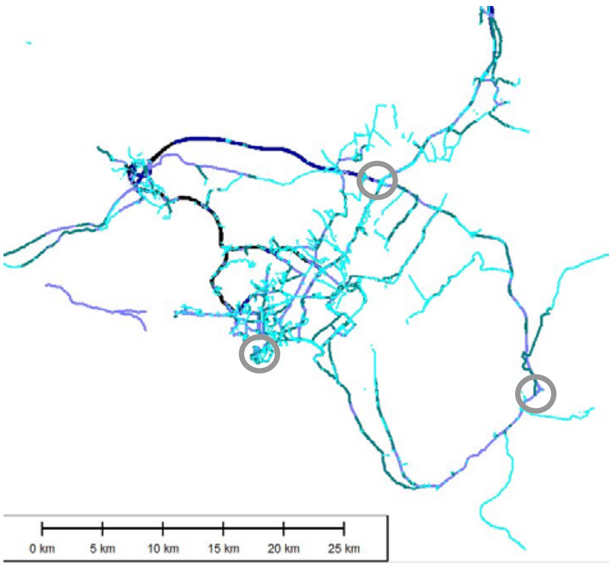
Three circles are used to mark points of reference that also appear in the traversal visualization of Figure 12 (b) and (c). In Figure 12(b) all road network edges with at least 20 traversals are colored. Black represents edges with at least 1000 traversals. These edges correspond to roads of higher categories as shown in Figure 12(a). Effectively and what was to be expected, there is a correspondence between number of traversals and the category of the specific road segment in question.

Figure 12(c) shows only edges with at least 100 traversals, i.e., fewer colored edges are expected. The colored edges are a subset of the edges shown in Figure 12(b). Major roads are clearly visible as those with many traversals.

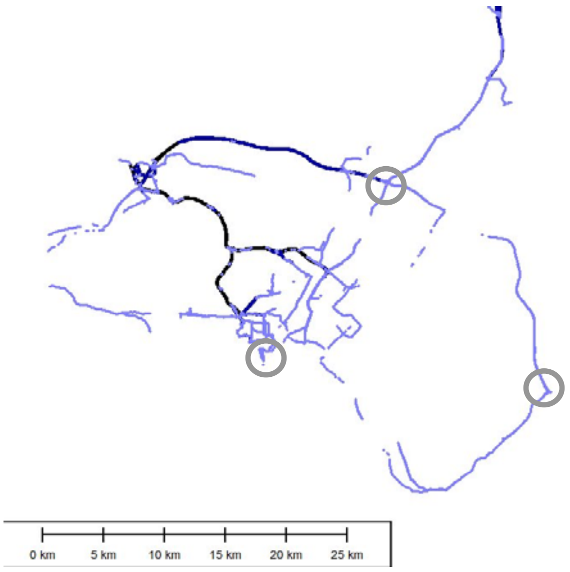
4.2.1 Results Overview



(a) Athens road network



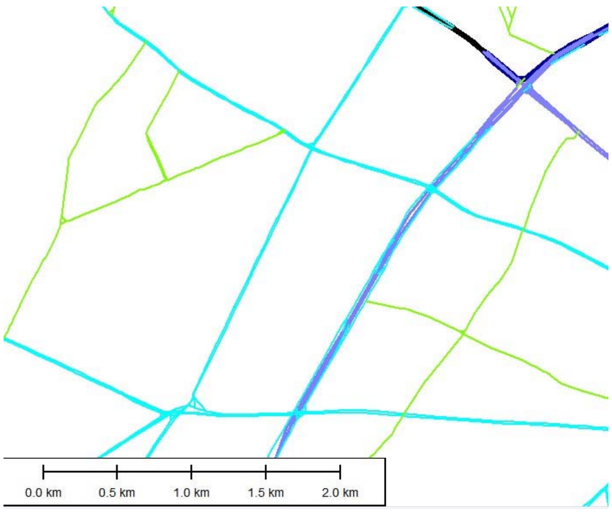
(b) links with > 20 traversals



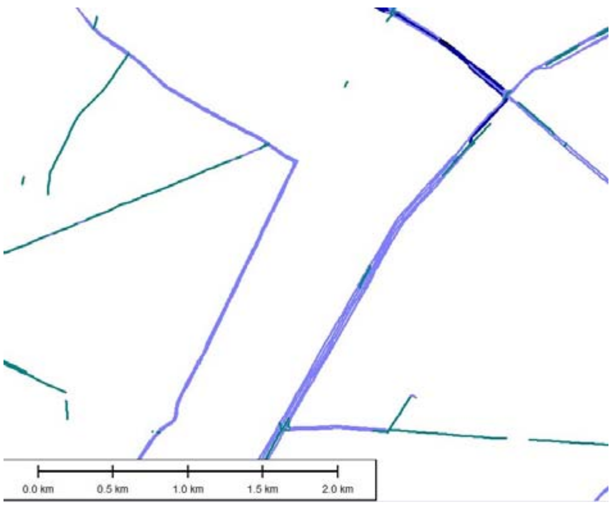
(c) links with > 100 traversals

Figure 12: Entire Athens road network and number of traversals based on the tracking data

Figure 13 shows an excerpt of the road network Figure 13(a) and its related number of traversals Figure 13(b). Again, as can be expected, roads of higher category (black, purple and also blue) are traversed more often.



(b) Road network detail



(c) traversals (>100)

Figure 13: Road network detail and number of traversals

5. Map Matching Component Technical Documentation

The map matching component as developed in TRACK&TRADE and described in the following encapsulates the global Fréchet-based map matching algorithm as described previously. The algorithm maps a vehicle's trajectory, i.e. a sequence of GPS position samples onto a sequence of road segments in a road map. Furthermore, by utilizing the timestamps of the GPS position samples, the component can be used to allocate the total time of the vehicle's trip to the different road segments it passed through (travel times per edge). What follows is (i) a description of the specifics of converting an algorithm to a software component and (ii) an overview of the class structure of the software in combination with use cases.

5.1 Implementing a Robust Map-Matching Algorithm

The global map-matching algorithm as described in Section 3 forms the basis for the implementation developed in the TRACK&TRADE project. However, to create a robust implementation a substantial development effort was needed with the most important aspects described in the following.

5.1.1 Consideration of a direct road network

The original algorithm was only tested using an undirected road network. With the availability of good quality and directed road networks for Athens, Berlin and Vienna, the algorithm required a substantial effort to adapt it.

This effort is of outmost importance since cars would otherwise be permitted to be mapped on roads leading in the opposite direction and falsify the overall collected travel times.

5.1.2 Quality of the road network

During the testing of the map-matching algorithm, it was evident that the road network used for map-matching (typically the current one) does not always reflect reality, or, match the road network the vehicle was moving in during the time of data collection.

The map-matching algorithm had to be equipped with a reliable algorithm (heuristic) that detects such mismatches and “skips” portions of the vehicle tracking data that fall outside the road network.

5.1.3 Computation speed

The code base of the map-matching algorithm that was initially available had not been developed for a large-scale matching task, but rather for experimentation in “laboratory conditions”.

Here, besides the robustness of the algorithm, computation speed was of outmost importance. The code of the algorithm was improved towards using efficient data structures for storing the road network data and generally relying heavily on the use of indexes in main-memory as well as for the data stored on disk.

5.1.4 Online computation

The map-matching algorithm as described in our project matches essentially one curve, the tracking data trajectory, to another curve, the road network path that most closely resembles the tracking trajectory.

In our case of matching online data no such trajectory would exist, i.e., only the current position in the “worst” case. However, as stated in D1.1, the tracking data is communicated to the collection site at 5-minute intervals, i.e., as a 5-minute trajectory. Experimentation showed that 5-minute trajectories can be matched with the same accuracy as longer trajectories. Essentially, to match GPS tracking data, in the case of any algorithm some historic data (trajectory) needs to be available as a simple point does not represent a movement geometry that can be accurately matched to a road network. This is in contrast to navigation systems in cars, which utilize in a addition to continuous GPS tracking, dead reckoning and any information available from other sensors in the car (speedometer, steering, ABS).

5.1.5 Map-matching software framework

The task requiring most effort was to transform code created for scientific experimentation into a software framework.

The following section describes the software developed in this process. Specific use cases are given to showcase the various classes and their interfaces.

5.2 Use cases

The map matching component is intended to be used by application developers when building systems where map matching functionality or travel time extraction is required, given data sets of GPS position samples. These two use cases are depicted in Figure 14.

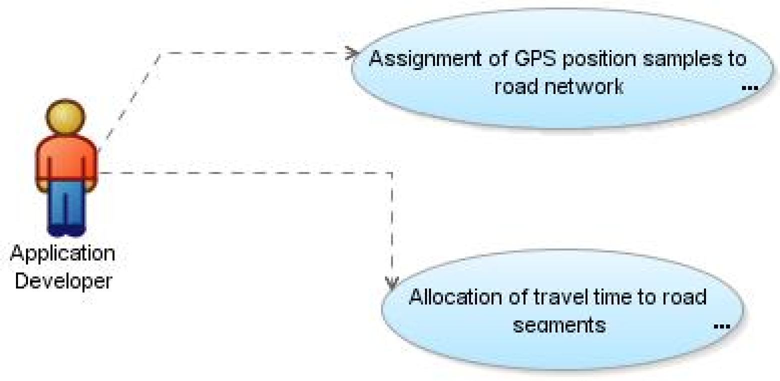

Assignment of GPS position samples to road network
Application Developer
Allocation of travel time to road segments

Figure 14 – Use case diagram for the map matching component

5.3 Main elements of the Map Matching Component

The user of the map matching component should be aware of the elements presented in the class diagram of Figure 15. MatchTask class is used to perform runs of the map matching algorithm under different input configurations and get the outputs in a structured way. Instances of TrajectoryPoint and StreetNetwork classes are used as inputs for the map matching process and instances of the MatchAssignmentRecord and TravelTimeRecord classes are used as map matching results placeholders. Note that the self-referencing association of the StreetNetwork class exists only because its implementation follows the singleton pattern - the class is responsible to create its own single instance and provide only references to instances of other classes.

In the following we present a more detailed description of the classes of Figure 15 plus a typical component usage scenario both in terms of a sequence diagram and of a code excerpt.

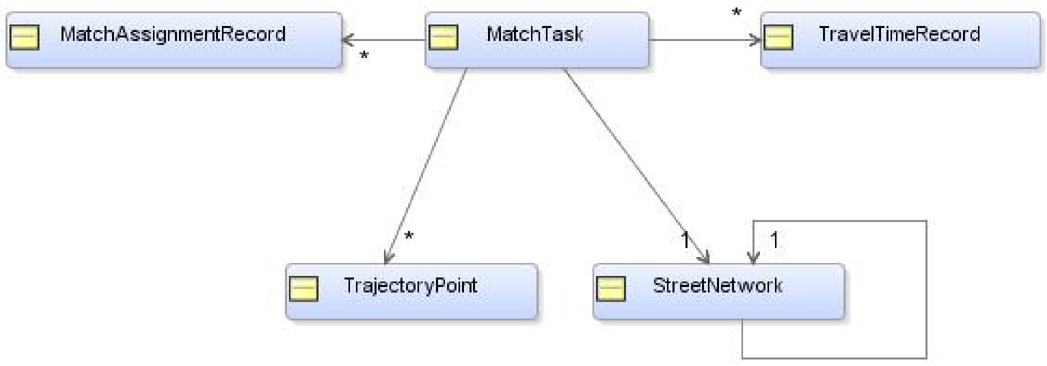


Figure 15 – Main class diagram of the map matching component

5.3.1 Class TrajectoryPoint

This class models a GPS device position sample. Its data members are shown in Table 1. The only methods of the class are accessors of its data members. A Vector of TrajectoryPoint objects is passed as input to the MatchTask class before the initiation of the map matching process.

Class attribute	**Comment**
double x	Position sample X coordinate
double y	Position sample Y coordinate
String srid	Spatial Reference System Identifier. Valid values are the ones of the EPSG database (http://www.epsg.org/)
int speed	Speed as measured by the GPS device in meters/second.
int direction	Direction as measured by the GPS device in degrees.
int year, month, day, hour, min, sec	Timestamp of the position sample analyzed in its parts.
String timeZone	The time zone where the timestamp referes to.
long companyId	Id of the company that provides us this trajectory point
long vehicleId	Id of the vehicle that corresponds to this position sample.
int vehicleTypeId	An identifier that denotes the type of the vehicle.

Table 1 – Data members of TrajectoryPoint class

5.3.2 Class StreetNetwork

StreetNetwork models a road map as a directed graph. The network is persistently stored in an Oracle Spatial database according to the Oracle Spatial Network model. StreetNetwork has methods that allow both full loading of the map into main memory and partial loading/unloading portions of the map based on a partitioning scheme defined with the hep of rectangular “tiles”. In all cases, memory management is transparent to the user of the class. The only thing that is required is to specify the size of the tiling scheme, i.e. the number of tiles in X dimension and the number of tiles in Y dimension, when creating an instance of the class. If a 1x1 tiling scheme is specified, then the whole map is going to be loaded in main memory.

The class provides methods for accessing Nodes and Edges of the Network that are used by the map matching algorithm. The user of the map matching component only needs to properly initialize a StreetNetwork object and pass it to a MatchTask object using the method MatchTask.setStreetNetwork(). An instance of StreetNetwork can not be created directly, since its implementation follows the singleton pattern. Instead, the method StreetNetwork.getInstance() should be used. The arguments of this method are shown in Table 2.

getInstance arguments	**comment**
boolean directedGraph	Specifies if the network is directed or not.
Connection con	A pre-initialized and open connection to an Oracle Spatial database.

String netname	The network name as defined in Oracle Spatial.
String net_srid	The spatial reference system identifier of the Oracle Spatial Network.
int number_of_tiles_x_dim	Number of tiles in the X dimension
int number_of_tiles_y_dim	Number of tiles in the Y dimension

Table 2 – StreetNetwork's getInstance() arguments

5.3.3 Class MatchTask

MatchTask encapsulates the map matching algorithm and travel time allocation algorithm into the runTask() method. Before calling runTask(), setTrajectory() and setStreetNetwork() have to called first. After calling runTask(), the methods getMatchAssignments() and getTravelTimes() return a Vector of MatchAssignmentRecord instances and TravelTimeRecord instances, respectively. The last two structures are defined in the following paragraphs.

5.3.4 Class MatchAssignmentRecord

The map matching process produces a Vector of MatchAssignmentRecord objects. Each object represents one (and only one) of the following:

- An assignment of a GPS position sample (*"GPS point"*) to a specific spatial position on an edge of the road map. We refer to this specific position as *"Position on the edge"*.
- An assignment of a node of the road map to a "GPS egde", meaning the edge that is formed by two consequetive GPS position samples. We refer to the position on the "GPS edge" as *"Leash Point"*. In this cases we record both the timestamps of the "GPS edge" start and end points.

We refer to the edges and the nodes of the road map using the unique identifiers of the edges and nodes as they are stored in the Oracle Spatial database.

In some cases, the trajectory cannot be matched as a whole to the street network, e.g. due to time gaps in the seqeunce of the GPS position samles. In these cases, the input trajectory is broken into *"trajectory parts"* and a trajectory part identifier is used for each part.

The attributes of the MatchAssignmentRecord class are shown in Table 3. The only methods of the class are accessors of its attributes.

Class attribute	**Comment**
long companyId	Identifier of the company that provided the data that were matched to the street netwotk.
long vehicleId	id of the vehicle, refering to the company id, that produced the GPS position samples

long trajectoryPartId	Trajectory part (curve piece) id
long sequenceNumber	A number denoting the sequence of the gps and leash points inside a trajectory part. Mostly used for debugging.
boolean isLeashPoint	A flag that detemines that the match assignment involves a "leash point" (true value; point between two gps points) or a gps point (false value).
double gpsOrLeashPointX	X coordinate of gps point or of leash point
double gpsOrLeashPointY	Y coordinate of gps point or of leash point
String srid	Spatial Reference System Identifier
String gpsPointTimeStamp	Timestamp denoting when the vehicle is at the specific gps or leash point. // The format of the string must be like "1997-01-31 09:26:50" for seconds precision
String leashPointStartTimeStamp	In case of leash points we record two timestamps, those of the gps points "enclosing" the leash point.
String leashPointEndTimeStamp	In case of leash points we record two timestamps, those of the gps points "enclosing" the leash point.
String gpsOrLeashPointTimeZone	Time zone in the format like "+2:00"
int matchAssignmentType	Match assignment type identifier - An integer denoting the methodology used for match assignment – Currenlty only one methodology is used.
long edgeOrVertexId	Matched edge or vertex identifier of the map.If the assgiment concerns a gps point (is_leash_point = false) this parameters refers to an edge (link) of the map. If the assgiment concerns a leash point (is_leash_point == true) this parameters refers to a vertex (node) of the map.
double xPositionOnTheEdge	Matched X position on an edge or a vertex of the map. If the assgiment concerns a gps point (is_leash_point == false) this parameter refers to X position on an edge (link) of the map, where the gps point has been matched. If the assgiment concerns a leash point (is_leash_point == true) this parameter refers the vertex X coordinate (node) of the map, where the leash point has been matched.
double yPositionOnTheEdge	Matched Y position on an edge or a vertex of the map. If the assgiment concerns a gps point (is_leash_point == false) this parameters refers to Y position on an edge (link) of the map, where the gps

	point has been matched. If the assgiment concerns a leash point (is_leash_point == true) this parameter refers the vertex Y coordinate (node) of the map, where the leash point has been matched.
int direction	Direction of the matched edge traversal by the vehicle. 0 if we enter from edge start node (vertex), 1 if we enter from edge end node (vertex). The parameter has no meaning when a leash point is matched to a map vertex.
double confidence	Confidence is a double value used as a confidence indicator.

Table 3 - Attributes of MatchAssignmentRecord class

5.3.5 Class TravelTimeRecord

After a sequence of GPS samples has been mapped to positions on an edge sequence of the road network, we can calculate the time spent while driving through an edge using the timestamps of the GPS sample points and linear interpolation. Class TravelTimeRecord records for every edge involed a timestamp denoting the time a specific vehicle entered that edge and the duration of the edge’s traversal. The attributes of the TravelTimeRecord class are shown in Table 4. The only methods of the class are accessors of its attributes.

Class attribute	**Comment**
long companyId	Identifier of the company that provides us with data for the specific vehicle used for extracting travel time from its trajectory
long vehicleId	Identifier of the vehicle used for extracting travel time from its trajectory. The id is unique in the context of the companyId.
int vehicleTypeId	The type of vehicle used for extracting travel time from its trajectory.
int travelTimeAssignmentType	An integer indicating the methodology used for producing the travel times. Currently, only one method is used.
long edgeId	The street network edge id of the travel time assignment.
String entranceTimestamp	Timestamp indicating the time we entered the edge according to the matching proccess. The format of the string must be like "1997-01-31 09:26:50" for seconds precision or "1997-01-31 09:26:50.124" that includes fraction of second.
String timeZone	Time zone in a format like "+2:00"

double travelTime	Travel time in seconds.
int direction	Direction of the matched edge traversal by the vehicle. 0 if we enter from edge start node (vertex), 1 if we enter from edge end node (vertex). The parameter has no meaning when a leash point is matched to a map vertex.
double confidence	Confidence is a double value used as a confidence indicator.

Table 4 – Attributes of class TravelTimeRecord

5.4 A typical component usage scenario

According to the above, the usage of the map matching component involves three steps:

- Input initialization.
- Running the task.
- Getting the output.

These steps are illustrated in the sequence diagram of Figure 16.

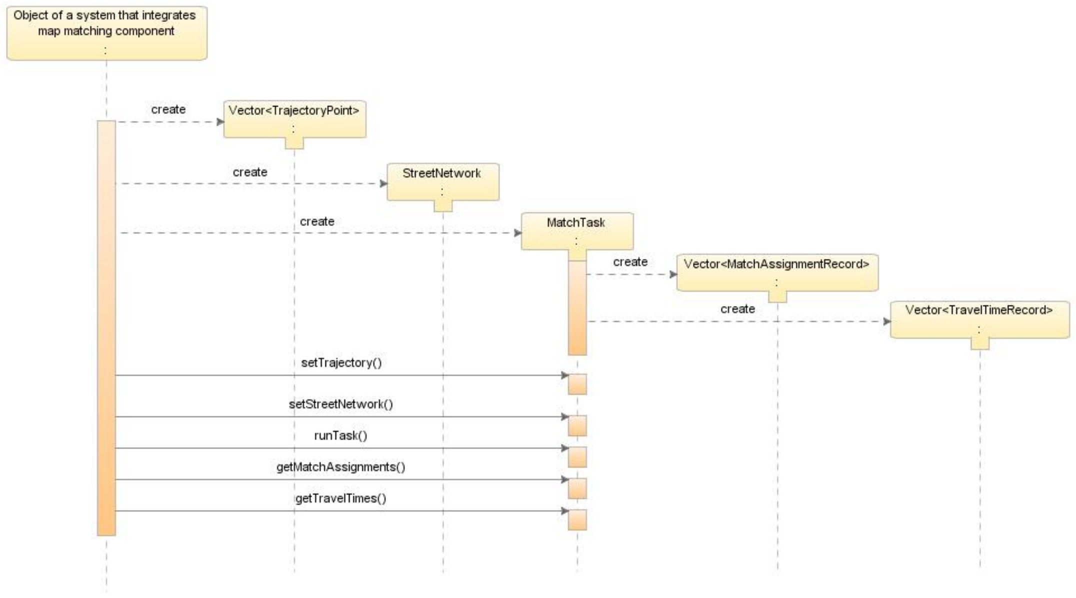


Figure 16 – Sequence diagram

References

[1] H. Alt, A. Efrat, G. Rote, and C. Wenk. Matching planar maps. J. of Algorithms, 49:262–283, 2003.

[2] H. Alt and M. Godau. Computing the Fréchet distance between two polygonal curves. Int. J. Comput. Geom. Appl., 5:75–91, 1995.

[3] S. Brakatsoulas, D. Pfoser, R. Salas, and C. Wenk. On map-matching vehicle tracking data. In Proc. 31st VLDB Conference, pages 853–864, 2005.

[4] A. Leick. GPS Satellite Surveying. John Wiley & Sons, Inc., 1995.

[5] M. Fréchet. Sur quelques points du calcul fonctionnel. Rendiconti del circolo Mathematico di Palermo, 22:1–74, 1906.

[6] D. Pfoser and C. S. Jensen. Capturing the uncertainty of moving-object representations. In Proc. 6th SSD conf., pages 111–132, 1999.

[7] C. Wenk, R. Salas, D. Pfoser. Addressing the Need for Map-Matching Speed: Localizing Globalb Curve-Matching Algorithms. In Proc. 18th SSDBM conf., pages 379-388, 2006.