

# Addressing the Need for Map-Matching Speed: Localizing Global Curve-Matching Algorithms

Carola Wenk<sup>1</sup>, Randall Salas<sup>1</sup>, and Dieter Pfoser<sup>2</sup>

<sup>1</sup> Department of Computer Science  
University of Texas at San Antonio  
San Antonio, TX 78249-0667, USA  
{carola,rsalas}@cs.utsa.edu

<sup>2</sup> RA Computer Technology Institute  
Akteou 11  
GR-11851 Athens, Greece  
pfoser@cti.gr

**Abstract.** Tracking data becomes an increasingly available sensor data resource that can be used in a range of applications related to traffic assessment and prediction. Of critical importance in this context is the *amount* of (i) historic and (ii) current tracking data available (data per spatial area). Using data of varying quality (sampling rate and accuracy) requires sophisticated map-matching algorithms. Having current data requires fast algorithms. Currently only *fast Incremental* and *slow Global algorithms* exist that produce low and high-quality results, respectively. This work proposes a fast map-matching algorithm that at the same time gives quality guarantees for the result. While employing a global matching strategy, we exploit auxiliary knowledge about the tracking data to improve the computation speed. The classical global matching approach is to find among all possible paths in the road network one path that is the most similar to the curve represented by the tracking data. Our novel *Adaptive Clipping* algorithm exploits worst-case error measures associated with the tracking data to limit the portions of the road network that need to be considered in the matching process (output sensitive algorithm), while using the *weak Fréchet distance* to measure similarity between curves. An experimental evaluation shows that the Adaptive Clipping algorithm runs as fast (and often faster) as the Incremental Algorithm, and produces high quality matching results comparable to those of Global algorithms.

## 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 a cheap solution for traffic assessment and prediction.

Being a by-product of another application 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 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 work proposes a new map-matching technique that is based on the Global algorithms as presented in [3]. Such an algorithm tries to find a path in the road network that resembles the trajectory of the tracking data based on the Fréchet distance measure between the curves [2][6]. However, due to its global nature, it considers the entire road network graph, the algorithm is slow but produces highly accurate matching results (cf. also Section 3 on the details of the algorithm). This work exploits worst-case error estimates for the tracking data in order to prune the road network graph in a provably correct manner yielding a fast algorithm and still guaranteeing highly accurate results. The resulting algorithm is termed *Adaptive Clipping*.

The competitive quality and running time of Adaptive Clipping is established in an experimental evaluation comparing the algorithm to the Incremental algorithm [3](cf. Section 3).

Work in the area of map-matching vehicle positions exists towards augmenting GPS positioning with other methods such as dead reckoning to reduce the measurement error and to achieve better map-matching results [11]. Greenfeld [8] introduces a map-matching strategy based on distance and orientation that does not assert any further knowledge about the movement besides the position samples. Civilis et al. [4][5] in their work on location update techniques for the tracking of users in location-based services introduce a map-matching algorithm that is based on edge distance and direction similar to [8]. The tracking data itself is obtained by using an active sampling technique based on predicted and measured positions. By controlling the sampling rate, the sampling error can be kept minimal and the map-matching algorithm is presented with an optimal dataset. Yin and Wolfson [15] propose an algorithm based on a weighted graph representation of the road network in which the weights of each edge represent the distance of the edge to the trajectory. The matched trajectory in the road network is found by using a Dijkstra shortest-path algorithm for the weighted graph. This algorithm is based on a measure related to the average Fréchet distance, however no overall quality guarantee on the matched curve is given. The authors claim that the algorithm produces high quality matches, however details on the data set, such as type and size are missing.

The outline of this paper is as follows. Section 2 discusses tracking data, its use for traffic assessment and the resulting requirements for map-matching algorithms. Section 3 briefly introduces the Incremental and the Global map-matching algorithms. The concept of output sensitive algorithms as a means to improve the running time is introduced and Section 4 details the Adaptive Clipping algorithm, an implementation of this concept. Section 5 shows the outcome of the experimental evaluation and, finally, Section 6 gives conclusions and directions for future research.

## 2 Background

Currently, the sensor data used for traffic assessment and prediction stems from sensors which are deployed throughout the road network to collect data on the traffic. For cost reasons and technical limitations, sensors cannot cover the network in its entirety. The most commonly used stationary type of sensor is an inductive loop detector, which is an expensive solution in case the whole urban area has to be covered and provides only incomplete and inaccurate data due to technology limitations.

For a couple of years, a new technology has been discovered to overcome that problem. Floating car data (FCD) is already a well known technology for various ITS application and has become an interesting complement to conventional traffic sensors. Floating car data (FCD) refers to using data generated by one vehicle as a sample to assess to overall traffic conditions (“cork swimming in the river”). Typically these data comprise basic vehicle telemetry such as speed direction and most importantly *the position of the vehicle*. Having large amounts of vehicles collecting such data for a given spatial area such as a city (e.g., taxis, public transport, utility vehicles, private vehicles) will create an accurate picture of the traffic condition in time and space [12]. The trajectories of vehicles are typically detected by GPS positioning. Various pilot projects based on GPS have been started delivering floating car data for traffic information services. However, on the whole, commercial activities suffered from low penetration of probe vehicles due to the low yearly mileage of private vehicles and the overall size of the general road network, high investment cost to equip vehicles with positioning technology, and high communication cost. These limitations to the utilization of FCD can be overcome by collecting such data from any type of vehicle that is available, e.g., taxis, delivery trucks, public transport, emergency vehicles, etc. In all these cases, FCD is produced as a by-product of another application (e.g., fleet management). Being a by-product has serious implications for the data quality, e.g., infrequent position samples. To still be able to use this data, sophisticated map-matching algorithms are needed to provide *accurate matches* of the tracking data to a path in the road network.

A technological means used to assess and predict traffic conditions is the *Dynamic Travel Time Map* (DTTM)[10]. Such a system can be used to overcome the problem of an unreliable travel time database for road network used in conventional routing and navigation systems. Per default, only static travel-times derived from road categories and speed limits are used to calculate the fastest or shortest path for a given trip. DTTMs aim at supplying on-the-fly computed speed types based on collections of large amounts of travel time data derived from FCD. The outdatedness of travel times is of critical importance here. To utilize FCD best, travel times have to be computed on-the-fly as *FCD is collected in real time*.

Based on the above requirements, *fast* and *accurate map-matching algorithms* are needed to utilize vehicle tracking data for traffic assessment and related applications. Figure 1 presents a schema illustrating the relationships between FCD data, DTTM and traffic-related applications. Map-matching algorithms play the essential role of populating the DTTM.

### 3 Available Map-Matching Algorithms

Two types of algorithms for map-matching vehicle-tracking data have been introduced in [3]. The *Incremental algorithm* matches the position samples sequentially by in each case considering only the connected portion of the road network, i.e., outgoing edges from the previously matched edge. The *Global algorithms* try to find a path in the road network that resembles the trajectory, i.e., pursuing a global strategy rather than a local one as in the case of the Incremental algorithm.

In terms of quality and speed, the Global algorithms produce high-quality matching results but are slow compared to the Incremental algorithm. Thus, as we will see

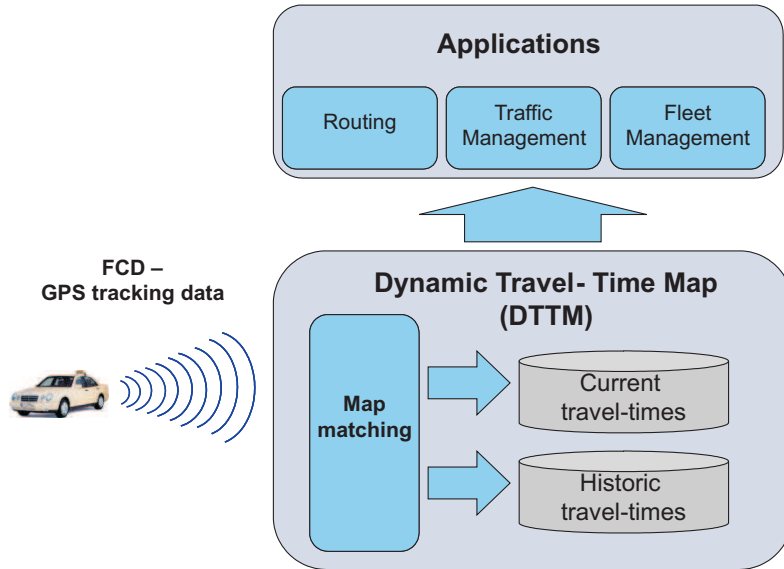


Fig. 1. Dynamic travel-time maps

in Section 4, the objective in this work is to *improve the speed* of a Global algorithm by *preserving its accuracy*.

### 3.1 Incremental Algorithm

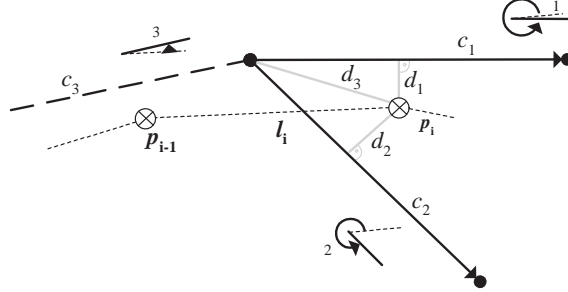
Using a greedy strategy based on local spatial characteristics, position samples comprising the trajectory are matched sequentially by in each case comparing the geometry of a portion of the trajectory to selected paths in the road network.

Given a series of position samples representing a vehicle trajectory, the map-matching algorithm pursues a position-by-position sample and edge-by-edge strategy. To match a position  $p_1$  to a road network edge, given that its previous position  $p_{i-1}$  has already been matched, the algorithm proceeds as follows (c.f. Figure 2). First, the candidate edges to be matched to the current position are identified as the set of the incident edges “exiting” the last matched edge (including also the matched edge itself). In Figure 2, these edges are labeled  $c_1, c_2$  and  $c_3$ , with  $c_3$  being the edge matched to  $p_{i-1}$ .

Two similarity measures are used to evaluate the candidate edges [8]. The measure  $s_d$  reflecting the *distance* from the position sample to the various edges is computed based on the weighted line segment distance<sup>3</sup>,  $d$ , of  $p_i$  from each candidate,  $c_j$ , using the scaling factors  $\mu_d$  and  $n_d$  as

$$s_d(p_i, c_j) = \mu_d - a \cdot d(p_i, c_j)^{n_d}.$$

<sup>3</sup> The *line segment distance* of a point to a line is either the perpendicular line distance if the projection of the point onto the line segment is between its endpoints, or, otherwise, it is the distance of the point to the closest endpoint of the line segment.



**Fig. 2.** Incremental map-matching example.

The measure  $s_\alpha$  reflects the *orientation* of the trajectory with respect to the candidate edge. It is computed based on the angle difference  $\alpha_{i,j}$  between the directed candidate edge  $c_j$  and the directed line segment  $l_i = \overrightarrow{p_{i-1}, p_i}$ , using the scaling factors  $\mu_\alpha$  and  $n_\alpha$  as

$$s_\alpha(p_i, c_j) = \mu_\alpha \cdot \cos(\alpha_{i,j})^{n_\alpha}.$$

The scaling factors  $\mu_{[d|\alpha]}$  and  $n_{[d|\alpha]}$  represent the maximum score (boosting distance or orientation) and a power parameter (rate of increase with deviation), respectively. For the specific dataset used in this work, we empirically established the following parameter settings  $\mu_d = 10$ ,  $a = 0.17$ ,  $n_d = 1.4$ , and  $\mu_\alpha = 10$ ,  $n_\alpha = 4$ .

The combined similarity measure is computed as the sum of the individual scores, i.e.,

$$s = s_\alpha + s_d.$$

*The higher the score of this measure, the better is the match.*

To improve this simple algorithm, a recursive “look-ahead” policy has been adopted. Recursively, for each candidate edge  $c_j$ , the score of the best candidate among its “exiting” edges  $c_{j,k}$  is calculated. This strategy aims at making a more global matching decision by exploring alternative branches rather than simple edges. Finally, only one choice is materialized in the matching result. The number of edges in the look-ahead is fixed. We established empirically that a look-ahead of 4 edges (the candidate edge plus three more edges) is optimal in terms of matching quality and time of computation.

To match a trajectory that consists of  $n$  position samples, the algorithm has to evaluate for each sample a finite number of adjacent road network edges  $a$  with a maximum look-ahead of  $l$  edges. Given that, both,  $a$  and  $l$  are essentially constants, the algorithm effectively runs in  $O(n)$  time.

### 3.2 Global Algorithms

Global map-matching algorithms find among all possible curves in the road network one curve that is the most similar to the vehicle trajectory. This requires a notion of similarity between two curves and has the advantage that the computed result

curve 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. Although at first sight considering all possible curves seems inefficient, there exist polynomial time algorithms [1, 3] for distance measures that are well-suited for curves.

The (*strong*) *Fréchet distance* and the *weak Fréchet distance*, see [2, 1, 3], are both well-suited distance measures for curves. Alt et al. [1] have designed an algorithm solving the global map-matching task using the Fréchet distance in  $O(mn \log^2 mn)$  time, where  $m$  is the total number of vertices and edges in the road network and  $n$  is the number of position samples of the vehicle trajectory. In [3] we gave an algorithm based on the *weak Fréchet distance* which runs in  $O(mn \log mn)$  time.

In the following sections we give basic definitions and sketch these algorithms. Since the algorithms have a quadratic runtime they are unfortunately too slow on real-world road networks. In Section 4 we show how we can localize the global map-matching algorithm based on the weak Fréchet distance from [3], which results in runtimes that are comparable to the incremental algorithm, while computing much better result curves.

**Fréchet and Weak Fréchet Distances** The Fréchet distance for two curves has been proposed by Fréchet [6]. 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*

Formally, the Fréchet distance between two curves  $f, g: [0, 1] \rightarrow \mathbb{R}^2$  is defined as

$$\delta_F(f, g) := \inf_{\alpha, \beta: [0, 1] \rightarrow [0, 1]} \max_{t \in [0, 1]} \|f(\alpha(t)) - g(\beta(t))\|,$$

where  $\alpha$  and  $\beta$  range over continuous and non-decreasing reparametrizations with  $\alpha(0) = \beta(0) = 0$  and  $\alpha(1) = \beta(1) = 1$  only. If we drop the requirement on  $\alpha$  and  $\beta$  to be non-decreasing, we obtain the *weak Fréchet distance*  $\tilde{\delta}_F(f, g)$  between  $f$  and  $g$ .

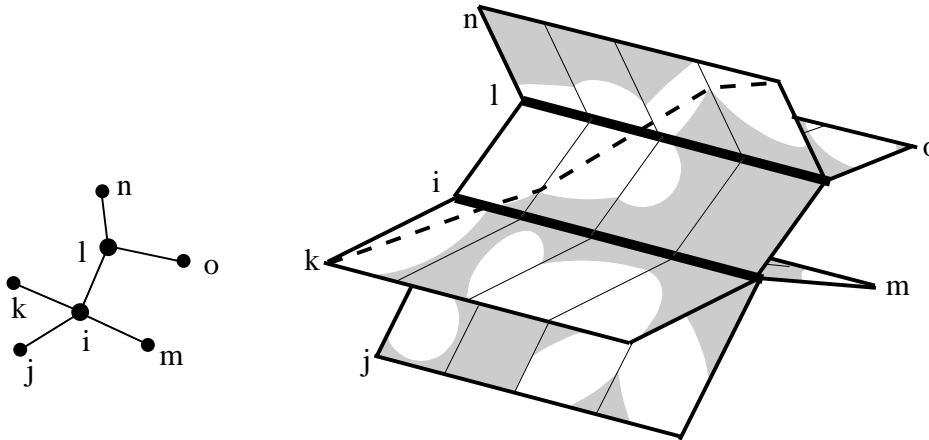
The (strong) Fréchet distance as well as the weak Fréchet distance are especially well-suited for the comparison of trajectories since they take the continuity of the curves into account, c.f. [1, 3]. Notice that  $\tilde{\delta}_F(f, g) \leq \delta_F(f, g)$ , since the weak Fréchet distance minimizes over more reparametrizations than the Fréchet distance.

**Freespace Diagram and Freespace Surface** The algorithms that compute the (weak or strong) Fréchet distance between two curves or which employ those distances in global map-matching algorithms are all based on the notions of the *free space diagram* or the *free space surface*. Usually the algorithms first consider their decision variant: For a fixed  $\varepsilon > 0$  decide whether the distance is at most  $\varepsilon$  or not. Afterwards the actual minimization problem can then be solved by either applying parametric search (which adds a log-factor to the runtime), or binary search (which is more feasible to implement in practice).

For two curves  $f, g: [0, 1] \rightarrow \mathbb{R}^2$  the set  $F_\varepsilon(f, g) := \{(s, t) \in [0, 1]^2 \mid \|f(s) - g(t)\| \leq \varepsilon\}$  is the *free space* of  $f$  and  $g$ . Here  $\| \binom{x}{y} \| = \sqrt{x^2 + y^2}$  denotes the  $L_2$ -norm. The

partition of  $[0, 1]^2$  into regions belonging or not belonging to  $F_\varepsilon(f, g)$  is called the *free space diagram*. The free space of two line segments is an ellipse and the free space diagram of two polygonal curves of  $m$  and  $n$  segments is composed of  $mn$  segment-segment free space diagrams. Any curve in  $F_\varepsilon(f, g)$  from the lower left corner to the upper right corner in the free space as a continuous mapping from  $[0, 1]$  to  $[0, 1]^2$  directly gives continuous reparametrizations  $\alpha$  and  $\beta$ . Therefore  $\delta_F(f, g) \leq \varepsilon$  iff there exists such a curve, and  $\tilde{\delta}_F(f, g) \leq \varepsilon$  iff there exists such a curve which is monotone. See [2] for more details and illustrations on this topic.

The definition of the free space between two curves generalizes to the free space between an embedded graph and a curve as follows: The free space of the road network and the trajectory is the union of all free spaces of all straight-line edges of the road network with the vehicle trajectory. Notice that the free space of a vertex  $v$  with the vehicle trajectory is a one-dimensional free space, and the individual free spaces of all edges incident to  $v$  with the trajectory share the one-dimensional free space at  $v$ . Thus we can glue together the two-dimensional free space diagrams along the one-dimensional free space they have in common, according to the adjacency information in the road network. We call this topological structure, which is the union of all edge-trajectory free space diagrams, the *free space surface* of the road network and the trajectory. Figure 3 gives an example road network (left) and its corresponding free space surface and 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.



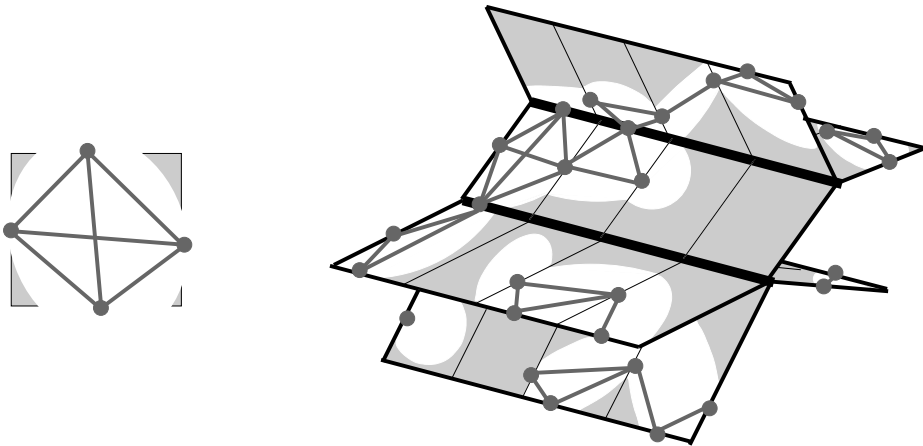
**Fig. 3.** A road network (left) and corresponding free space surface (right).

**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  $\Theta(mn)$  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 swepline 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 swepline. 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.

Both algorithms need  $\Theta(mn)$  space since they have to compute the free space diagram or surface in advance.

**Weak Fréchet Distance and Free Space Graph** 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. The traditional approach [2] constructs the free space diagram in  $\Theta(mn)$  time and space and then runs a graph traversal algorithm on the free space for a total of  $\Theta(mn)$  time. In this approach the free space is implicitly interpreted as a *free space graph* whose vertices are the white intervals on the boundary of a free space cell, and all intervals incident to the same cell are connected by edges. See Figure 4 (Left) for an illustration of the free space graph of one free space cell.



**Fig. 4.** The free space graph of one free space cell (left) and of the free space surface of Figure 3 (right).

This *free space graph* encodes all connectivity information of the free space. We can modify this graph representation in order to solve the optimization problem directly, without the decision problem: Every pair  $(e, v)$  with an edge  $e$  on one curve and a

vertex  $v$  on the other curve defines a possible free space interval, which in turn defines a vertex in the free space graph. For each such pair  $(e, v)$  we compute the smallest  $\epsilon$  for which the free space interval exists, which by definition is the smallest distance from  $v$  to  $e$ . We weigh the corresponding free space graph vertex with that distance. Let us define the *weight of a path* in the free space graph to be the maximum of the weights of the free space graph vertices it visits. Then a path from the start to the end with minimum weight  $\epsilon^*$  corresponds to a path in the free space diagram for  $\epsilon^*$ , and any  $\epsilon < \epsilon^*$  for which there is a path in the free space diagram would contradict the definition of  $\epsilon^*$ . Hence  $\epsilon^*$  is the optimal  $\epsilon$ , or in other words the weak Fréchet distance. We can compute such a *shortest path* using Dijkstra’s shortest-path algorithm in  $O(mn \log(mn))$  time and  $\Theta(mn)$  space.

In [3] we have shown that the depth-first strategy for the decision problem can be generalized to the global map-matching problem by applying depth-first search in a similar manner to the free space surface. Here, every (road network edge, trajectory point)-pair and every (trajectory edge, road network vertex)-pair define a free space graph vertex. Two free space graph vertices are connected by an edge iff the corresponding intervals are incident to the same cell in the free space surface. See Figure 4 for an illustration. We weigh each free space graph vertex just as described before for the curve-curve case and use Dijkstra’s algorithm to compute a shortest path in  $O(mn \log(mn))$  time. The space complexity is  $\Theta(mn)$  for the initial construction of the free space. The beauty of this algorithm is that it runs just like the algorithm for the computation of the weak Fréchet distance for two curves, only that the underlying free space graph is a bit more complicated.

**Output Sensitive Algorithm** All previous algorithms for the computation of the strong or the weak Fréchet distance [2] or for the strong or weak Fréchet distance based map-matching algorithms [1, 3] store the whole free space of size  $\Theta(mn)$ . Our shortest path algorithm on the free space graph as described in Section 3.2 however (for the curve-curve case and for the map-matching curve-graph case as well) can be implemented to explore and construct necessary portions of the free space graph on the fly during the traversal. We use hash tables to keep track of previously visited free space graph vertices. Assuming that a hash table operation needs  $O(1)$  time this yields an *output sensitive* algorithm that runs in  $O(K \log K)$  time, where  $K$  is the size of the *traversed free space*. The *traversed free space* is that part of the free space that has to be traversed in order to find a shortest path from any start free space graph vertex  $(e, p_0)$  to any end free space graph vertex  $(f, p_n)$ , where  $e, f$  are road network edges and  $p_1$  is the first and  $p_n$  the last vertex of the GPS curve. Clearly  $K = O(mn)$ , but since the algorithm stops when it finds the first end vertex  $K$  might be much smaller than  $O(mn)$ .

This output sensitive algorithm has the potential to be much more efficient especially for large road maps. In addition, there are several pruning techniques for Dijkstra’s algorithm on large graphs, see e.g. [13, 14, 7], which could be used to speed-up the algorithm even more in practice.

For our application we do not use any of these general and often heuristic techniques. Instead, we exploit special knowledge of our data in order to prune the graph in a provably correct manner yielding a fast algorithm. We describe this algorithm in Section 4.

## 4 Localizing Global Map-Matching Strategies

The additional knowledge about tracking data that we can exploit is a worst-case movement estimate with respect to reached distances in-between position samples. The following sections (i) briefly illustrate of how to compute the worst-case movement estimate and (ii) how to exploit it in the algorithmic map-matching design.

### 4.1 Worst-case Movement Estimate

The tracking data is obtained by sampling the positions using typically 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 [9].

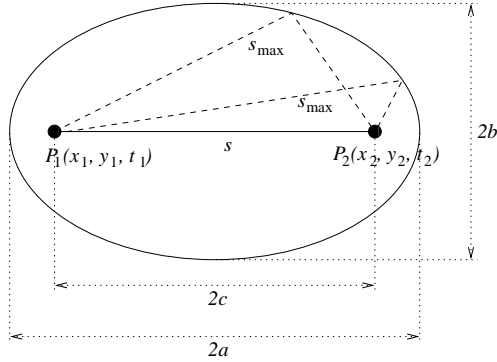
Although the GPS error can be substantial in certain situations (shadowed and reflected signals), with the use of GPS signal augmentation (WAAS, EGNOS) the typical error is in the range of  $8m$  to  $2m$ . The sampling error, which in the case of vehicle tracking data is by two magnitudes larger than the measurement error, has consequently a larger impact on the algorithmic design for map-matching. This error is directly related to the *sampling rate*, the frequency with which position samples are taken. The sampling error can be constrained using its worst-case estimate. Such a worst-case estimate are the points a vehicle can reach during the sampling, assuming that it travels with *maximum speed*.

To illustrate the impact of the sampling rate on the imprecision of the interpolated trajectory data, consider sampling the position of a vehicle every  $30s$ , which is a typical sampling rate used in vehicle tracking applications. At a speed of  $50km/h$ , the traveled distance between position samples is as much as  $417m$ !

The authors in [9] show that the sampling error between two positions,  $P_1$  and  $P_2$  in the time interval  $[t_1, t_2]$  and a given *maximum speed*  $v$ , 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. Fig. 5). 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  $b^2 = a^2 - c^2 = 25/36^2 \cdot v^2 r^2 - c^2$ . 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.

### 4.2 Adaptive Clipping

Let us call any two consecutive positions in the vehicle trajectory an *edge* of the vehicle trajectory. For any such edge  $e$  the sampling error and the measurement error describe a region in the plane which consists of all positions at which the vehicle could have been while the endpoints of  $e$  were recorded using the GPS receiver. As described in Section 4.1 the sampling error describes an error ellipse around  $e$  while



**Fig. 5.** Error ellipse.

the measurement error usually describes a circular region around a trajectory position. For a given velocity  $v$ , sampling rate  $r$ , and measurement error  $\mu$  we define the *active region*  $A(e)$  to be the Minkowski sum of the error ellipse (defined by  $e$ ,  $v$ , and  $r$ ) with a disk of radius  $\mu$  (centered at the origin).<sup>4</sup> Intuitively  $A(e)$  is a “thickened” ellipse whose boundary has been thickened by a disk of radius  $\mu$ . The active region  $A(e)$  is a combination of measurement and sampling errors and contains all positions in the plane which could correspond to any point on  $e$ . In the implementation we approximate  $A(e)$  by its axis-aligned bounding box.

Let us call the union of all active regions of all vehicle trajectory edges the *active region* of the trajectory. One speed-up possibility would be to compute the active region of the trajectory and preprocess the road network by clipping it to the part that lies inside the active region. This is however somewhat impractical because it requires querying the whole road network database in advance. Also in this setting the global map-matching algorithms are allowed to match a position in the roadmap that lies in  $A(f)$  to a point on  $e$ , where  $e$  and  $f$  are two different trajectory edges.

Our new *Adaptive Clipping* algorithm follows an incremental clipping approach: Our output sensitive weak Fréchet algorithm from Section 3.2 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  $p_1, \dots, p_n$ , and denote by  $\overline{p_i p_{i+1}}$  the trajectory edge between  $p_i$  and  $p_{i+1}$ .

**Stage 1.**

We seed Dijkstra’s algorithm with the start free space graph vertices  $(p_1, e)$ , where  $e$  is any road network edge in  $A(\overline{p_1 p_2})$ . We then execute Dijkstra’s algorithm on the part of the free space graph induced by the free space graph vertices  $(p_1, e)$ ,  $(v, \overline{p_1 p_2})$ ,  $(p_2, e')$ , where  $v$  is any vertex and  $e$  any edge in  $A(\overline{p_1 p_2})$ , and  $e'$  is any edge in

<sup>4</sup> The Minkowski sum (also called vector sum) of two point sets  $C, D \in \mathbb{R}^2$  is defined as  $C \oplus D = \{c + d \mid c \in C, d \in D\}$ .

$A(\overline{p_1 p_2}) \cap A(\overline{p_2 p_3})$ . This computes shortest paths from a start vertex to any vertex  $(p_2, e')$ .

**Stage  $i$  for  $2 \leq i < n$ .**

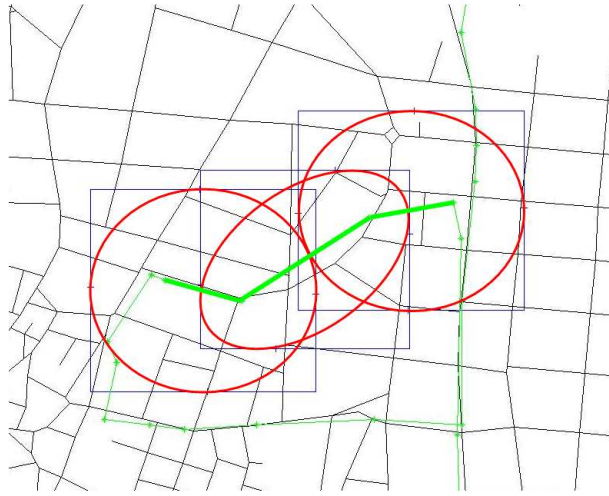
Stage  $(i - 1)$  has computed shortest path values for all free space graph vertices  $(p_{i-1}, e')$ . We use those as the seed for Dijkstra's algorithm (including their associated shortest path values), and then execute Dijkstra's algorithm on the part of the free space graph induced by the free space graph vertices  $(p_{i-1}, e)$ ,  $(v, \overline{p_{i-1} p_i})$ ,  $(p_i, e')$ , where  $v$  is any vertex and  $e$  any edge in  $A(\overline{p_{i-1} p_i})$ , and  $e'$  is any edge in  $A(\overline{p_{i-1} p_i}) \cap A(\overline{p_i p_{i+1}})$ . This computes shortest paths to any vertex  $(p_i, e')$ .

**Stage  $n$ .**

Same as stage  $i$  for  $i \geq 2$ , just that the end free space graph vertices are all  $(p_n, e)$ , where  $e$  is any graph edge in  $A(\overline{p_{n-1} p_n})$ .

**Traceback.**

All stages store predecessor trees to allow the construction of shortest paths. We only construct one short path in the very end, which traces back through all stages.



**Fig. 6.** An excerpt of the road network, the trajectory, and three active regions with their bounding boxes.

See Figure 6 for an illustration of the active regions in the road network. This algorithm computes a shortest path in a restricted free space graph. We claim that this restricted free space graph allows us to find a curve in the roadmap that lies in the active region of the trajectory and that has minimum weak Fréchet distance to the trajectory. In fact, we claim the even stronger property that the optimal reparametrization matches points on a trajectory edge  $e$  only to road network positions in  $A(e)$  (and not in  $A(f) \setminus A(e)$  for any other edge  $f$ ).

Indeed, the shortest paths computed in stage 1 encode reparametrizations between the trajectory and curves in the roadmap that start in  $(p_1, e)$ , lie in  $A(\overline{p_1 p_2})$ , and end in  $A(\overline{p_1 p_2}) \cap A(\overline{p_2 p_3})$ . Notice that any point in the road network assigned to  $p_2$  will have to lie in this intersection of active regions. The  $i$ -th stage computes reparametrizations between the trajectory and curves in the roadmap that start in  $A(\overline{p_{i-2} p_{i-1}}) \cap A(\overline{p_{i-1} p_i})$ , lie in  $A(\overline{p_i p_{i+1}})$ , and end in  $A(\overline{p_{i-1} p_i}) \cap A(\overline{p_i p_{i+1}})$ . An inductive argument shows that, since the algorithm uses the results from the previous stage as the seed for the current stage, it extends the shortest paths of the previous stage to shortest paths in the current stage which encode only those reparametrizations that match points on a trajectory edge  $e$  to a road network position in  $A(e)$  and a trajectory vertex  $p_i$  only to a road network position in  $A(\overline{p_{i-1} p_i}) \cap A(\overline{p_i p_{i+1}})$ .

The running time of this algorithm is  $O(\sum_{i=1}^{n-1} M_i \log M_i + n)$ , where  $M_i$  is the number of edges and vertices of the road network in the active region  $A(\overline{p_i p_{i+1}})$ . In general  $M_i \in O(M)$  for each  $i$ , however this is a very rough upper bound. In our application the data suggests the following two special cases: (1) It seems that most of the times active regions  $A(\overline{p\bar{q}})$  and  $A(\overline{p'q'})$  only intersect if they are adjacent, i.e., if  $q = p'$ . In this case  $O(\sum_{i=1}^{n-1} M_i) = m$  and the total runtime simplifies to  $O(m \log m)$ . (2) In our application it also seems that the parts of the road network in each error ellipse are very small, almost of constant complexity. If each  $M_i$  is a constant then the runtime of course simplifies to  $O(n \log n)$ .

## 5 Experimental Evaluation

The objective of this section is to contrast the performance of Adaptive Clipping with the Incremental algorithm in terms of (i) the quality of the map-matching result, (ii) running time, and (iii) database IO operations.

### 5.1 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 15727 position samples. The smallest and the largest trajectory consist of 207 and 1003 edges, respectively<sup>5</sup>. The tracking data was collected in the municipal area of Athens (40x40km) through the years 2000 to 2003. The road network consists of 108000 vertices and 150000 edges.

An Oracle DBMS version 9i Spatial was used to store the road network graph. The DBMS was installed on a Dell Precision Workstation, Pentium 4, 3.6 GHz Intel EM64T with 2GB RAM and SATA hard drives. The database block size was set to 6KB and the database cache size to 192MB.

The graph was stored by means of tiles. Thus, only portions of the road network are kept in main memory. Should the map-matching algorithm require an additional network portion, the respective tile is fetched from the database. In our experiments we partitioned the road network into  $16 \times 16 = 256$  tiles and used a LRU main-memory buffer scheme of size 50 tiles. The tiling parameter was established during a brief

<sup>5</sup> The vehicle tracking data was supplied by Emphasis Telematics, a co-operating telematics company and fleet management service provider.

empirical study. However, further research into identifying the optimal parameter setting for each algorithm is necessary.

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 9(a)).

## 5.2 Accuracy

To compare the two algorithms, map-matching results for the tracking data were evaluated using the (i) weak Fréchet distance, (ii) the strong Fréchet distance (c.f. Section 3.2) and (iii) the *average Fréchet distance* with sampling distance  $2m$  (cf. [3]). The average Fréchet distance computes an average of the distance between matched points (in contrast to the weak or strong Fréchet distances which compute the maximum).

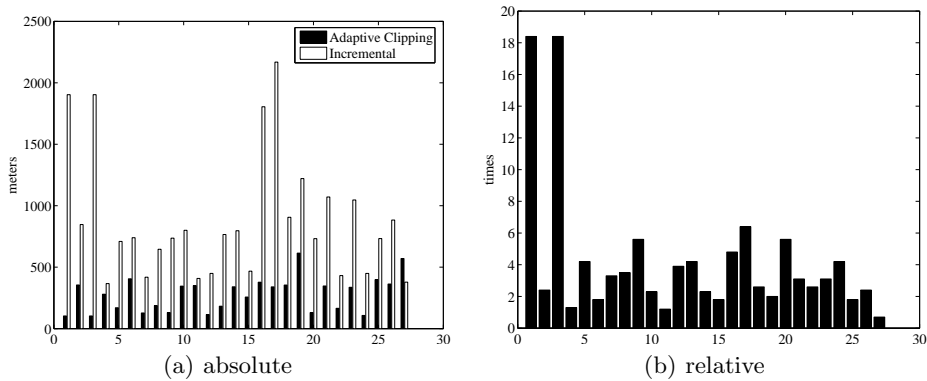
All 27 data sets contain noise in the sampling rate (i.e., two consecutive samples are significantly more than 30 seconds apart), and the road network misses several roads (such that no corresponding path can be found in certain areas). We cope with this noise by chopping noise regions off the data sets (online, during the execution of the map-matching algorithm), which results in each data set now being a set of sub-trajectories. The smallest number of sub-trajectories was 4, the largest 23, and the average 15. Notice that we still match all the trajectory positions, we just do not consider certain trajectory edges to be part of the data set. We generalize the weak and strong Fréchet distances to this setting by taking the maximum of the distances for each sub-trajectory. The average Fréchet distance takes the average of all individual distances.

The Adaptive Clipping algorithm computes along with the result curve the weak Fréchet distance based on the *restricted* free space graph. This distance is, since it considers only a subset of reparametrizations, greater or equal to the weak Fréchet distance between the trajectory and the result curve in the road network. Interestingly, out of the 399 sub-trajectories only 3 have a smaller weak Fréchet distance. Also, in only one case the Fréchet distance is greater than the weak Fréchet distance. Hence, the tracking data exhibits special properties in which all three distances happen to almost always coincide. For this reason, we consider in the following experiments only the weak Fréchet distance and the average Fréchet distance to determine the quality of the respective matching results.

Figures 7 and 8 determine the quality of the matching result by means of the weak Fréchet and the average Fréchet distance, respectively. Figures 7(a) and 8(a) give the absolute distances, whereas the relative quality is shown in Figures 7(b) and 8(b). For the relative distance, the Adaptive Clipping algorithm is the benchmark, e.g., -20% indicates that the measured distance in the result of the Incremental algorithm was 20% smaller than that of Adaptive Clipping algorithm.

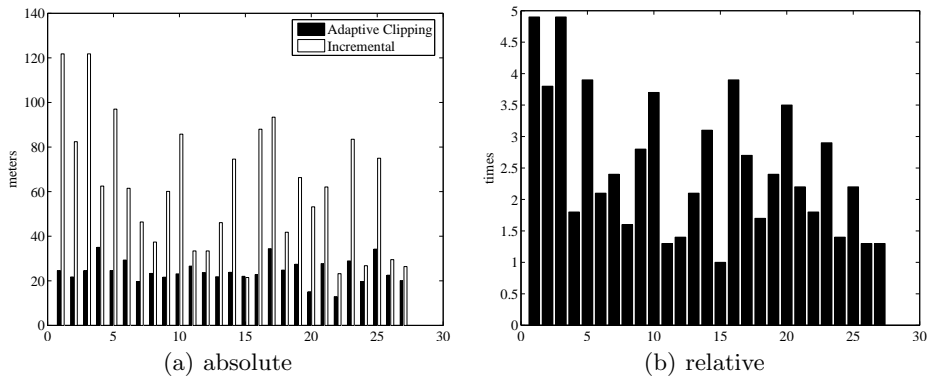
What can be readily observed is that the Adaptive Clipping algorithm produces matching results of superior quality. The measured weak Fréchet distance for the Incremental algorithm is an average of 4 times larger (worst case 18 times). Using the average Fréchet distance, it is still on average 1.5 times larger (worst case 5 times).

The absolute distances in Figures 7(a) and 8(a) indicate that the trajectory and the matched curve exhibit (i) a worst case distance (weak Fréchet) of up to  $2000m$



**Fig. 7.** Weak Fréchet distance quality measure

and 600m and (ii) an average distance (average Fréchet) of up to 120m and 35m for the Incremental and the Adaptive Clipping algorithm, respectively.



**Fig. 8.** Average Fréchet distance quality measure

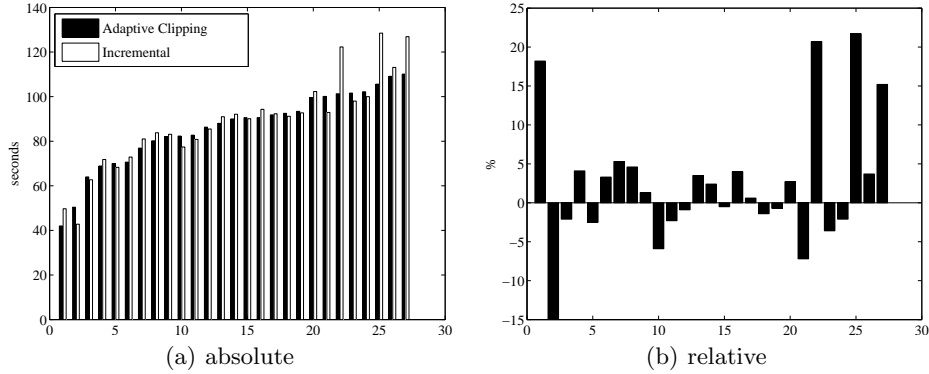
### 5.3 Speed

The speed of computation is measured in terms of (i) the algorithmic running time and (ii) the number of database IO operations.

Figures 9 and 10 give the overall running times and time attributed to database access, respectively. Again, absolute and relative times (percentages) are shown, with the Adaptive Clipping algorithm serving as a benchmark for the Incremental algorithm.

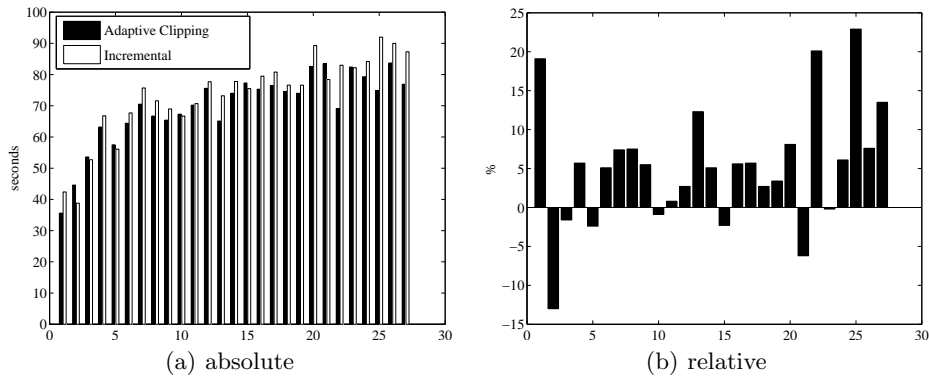
An important objective for this work was to use it for real-time map-matching of tracking data. Figure 9(a) shows that matching the smallest trajectory, which consists of 207 position samples takes 40s (50s). With a sampling rate of 30s, this

data was collected over a period of 6210s (1h 40min). Thus, the sampling rate could be as low as 0.2s (0.25s) (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.



**Fig. 9.** Total running times of the map-matching algorithms

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 -  $O(n)$ , 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.



**Fig. 10.** Running times attributed to database access

A large portion of the running time (75% to 85%) is attributed to database access. Hence, this time largely determines the overall running time. However, interesting deviations (cf. trajectories 10 to 15) exist. When Adaptive Clipping is slower or equal in terms of database access time, the overall relative running time of Adaptive Clipping is even worse, i.e., the algorithm draws its performance advantage largely from a more intelligent choice of which portion of the road network to load from the database.

The number of logical IO operations is shown in Figure 11. Logical IO is the sum of the number of buffers read from disk and from the memory cache. Using a buffer cache size of 192MB the number of disk accesses was kept low. To force disk accesses, we conducted an experiment with a smaller cache size (1MB). The charts were similar to what is presented in 11 and, most importantly, did not present different performance characteristics relative between the trajectories and in relation the running times of Figure 10.

The number of IO operations show a more balanced picture between the two algorithms but still confirm the performance advantage of Adaptive Clipping. Comparing Figures 11(b) and 10(b) show that the running time attributed to database access is not solely determined by the number of IO operations but also by additional factors related to query execution.

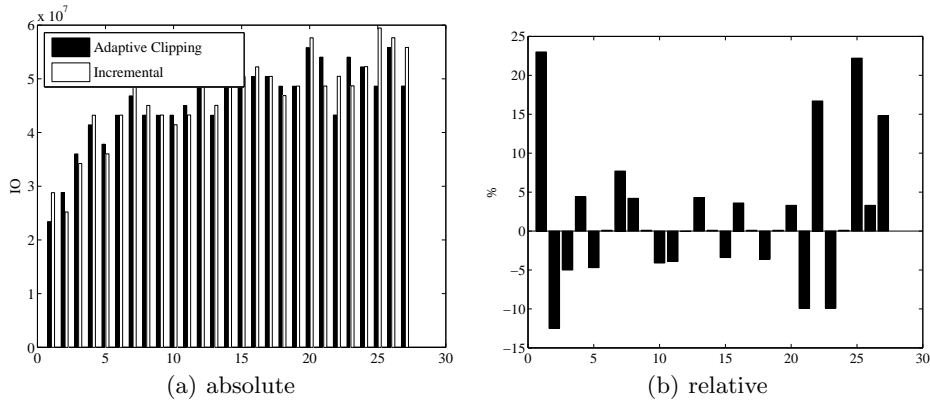


Fig. 11. Number of database IO operations

#### 5.4 Summary

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.

## 6 Conclusions and Future Work

Map-matching algorithms are an enabling technology to the use of vehicle tracking data for traffic assessment and related applications such as routing. To possibly include a large variety of datasets and to guarantee the timeliness of the data, the key objective in this work is to provide a *fast* and *accurate map-matching algorithm*.

We present the *Adaptive Clipping* algorithm, which combines the best of both worlds of previously introduced algorithms. It is *fast* in that it intelligently uses tracking data metadata, worst-case error estimates, to limit the portions of the road network used in its matching process (output sensitive algorithm). It is *accurate* since it uses a curve-matching approach and here the Fréchet distance to find a path in the road network that closely (in terms of distance measure) resembles the actual trajectory.

The performance study establishes the running time of Adaptive Clipping to be lower than that of the fastest available algorithm, the Incremental algorithm. Given its asymptotic bound of  $O(n \log n)$ , it is almost linear in terms of the number of trajectory edges. Also, the quality of the matching result is in all cases superior to that of the Incremental algorithm. Overall, Adaptive Clipping is a fast algorithm producing high-quality matching results and thus can be used in a real-time tracking data collection scenario.

The directions for future work are as follows. Having reached a certain level of maturity, Adaptive Clipping should be applied as a map-matching algorithm in a life data collection scenario. Also, given that in such a scenario typically many concurrent data streams exist, we have to evaluate of how well the overall architecture (DBMS and algorithm) scales with concurrent map-matching threads. With respect to algorithmic improvements, we will explore and evaluate other approaches that belong to the category of output sensitive algorithms (cf. Section 3.2).

## Acknowledgments

This research is supported in part by the the IXNILATHS project funded by the Greek General Secretariat of Research and Technology, and the Faculty Research Award Program at the University of Texas at San Antonio.

## 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. Civilis, C. S. Jensen, J. Nenortaite, and S. Pakalnis. Efficient tracking of moving objects with precision guarantees. In *Proc MobiQuitous conf.*, pages 164–173, 2004.
5. A. Civilis, C. S. Jensen, and S. Pakalnis. Techniques for efficient road-network-based tracking of moving objects. *IEEE Transactions on Knowledge and Data Engineering*, 17(5):698–711, 2005.

6. M. Fréchet. Sur quelques points du calcul fonctionnel. *Rendiconti del circolo Matematico di Palermo*, 22:1–74, 1906.
7. A. Goldberg and C. Harrelson. Computing the shortest path:  $a^*$  search meets graph theory. In *Proc. of the Symposium of Discrete Algorithms (SODA)*, pages 156–165, 2005.
8. J. Greenfeld. Matching GPS observations to locations on a digital map. In *Proc. 81th Annual Meeting of the Transportation Research Board*, Washington, DC, 2002.
9. D. Pfoser and C. S. Jensen. Capturing the uncertainty of moving-object representations. In *Proc. 6th SSD conf.*, pages 111–132, 1999.
10. D. Pfoser, N. Tryfona, and A. Voisard. Dynamic travel time maps: An accurate database for routing algorithms. Technical report, RA Computer Technology Institute Technical Report No. 2005/03/02, Patras, Greece, 2005.
11. M. Quddus, W. Ochieng, L. Zhao, and R. Noland. A general map matching algorithm for transport telematics applications. *GPS Solutions Journal*, 7(3):157–167, 2003.
12. R.-P. Schaefer, K.-U. Thiessenhusen, and P. Wagner. A Traffic Information System by Means of Real-time Floating-car Data. In *Proc. ITS World Congress*, Chicago USA, 2002.
13. F. Schulz, D. Wagner, and K. Weihe. Dijkstra’s algorithm on-line: An empirical case study from public railroad transport. *Journal of Experimental Algorithmics*, 5(12), 2000.
14. D. Wagner and T. Willhalm. Geometric speed-up techniques for finding shortest paths in large sparse graphs. In *Proceedings of the 11th Annual European Symposium on Algorithms (ESA ’03)*, volume 2832 of Lecture Notes in Computer Science, pages 776–787. Springer, 2003.
15. H. Yin and O. Wolfson. A weight-based map matching method in moving objects databases. In *Proc. 16th SSDBM conf.*, pages 437–438, 2004.