Algorithms for Matching and Predicting Trajectories

Jochen Eisner* Stefan Funke* Andre Herbst† Andreas Spillner† Sabine Storandt*

Abstract

We consider the following two problems: *Map Matching:* Given a sequence of (imprecise) location measurements from a mobile user moving on a road network, determine the most likely path in the network this user has travelled along. *Prediction of Trajectories:* Given the path of where a mobile user has moved along in a road network up to now, predict where he will travel along in the near future.

Our map matching algorithm is simple and efficient even in case of very imprecise measurements like GSM-localizations and allows for the real-time tracking of a large number of mobile users on modest hardware. Our proposed path prediction algorithm is equally simple but yields extremely accurate predictions at a very low computational cost.

1 Introduction

Nowadays, almost every cell phone is equipped with GPS or at least allows for localization derived from nearby cell phone base stations. This gives rise to a plethora of exciting applications as well as research questions.

From an application point of view, often we are more interested in higher level location information like *Where has the mobile user moved along in the road network?* rather than a pair of coordinates specifying its position in longitude and latitude. The generation of such higher level location information becomes even more challenging when the location measurements are imprecise. GPS (in the non-military context) typically incurs imprecisions in the range of several meters. If GPS is not available, for example due to obstructions by tall buildings in an urban environment or foliage, localization derived from nearby cell phone base stations allows for location measurements with an uncertainty of several hundred meters to few kilometers.

In the first part of this paper we consider the *Map Matching Problem*, that is, given a sequence of (imprecise) location measurements of a mobile user and an underlying road network, compute a reasonable route where the user has traveled along – in other words we aim at matching a discretized and fuzzy trajectory to a route in a road network. We are particularly concerned with the scenario where the location measurements are very imprecise (imprecision up to few kilometers), since there are still numerous situations where more precise location information is not available or can only be acquired at high cost.

In the second part of the paper we aim at even higher level position information: given the path a mobile user has moved along in a road network up to the current moment, predict where the user will move along in the near future. We call this the *Path Prediction Problem*. In contrast to known solutions to this problem we will compute our prediction not only based on the geometry of the known path (using extrapolation) or directional information implied by the underlying road network but make explicit use of the structure of the space of shortest paths in the network.

If we do not make any assumption about a 'typical' behavior of a mobile user, there is little hope for effective map matching or path prediction schemes. If, for example, the mobile user at every crossing chooses an *arbitrary* turning, a good guess of the actual route the user took based on measurements with several kilometers of imprecision seems difficult. It appears even more challenging to make an educated guess about where the user will be in 30 minutes if nothing is known about his behavior. So all our proposed algorithms will be based on the assumption that users travel on (at least piecewise) shortest/quickest paths. This seems like a natural assumption to make since most people tend to travel with the aim of getting to a certain destination as quickly as possible. We have little hope to be able to track or predict people cruising around unless some additional information is known.

Related Work In many applications, map matching is an *online* problem, that is, location measurements have to be aligned with the road map as soon as they are taken (see e.g. [11] for more on this version of the problem). In contrast, here we focus on *offline* map

*Universität Stuttgart, Institut für Formale Methoden der Informatik, 70569 Stuttgart, Germany; jochen.eisner, stefan.funke,sabine.storandt@fmi.uni-stuttgart.de

†Universität Greifswald, Institut für Mathematik und Informatik, 17487 Greifswald, Germany; andre.herbst,andreas.spillner@uni-greifswald.de

matching, that is, from a sequence of location measurements taken along a route we want to reconstruct this route. While, at least in principle, any algorithm for online map matching can also be used for offline map matching, the low accuracy of the reconstructed route can be a problem (see e.g. [10]). Existing approaches for solving the offline problem directly involve a scoring of routes in the road network reflecting how well the sequence of measurements fits on a route. The scoring can be based on e.g. the Fréchet distance (or variants of it) [1, 4] or a weighting of road segments according to their relative position with respect to the location measurements [14]. To reconstruct a route often a heuristic search for a route with optimal score is performed (but see also [13] where an optimal route is found exactly by integer programming). Recently, in [9] a novel scheme for offline map matching is presented especially designed for situations where the density of measurements along the route is small and the level of imprecision in the measurements is up to 100 meters. We will go into more detail about this approach in Section 3.

The related work with respect to path prediction that we are aware of can be found in the context of dead-reckoning protocols. Here a mobile user is to send periodically its position to a server. When the required bandwidth for transmitting this position information is an issue, predicting the motion of a user both on the client/user side as well as on the server side can result in considerable savings by only transmitting deviations from the prediction. Different kinds of prediction schemes have been examined in this context. *Linear prediction* assumes that the mobile user keeps on moving along a line given by the reported position and direction. This already yields quite good results. The more advanced *higher-order prediction* uses higher-order functions (curves or splines) which, for example, could capture the object's movements in a curved road segment. Different from these strategies which essentially work in the open space, *map-based prediction strategies* make use of the fact that mobile users are often moving along a road network, as a person walking along the streets of a city or a car travelling on a freeway. A map-based prediction strategy tries to match the user's position to a road of an underlying map and assumes that he keeps on moving along this street. At an intersection the strategy tries to choose a direction the mobile user is most likely to follow. [8] gives a detailed survey and summary of the main advantages and disadvantages of these approaches. The map-based prediction strategy as up to now most efficient prediction strategy will be used as a baseline against we measure our new prediction algorithms in Section 4.

Our Contribution We provide simple and sound solutions for both the map matching as well as the path prediction problem. For the map matching problem our proposed algorithm yields running times that are much faster than previous approaches; only by this speed-up these matching techniques become applicable for dealing with large imprecisions and for tracking large numbers of mobile users. For a single mobile user a new measurement can be processed in few milliseconds, so if measurements are taken every few minutes, several hundreds of thousands of mobile users can be tracked on a single multi-core server. By making use of the structure and the use patterns of the road networks, our path prediction algorithm allows for faithful prediction of trajectories up to several hundred kilometers outperforming known prediction strategies in terms of quality. The computational effort is so little that a large scale prediction of mobile users is also possible.

2 Preliminaries

2.1 Formal Problem Definitions A *road network* is an edge-weighted directed graph $G = (V, E, \ell)$ whose vertices are mapped to pairs of coordinates (usually longitude and latitude). The *length* $\ell(e)$ of an edge e in a road network is often simply the distance between the end points of e but can also represent other 'costs' such as travel time. A *location measurement* is a triple (x, y, r) consisting of longitude and latitude coordinates x and y, respectively, and a radius r. That is, we model a measurement by a circle and the degree of uncertainty about the true location at the time of measurement, also referred to as *imprecision*, is captured by the radius of this circle.

The input to the *map matching* problem consists of a road network $G = (V, E, \ell)$ and a sequence $M = m_1, m_2, \ldots, m_s$ of location measurements that have been taken while traveling along a *path* $\pi = v_0, v_1, \ldots, v_k$ in G, that is, a sequence of vertices such that there is an edge directed from v_{i-1} to v_i in G for all $i \in \{1, \ldots, k\}$. From M we want to reconstruct π, that is, find a path $\pi' = u_0, u_1, \ldots, u_{k'}$ in G that is close to π, see Figure 1 for a simple problem instance.

The input to the problem of *predicting trajectories* is a road network $G = (V, E, \ell)$ and a prefix $\pi = v_0, v_1, \ldots, v_i$ in G of a path $\pi^* = v_0, v_1, v_2, \ldots, v_l$ in G. The goal is to predict π^* if only π is known. That is, we want to compute a path $\pi' = v_0, v_1, \ldots, v_i, v_{i+1}, \ldots, v_k$ in G, see Figure 2.

2.2 Quality Metrics For both problems we want to quantify how close the reconstructed/predicted path π' is to the 'true' path π, π^* respectively. We adopt the two accuracy metrics also used in [9]: the ratio of the

total number $A_N(\pi, \pi')$ and the total length $A_L(\pi, \pi')$ of edges of π that have *not* been recovered correctly in π' to the number of edges and the length, respectively, of π.

2.3 Performance Metrics Our proposed algorithms are all based on variants of Dijkstra-like graph traversals. As with ordinary Dijkstra, the running time is essentially determined by the number of Dijkstra operations (polls from the priority queue). Furthermore, the underlying graph representation, the implementation language and the hardware on which the algorithms are run crucially influence the experienced running times. For example, a reasonable C++ implementation on a standard Desktop PC allows for about 1 Million Dijkstra operations per second. For easier platform- and implementation-independent comparison we mostly state the running times of our algorithms in terms of Dijkstra operations.

2.4 Contraction Hierarchies (CH) One important ingredient to ensure the efficiency of our map-matching algorithm is a general technique to speed-up shortest path computations. There are many speed-up schemes like reach [7], highway hierarchies [12], or transit nodes [2]. We chose in our implementation *contraction hierarchies* [5] due to their simplicity and efficiency. Such a contraction hierarchy augments the given road network $G = (V, E, \ell)$ to a road network $G' = (V, E', \ell')$ by adding new directed edges to G based on an ordering $\omega = v_1, v_2, \ldots, v_n$ of the vertices of G. Essentially the vertices are contracted in this order and edges are added such that shortest-path distances in the remaining network are preserved.

Shortest path queries in G can then be answered more efficiently based on the fact that in G' there exists a shortest path for any pair of vertices (s, t) such that the indices of the vertices on this shortest path are monotonously increasing first and decreasing afterwards with respect to the ordering ω. Therefore, to identify a shortest path from s to t, a bidirectional Dijkstra algorithm can be adopted. More precisely, one can use one Dijkstra run (forward) starting in s and a second Dijkstra run (backward) starting in t. The forward run considers only outgoing edges which lead to a vertex with a larger index while the backward run considers only incoming edges which come from a vertex with larger index. The two runs have to meet in at least one vertex of the shortest path, thus computing a shortest path from s to t with a drastically reduced search space. Thus, one-to-one shortest path queries can be answered at a fraction (about 1/1000th) of the time of a full Dijkstra with the help of the augmented $G' = (V, E', \ell')$.

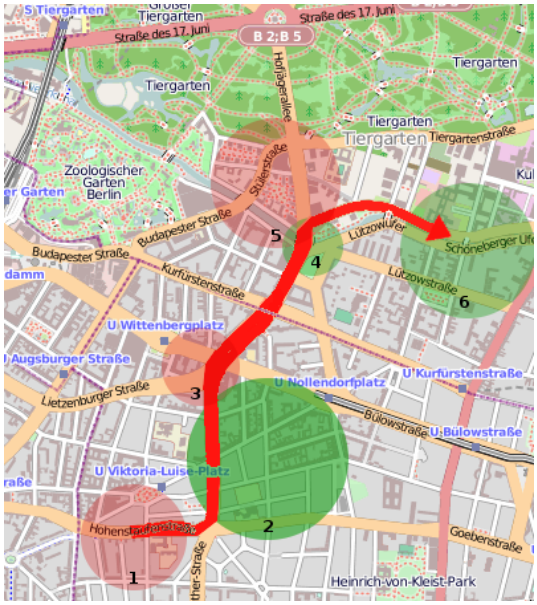


Figure 1: The Map Matching Problem: We are given a sequence of (here: 6) imprecise location measurements and want to reconstruct the original (red) path

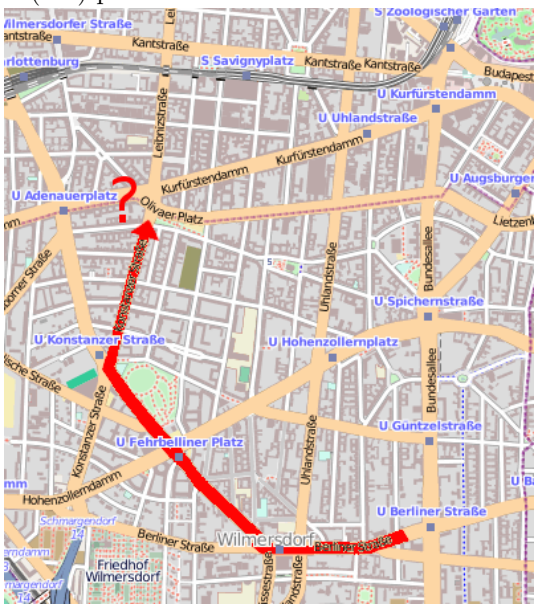


Figure 2: The Path Prediction Problem: We are given the route a mobile user has travelled up to now and want to predict its route in the near future.

In our approach to the map matching problem, however, one-to-many queries play an important role (cf. Section 3.2). The major drawback of using CHs directly is that for every target node a new Dijkstra has to be started. Therefore, the total query time might be even worse than just a simple Dijkstra in the original graph which stops after all target nodes have been settled. As a remedy, a technique suggested by Batz et al. in [3] in the context of time-dependent CHs, can be used. For time-dependent edge costs, a backward Dijkstra seems difficult, as the exact arrival time is unknown. So Batz et al. replace the backward Dijkstra run by a breadth first search (BFS), marking at each step all incoming edges that come from a vertex with larger index. Then the forward Dijkstra run can be used with the slight modification, that an edge is considered if it leads to a vertex with larger index or is marked. So, once edges have been marked by performing a BFS on the set of target vertices, shortest paths from the source to all target vertices can be found using one forward Dijkstra run until all targets are settled. This creates a fair compromise between thinning out the search space and still being able to settle all target nodes at once.

2.5 Reach-type Edge Classification Some of our path prediction strategies make use of the concept of *edge reach*, which will be explained briefly in the following.

The observation that when travelling long distance, one typically uses small roads only in the vicinity of the start and the destination of the trip and 'important' roads in between, has long been used to heuristically speed-up the Dijkstra search by not considering 'small' roads in the search when being far from both start and destination. Relying on the given road classification (freeway, highway, local road, track, ...) in the search for a shortest path does not guarantee optimality of the computed path, though. Gutman in [7] was the first to formalize this idea (he did this on vertices instead of edges, but the idea remains the same). One defines an edge (or a vertex) to be important ("of high reach") if it lies in the middle of a long shortest path, or more formally, for an edge $e = (u, v)$:

$$\text{reach}((u,v)) = \max_{(u,v)\in\pi=s\rightsquigarrow t} \min\left(d(s,v), d(u,t)\right)$$

that is, maximized over all shortest paths π which contain (u, v) we essentially take the minimum distance of this edge to either end of π as reach. It is not clear whether this reach value can be computed efficiently (without considering n^2 shortest paths), but *upper* bounds can computed very quickly.

The computation of Dijkstra can be accelerated using this idea as follows: when Dijkstra considers an outgoing edge $e = (u, v)$ and some lower bound $\phi(u)$ on the distance from u to the final target is known, only edges with reach $\geq \min(d(u), \phi(u))$ need to be considered. This idea, together with further ingredients like edge contractions has led to query times which are about a factor of 1000 faster than ordinary Dijkstra. Goldberg et al. [6] and Sanders/Schulte [12] are incarnations for this idea, with the latter being interesting as it uses a different metric (than the one defined by the edge costs) to determine what 'in the middle' means.

We will make use of the edge reach concept in our path prediction strategies.

3 The Map Matching Problem

As above, let $m_1, \ldots, m_s$ be an (ordered) sequence of location measurements. It will be convenient to view each m_i as a disk D_i with center c_i and radius r_i and it is reasonable to assume that the true location where measurement m_i was taken lies inside D_i (an example of a sequence of such disks is depicted in Figure 3). Thus, without any additional assumptions all points/vertices of the road network inside D_i are *candidate points*, that is, could be the location where measurement m_i was taken.

Figure 3: Shortest path (red) together with a sequence of disks each representing an imprecise location measurement taken along this path.

3.1 Previous approaches used for benchmarking One of the conceptually simplest approaches to map matching, also known as *point-to-point matching* [11], selects for each measurement m_i a vertex (or point on an edge) in the road network $G = (V, E)$ that has minimum distance to the point c_i and then con-

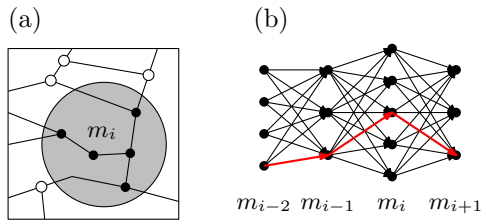


Figure 4: (a) The disk associated to a location measurement m_i. The vertices drawn as black dots form candidate points associated to m_i. (b) Part of the layered graph formed by the candidate points. The red directed edges correspond to part of the reconstructed route.

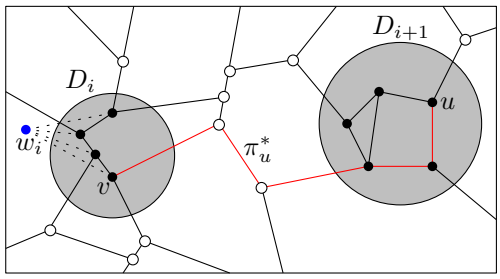


Figure 5: For each vertex $u \in V_{i+1}$ we compute a shortest path from w_i to u. This yields the path π_u^* (red).

nects consecutive vertices/points by shortest paths in the road network. While it is not hard to find examples where this approach will reconstruct the route only very poorly, it is very fast and we will use it as a benchmark for the run time of our new method.

To also provide a benchmark for the run time of more sophisticated recent approaches we re-implemented the one presented in [9] where the reconstruction of the route is based on a layered graph. In this graph each layer corresponds to the candidate points associated to a particular measurement. Directed edges connect candidate points in consecutive layers. The length of such an edge results from the length of a shortest path between the candidate points in the road network. Concatenating such shortest paths, selected according to some scoring function, yields the reconstructed route (cf. Figure 4).

As reported in [9], for this approach to be practically feasible the restriction to a small number of candidate points within each disk D_i is crucial, since otherwise the run time increases dramatically. To discard vertices and edges inside D_i as candidates, assumptions about the distribution of the errors involved in each measurement and on the time stamps attached to it are used. In the following we present a simple observation that allows to avoid the discarding of vertices and edges inside D_i and the explicit construction of the layered graph altogether while, at the same time, speeding up the computation by exploiting directly the fact that travelers tend to use shortest paths.

3.2 Our new approach Let V_j, $1 \leq j \leq s$, denote the vertices contained in D_j. Introduce a new vertex w_1 and directed edges, each of length 0, from w_1 to each vertex in V_1. Assume we have already constructed shortest paths π_v from w_1 to each vertex $v \in V_i$ for some $1 \leq i < s$ and each of these shortest paths visits the disks $D_1, D_2, \ldots, D_{i-1}$ in that order. Note that for $i = 1$ such paths are available immediately by construction.

Now, to compute shortest paths from w_1 to each vertex in V_{i+1} that visit the disks $D_1, D_2, \ldots, D_i$ in that order, it is not hard to see that this amounts to extending the shortest paths we already have from w_1 to the vertices in V_i. More specifically (cf. Figure 5), if $i \geq 2$ we introduce a new vertex w_i and directed edges from w_i to each vertex in V_i. In addition, the length of the edge from w_i to v is the length of π_v for every $v \in V_i$. Then we compute shortest paths from w_i to each vertex $u \in V_{i+1}$, denoting by π_u^* this path with vertex w_i removed. For each $u \in V_{i+1}$ consider the first node $v \in V_i$ on the shortest path π_u^*. Then we obtain a suitable shortest path π_u from w_1 to u that visits the disks $D_1, D_2, \ldots, D_i$ in that order by concatenating the paths π_v and π_u^*. So, as mentioned in Section 2.4, the reconstruction of the path along which the location measurements were taken can be reduced to a sequence of one-to-many shortest path queries.

3.3 Computational experiments It is not hard to see that the approach outlined in Section 3.2 will yield the same reconstructed path (up to the breaking of ties in the computation) as the approach presented in [9] when no vertices or edges contained in any of the D_i are discarded. To demonstrate that the run time is still feasible, we performed a range of experiments on simulated input data. The underlying road network we used was a map of Germany obtained from OpenStreetMap[1] containing 18 million vertices and 35 million edges. The diameter of this network was 886 km. In this network we took a shortest path of length 400km between two fixed vertices as a *base path*, generated measurement disks of random but bounded radius and distance (between consecutive disks). In particular, we investigated the impact of different average sizes of disks (corresponding to GPS

[1] `http://download.geofabrik.de/osm/`

Precision Sampling Rate	GPS (50m-100m)	GPS (1km-5km)
Point-2-Point	3800585	173920
New	12335325	426544
New(CH)	170816	27546
Precision Sampling Rate	GSM (3km-5km)	GSM (10km-50km)
Point-2-Point	3967486	4821118
New	1008816266	72291484
New(CH)	3309238	388590

Table 1: Comparison of the total number of Dijkstra operations of three map matching methods: point-to-point matching, our new approach without using CHs, our new approach using CHs. The table entries are averaged over 1000 simulated location measurement sequences along a path of length 400km. The columns correspond to different levels of imprecision (GPS and GSM) and different average distances between consecutive measurements.

(3m - 5m) and GSM (2km - 5km) measurements) and of different average distances between consecutive disks (varying between 50 meters and 50 kilometers). The results of these experiments are summarized in Table 1. As can be seen from this table, the use of contraction hierarchies drastically reduces the number of Dijkstra operations and, for GPS-sized measurement imprecisions even outperforms the simple point-to-point matching approach. Unfortunately, we were not able to finish the same set of experiments also with our re-implementation of the approach presented in [9] due to excessive run times. To give an impression: the average number of Dijkstra operations was 978926 even for GPS-sized measurement imprecision and 1km-5km distance between consecutive measurements.

With our approach, the total effort one needs to spend for matching a 400km long route is only a few million (GSM localization) /hundred thousand (GPS localization) Dijkstra operations ($\approx$ few seconds/fraction of a second actual running time on a state-of-the-art CPU core). If cars move at about 100km/h, we can match several thousand (GSM) or several tens of thousands (GPS) cars on a single CPU core. A sub-10k-USD compute server with 48 cores can hence easily match and track hundreds of thousands (GSM) or even millions (GPS) of vehicles in real-time.

In a second set of experiments we investigated the the impact of several parameters on the reconstruction accuracy of our approach. We used 6 different base paths (respectively two of short (125 km), middle (250 km) and long (400 km) length). First we investigated the impact of different average sizes of disks (corresponding to GPS (3m - 5m) and GSM (2km - 5km) measurements) and of different average distances between consecutive disks (varying between 50 meters and 50 kilometers). For each base path and each combination of parameters (i.e. average size of disk and average distance between consecutive disks) we generated 1000 random sequences of measurements. As an example, we display in Figure 6 a histogram of distances A_L between the base path and the reconstructed path for a long base path, GPS-sized measurement imprecisions and average distance between consecutive disks of 75 meters. The results obtained for other combinations of parameters or using the metric A_N instead of A_L were similar. Even for GSM-sized measurements imprecisions we can reconstruct between 90 and 95% of the base path, see Figure 7.

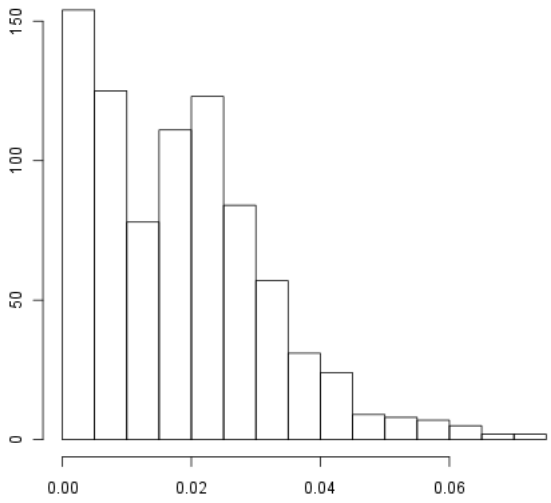


Figure 6: This histogram approximates the distribution of the distance between a base path and the reconstructed path for sequences of measurements generated randomly along the base path. To generate the histogram 1000 simulated sequences of measurements were used and the range $[0, 1]$ of the metric A_L was divided in sub-intervals of length 0.005. We used a path of length 400km, GPS-sized measurement imprecisions and an average distance between consecutive measurements of 75m.

In an additional experiment we investigated the impact of the deviation of the base path from being a shortest path between two vertices in the network on the reconstruction accuracy. To this end, the base

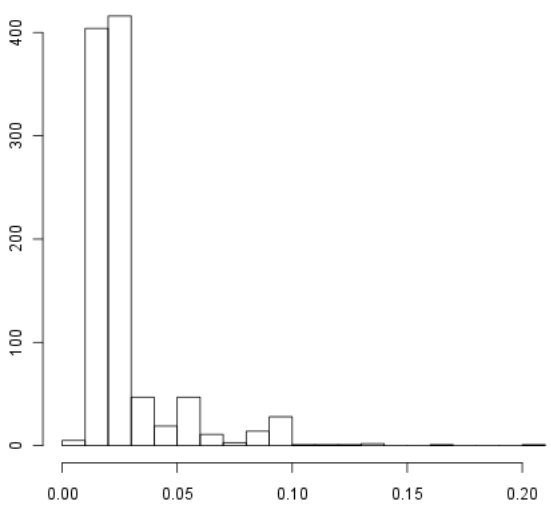


Figure 7: A histogram approximating the distribution of the distance between a base path and the reconstructed path for GSM-sized measurement imprecisions. Length of the base path and distance between consecutive measurements is the same as in Figure 6. Note that in this histogram the range of the metric A_L was divided in sub-intervals of length 0.01.

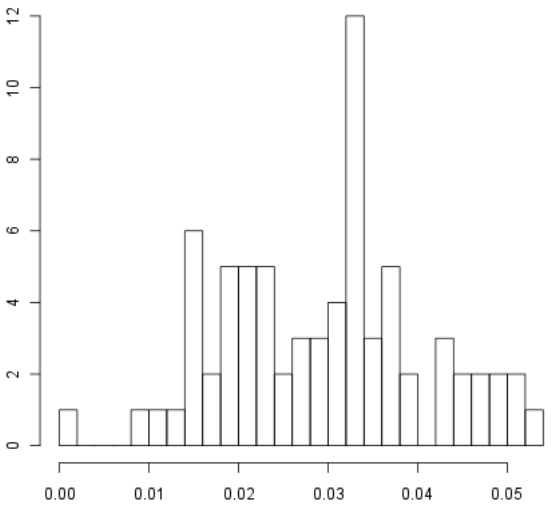


Figure 8: A histogram approximating the distribution of the distance between a base path formed by concatenating five shortest paths and the reconstructed path for GPS-sized measurement imprecisions. Here we used 100 randomly generated measurement sequences along the base path. Note that in this histogram the range of the metric A_L was divided in sub-intervals of length 0.002.

path was chosen to be a concatenation of five shortest paths. Along this path we simulated 100 sequences of measurements. As shown in Figure 8, the reconstruction accuracy slightly decreases compared to Figure 6, which was for a single shortest path, but stays above 90%.

4 Prediction of Trajectories

Recall the problem definition for the problem of *predicting trajectories* on a road network $G = (V, E)$. For a given path $\pi = v_0, v_1, \ldots, v_i$ in G we want to predict how π continues through G, that is, we want to compute a path $\pi' = v_0, v_1, \ldots, v_i, v_{i+1}, \ldots, v_k$ in G. Why is there any hope that good predictions can be computed? Typically, people try not to waste time and move at least partially on shortest or quickest paths. It is only this observation that allows us to come up with any prediction at all. We start with the assumption that people are moving on a path π which is a shortest path from some s to some t and later also consider paths that are not overall shortest but *piecewise shortest-paths*, as this more realistically models the case that on the way back home from work one drops by the butcher, the bank and the bakery.

4.1 Quality Metrics For measuring the performance of our proposed prediction strategies we generate random (partially) shortest paths $\pi = v_0, v_1, \ldots, v_k$ and provide the partial path $\pi' = v_0, v_1, \ldots, v_i$ for all $0 \leq i < k$ as input for the respective prediction of v_{i+1}. A prediction error will result in a mismatch of the node v_{i+1} in π and the proposed v'_{i+1} by the predictor.

As it turns out, the error rate not only depends on the choice of the prediction strategy but also on the relative position on π i.e. the ratio from i to k. This is natural, since near the end of a path, the final destination could be essentially around 'any' corner. All measurements do have in common that a large number of random paths π, which may obey further limitations, are drawn with the intention of evaluating each strategy on an "average" path in G.

The *expected distance between failure* metric uses paths of the same length and maps the quantized relative position on a path in percent ($\frac{i}{k} \cdot 100$) to the expected distance of the next prediction error along with its standard deviation. So if you know your relative position on a concrete path and use a specific prediction

strategy, this metric will tell you how long a correctly predicted path fragment you can expect at that position. A superior strategy naturally exhibits larger distances, in fact the optimum value is the remaining path length. Therefore this value converges to zero at the end of a path.
The *absolute number of errors* metric uses the same quantized relative path position and maps the absolute number of prediction errors which occurred at this relative path point. A superior strategy will not only exhibit a smaller overall sum of this values but different strategies also result in different profiles along the path.

Note that all strategies implicitly assume the ground truth to be a path of infinite length, so the error rate in the second half is likely to be higher. As mentioned before, though, it is natural that it is hard/impossible to make any educated guess about the future path when being close to the final destination as this could be around any corner.

4.2 Strategies We categorize our proposed strategies into two main classes, according to the limitations they have to obey at query time.
Offline strategies may employ extensive precomputation but the stored precomputation data is restricted by a linear space bound with regard to the size of the graph. At query time, an offline strategy is allowed to spend time proportional to the maximum degree of the graph. In particular, offline strategies are not allowed to start complex graph explorations.

Online strategies on the other hand have to obey the same precomputation and storage limitations as offline strategies but are allowed to perform extensive computations at query time, in particular (partial) Dijkstra computations are allowed.

We make this distinction to differentiate between algorithms that can potentially be employed on the simplest mobile devices and such that require at least some computing power at query time.

For the description of the employed strategies assume that we choose a path $\pi = v_0, \ldots, v_{i-1}, v_i, v_{i+1}, \ldots, v_k$ with the intention of predicting v'_{i+1}, so a prediction strategy is given $\pi' = v_0, \ldots, v_{i-1}, v_i$ as input (an offline strategy is given a constant-sized suffix thereof). Observe that the prediction of degree 2 nodes is trivial in this context. Nodes of degree 2 are therefore ignored.
At this point the task of predicting a path trajectory breaks down to the question of estimating the one outgoing edge of v_i we think of being the most likely one to continue the path.

4.2.1 Offline strategies In [8], a very straightforward strategy – which we refer to as *baseline* strategy (`OF_b`) – was proposed. When coming from vertex v_{i-1} and being at v_i we pick as next edge/vertex the outgoing edge from v_i, $\overrightarrow{v_i v'_{i+1}}$ with minimal change of direction compared to $\overrightarrow{v_{i-1} v_i}$. Clearly, this is an offline strategy according to our categorization. The intuition behind this strategy is that shortest/quickest paths tend to be rather straight.

A slightly more involved strategy which we refer to as *simple Dijkstra* (`OF_sD`) works as follows: In a precomputation step we compute a full Dijkstra from each vertex v in the network and remember for each outgoing edge of v how large a subtree (in terms of # of nodes) is hanging below this edge. Clearly this is very time consuming but requires only linear space. At query time, when being at node v_i we choose the outgoing edge *not* leading to v_{i-1} which bears the largest subtree in the shortest path tree from v. Conceptually this is close to choosing the edge where most likely a target chosen uniformly at random in its shortest path subtree resides.

To reduce the enormous computational cost (even though this can be easily parallelized and computed in a few days on a small cluster also for large networks like the US or the whole of Europe), we can slightly modify the precomputation step and start an unidirectional reach-[7]-based Dijkstra at every $v \in V$ and order the outgoing edges of v according to the *longest* path discovered during these searches. This reduces the precomputation time by orders of magnitudes without really affecting the quality of the prediction as we will see. As we employed a reach-based search, we refer to this strategy as *simple reach based Dijkstra* (`OF_rbD`). As this strategy turned out to be essentially equal to `OF_sD`, we have omitted the latter in our experiments. For the precomputation time, we are in the range of a few milliseconds per vertex (on a standard PC, not including the precomputation time for the reach or HH information itself), so even large networks can be preprocessed in a few hours.

Having introduced the reach concept, another obvious strategy is to simply return the edge with highest reach (see Section 2.5) which does not lead back to v_{i-1}. This we call the *reach based* strategy (`OF_rb`). Intuitively, high reach means important edge whereas low reach means unimportant edge.

4.2.2 Online strategies Under the assumption that the ground truth path π is a shortest path from s to some random t, the arguably optimal strategy constructs a shortest path tree (by Dijkstra) starting in s. At any point in time, the known path segment $v_0, \ldots, v_i$ is part of this shortest path tree, and as t is randomly

chosen, picking the outgoing edge of v_i which contains the most nodes in its subtree is the optimal prediction strategy. We call this the *full Dijkstra* (`ON_fD`) strategy. Apart from being quite demanding computationally at query time (a full Dijkstra computation), this strategy breaks down if the path π is not a shortest path but consists of *piecewise* shortest paths only – think of stopping by the bakery, the grocery, and the butcher on your way home from work.

To cope with this problem, we need to detect when the path $v_0, \ldots, v_i$ leaves the shortest path tree rooted at $s = v_0$. If this happens at v_i, we start a Dijkstra[2] at v_i exhibiting the longest suffix of $v_0, \ldots, v_i$ that is (in reverse order) a subpath in the shortest path tree rooted at v_i. Let $v_k, v_{k+1}, \ldots, v_i$ be this suffix (note that we can abort Dijkstra if we have settled v_{k-1} in a different subtree of the shortest path tree). We then start a (full) Dijkstra in v_k and use this for prediction. We call this strategy *lazy full Dijkstra* (`ON_lfD`). Due to space restrictions and the still somewhat high computational demand at query time, we will not report results on that but on the following variant thereof.

Similar to our heuristic offline strategy we simply replace the full Dijkstra computation by a reach-based Dijkstra computation which prunes out the edges with increasing distance from the source. So as long as we move on a path of the shortest-path tree of the reach-based Dijkstra started in v_0, we always pick the edge leading to the 'furthest' node. If we leave the shortest path-tree at v_i we start a backward reach-based Dijkstra starting in v_i to exhibit a suffix $v_k, \ldots, v_i$ as before and use a reach-based Dijkstra rooted at v_k for the further prediction. The next time we leave this shortest path tree, we again start a search for such a suffix. This strategy – which we call *lazy reach-based Dijkstra* (`ON_lrbD`) – allows both for adaptivity in case of piecewise shortest paths as well as good running times due to the heavy pruning of edges in the Dijkstra computation.

4.3 Results

Quality Our two test graphs are based on OpenStreetMap[3] data which was stripped of all features impassable by car. For the two resulting graphs with 321k and 18.57M vertices respectively, we computed the reach of all edges based on a travel time metric as proposed in [7]. From the extensive meta data of the OSM dataset we picked the street type to derive a speeds for individual edges. The larger graph [GER] represents a street map of Germany, the smaller one [MV] of its federal state Mecklenburg-Vorpommern. Due to the fact that part of the OSM data is generated by GPS plots, each road segment is composed of a larger amount of degree two nodes. The average chain length is ≈10.7 segments on [MV] and ≈10.2 on [GER] which corresponds to an average length of less than one kilometer. As direct result ≈80% of both graphs nodes are of degree 2.

For the *expected distance between failure* metric the random shortest paths π were sampled with their length being limited to the interval of 100 km ± 2.5% in the case of [MV] and $500km \pm 2.5\%$ in the case [GER]. For the other metric a lower bound of 150 nodes resulted in minimal path lengths of ≈15 km, being only limited upwards by the respective graphs diameter of $\approx 331km$ for [MV] and ≈1044 km for [GER].

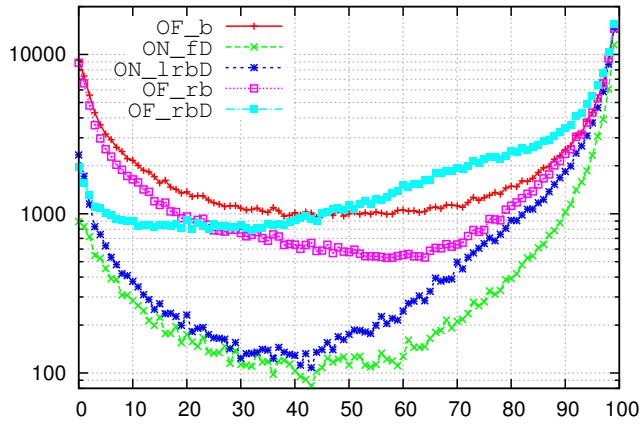
In Figure 9 the *absolute number of errors* metric was employed for 25k random shortest paths on [MV] and 10k random shortest paths on [GER]. Common to all strategies – be it on- or offline – is the fact that close to the start and the end, the prediction quality is pretty bad, which was to be expected. Furthermore – no surprise! – online strategies are far more accurate than offline strategies. Amongst the offline strategies, the baseline predictor (`OF_b`)fares worst on the first half, leading to a total prediction failure rate of 6.25% on [GER], that is, on 6.25% of all nodes on the path *with degree larger than 3* a prediction error has occurred. Employing the reach-based Dijkstra strategy (`OF_rbD`) we obtain much better results at the beginning of the paths but getting worse towards the end, resulting in a 5.98% failure rate overall. The purely reach based strategy `OF_rb` uniformly exhibits better performance than the baseline strategy and except for the beginning also than `OF_rbD`, its total failure rate is 4.03%. It is natural to combine the purely reach-based strategy with the reach-based Dijkstra strategy for the best offline prediction rate. Unfortunately, it is not so clear a priori when one gets better than the other; in case of [MV] this point was at around 22% of the path length, in [GER] it was around 15%. With some tuning we were able to find mixing parameters such that the prediction failure rate of such a mixed offline strategy became 3.21%.

Our online prediction strategies perform much better. Given that we are working with single shortest paths only, the *full Dijkstra* strategy (`ON_fD`) yields an almost perfect result with a total failure rate of 0.95% only. The more practical *lazy reach-based Dijkstra* strategy is not much worse with a total failure rate of 1.59%. In general, the results for [MV] and [GER] are comparable.

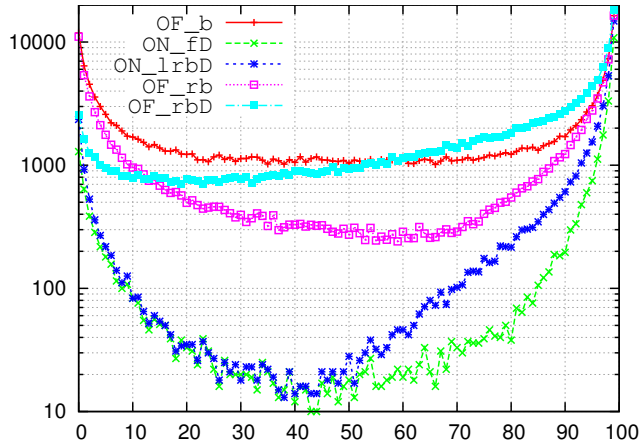
Finally we consider the *expected distance between failure* metric, which is arguably the most intuitive one,

[2] If the graph has asymmetric edge costs, we run this Dijkstra on the reversed graph.

[3] `http://download.geofabrik.de/osm/`



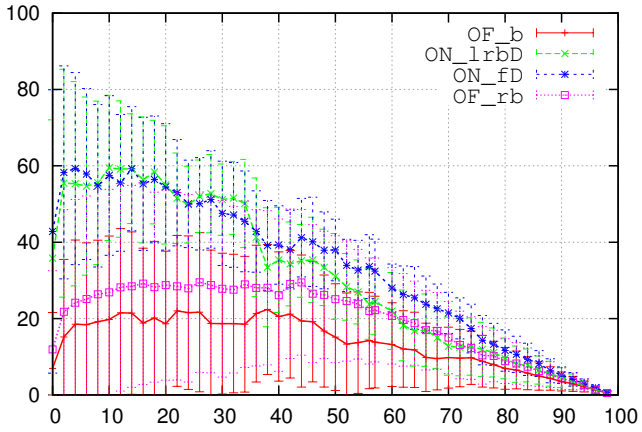
(a) Absolute amount of prediction errors by the respective relative position on the path for MV, sampled on 25K paths



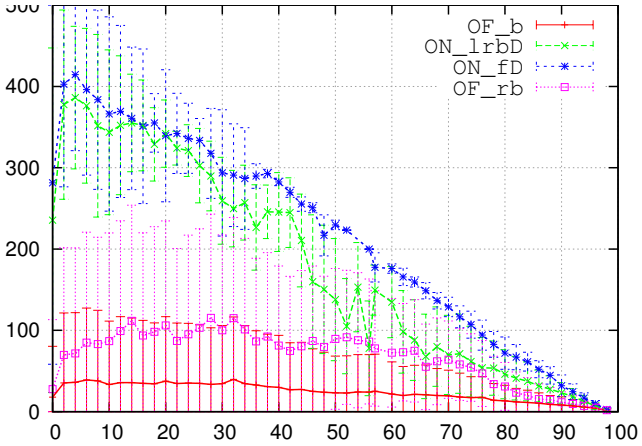
(b) Absolute amount of prediction errors by the respective relative position on the path for GER, sampled on 10K paths

Figure 9: *Absolute number of errors* metric. For fig. (a) 25k random paths on [MV] graph were sampled. Accordingly 10k paths on the [GER] graph for fig. (b).

in Figure 10. For each strategy we have depicted the expected length of a correctly predicted path chunk when being at a certain relative position, comparing our two online strategies with the baseline strategy and the simple reach-based Dijkstra offline strategy. It is clear that our online strategies are far superior to the baseline strategy. In this metric the choice of the underlying road network makes a bigger difference. Both online strategies are gaining a significant amount of accuracy on the [GER] graph, to the point that the average distance to the next prediction error is always more than ≈80% of the remaining path length, compared to ≈60% on the [MV] graph. In contrast to that, the *baseline* strategy's average prediction distance is always below 50 km or 10% on the [GER] graph compared to 20 km or 20% on the [MV] graph. The other reach-based offline strategy performs slightly better but is still way below our online strategies.



(a) 25k sampled paths of 100km length on [MV]



(b) 10k sampled paths of 500km length on [GER]

Figure 10: The *expected distance between failure* metric is limited by the remaining path length. The *full Dijkstra* strategy shows nearly optimal results.

In real-world scenarios, the trajectories of mobile users are often not exact shortest s-t-paths but they tend to be composed of several shortest-path segments (from work to the bakery, then to the butcher, then to the florist before driving home). In Figure 11 we present results for such problem instances constructing a series of 50+90+30+100+70 km shortest path pieces resulting in an overall length of 340 km. Both *lazy reach-based Dijkstra* approaches and the *baseline* approach are producing characteristic profiles along each shortest path piece being limited by the distance of the actual piece ending, while again the latter approach is far inferior in absolute performance. The *full Dijkstra*

approach on the contrary completely fails to produce reliable predictions behind the first shortest path piece as the available shortest path information at v_0 is meaningless from there on.

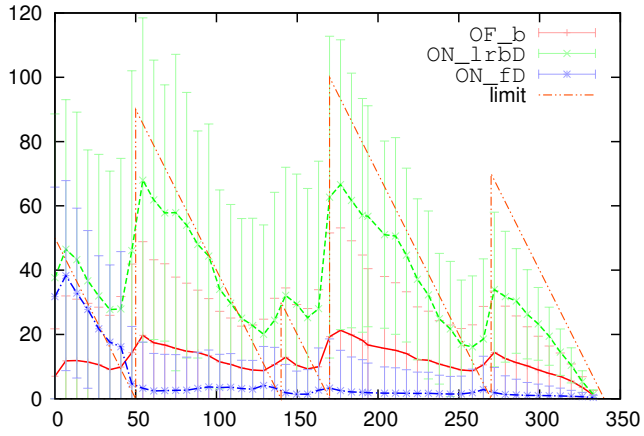


Figure 11: *Expected distance between failure* metric for series of piecewise shortest paths with a length of $50+90+30+100+70 = 340$ km. The "limit" line plots the remaining length of the actual shortest path.

Cost Let us briefly review the cost for predicting trajectories at runtime; we focus on the number of Dijkstra operations as this is determining the real-time applicability of our algorithms. We count the total number of Dijkstra operations that were performed while running our prediction strategies on a shortest path of 500km length.

strategy	# Dijkstra polls
`all Offline strategies`	0
`ON_fD`	$\approx 18.5 \times 10^6$
`ON_lrbD`	$\approx 6.32 \times 10^5$

Table 2: Computational cost per path for different strategies in terms of Dijkstra polls on the [GER] graph at prediction time. Offline strategies only spend constant time and no Dijkstra poll at runtime, our most competitive online strategy is `ON_lrbD`. Average over 10k random shortest paths of length $\approx$ 500km.

The good performance of `ON_lrbD` in Table 2 is due to two reasons: first, a single reach-based Dijkstra visits only a small fraction of the nodes compared to a full Dijkstra, furthermore, only few prediction requests trigger a reach-based Dijkstra computation. More concretely, 99.25% of all prediction requests in our experiment can be answered using the already existent shortest path tree. Only for the remaining 0.75%, a reach-based Dijkstra has to be started, resulting in the total number of polls as shown in Table 2.

Strategy `ON_lrbD` in total for the *whole path* uses less than one tenth of the Dijkstra polls that are necessary for a full Dijkstra on the network (which equals the strategy `ON_fD` which is much worse for only piecewise-shortest paths, though). Again, if we assume the car to drive at about 100 km/h (so the trip takes $\approx$ 5 hours), a single core can perform predictions for more than ten thousand vehicles at the same time. Hence with a not too expensive compute server with let's say 100 cores, we can easily predict trajectories for one Million vehicles in real-time.

4.4 Extensions So far we have only considered the problem of predicting the path. A natural extension is to compute a mapping Time$\rightarrow \mathbb{R}^2$, that is, predict at what time one expects the mobile user to be at what position. This has to take into account both existing speed-limits on the road segments of the predicted route as well as the observed driving speed within the known trajectory.

5 Applications, Combinations and more

While the community driven OpenStreetMap project has become a serious contender as provider of accurate road map data, commercial vendors like Navtec or TeleAtlas still have an edge when it comes to higher level data like traffic flow and traffic density. Acquisition of such data is difficult, though, and requires either access to roadside equipment for traffic census or cooperations with mobile phone network providers. Our proposed map matching algorithm is simple enough that widespread employment on cell phones of community members seems feasible (even on devices without GPS) allowing for large scale acquisition of such traffic data.

One immediate application for path prediction methods is in the context of dead-reckoning protocols. Here a mobile user is to periodically send its position to a server. When the required bandwidth for transmission is an issue, predicting the motion of a user both on the client/user side as well as the server side can result in considerable savings by only transmitting deviations from the prediction as was shown in [8]. With our improved prediction method, these savings can be even further increased. Navigation systems, in particular the ones built-in by car manufacturers, have become 'always-on' devices, so even if no target has been given by the user and no route planning takes place, they acquire the current position. Accurate path prediction routines allow the device to call attention to points of interest that lie on the predicted route. Examples are gas stations, restaurants, shopping malls, parking lots

but also areas of difficult road conditions or traffic jams.

The combination of Map Matching and Path Prediction also makes a lot of sense, in particular when the mobile device is not equipped with GPS: for a given sequence of location measurements our map matching routine identifies an initial path, which is then continued via path prediction routines. As map matching under very imprecise location information tends to be inaccurate for the first few and the last few measurements, one would discard those parts and employ our path prediction routines on the remaining path fragment. This could be the basis for a rudimentary navigation system that works only based on GSM location data.

One apparent danger is the fact that cell phone providers with the help of map matching and path prediction techniques have access to massive amounts of quite precise real-time trajectory data which – even worse – is linked to individual persons. With a medium-sized cluster, millions of users could be traced in real-time. While the generated data is certainly valuable e.g. for detection or prediction of traffic conditions, it is also highly sensitive with respect to the privacy and anonymity of the individual. An interesting open question is how the raw cell phone data can be obfuscated early on in the process such that purposive surveillance gets impossible.

In a broader context, path prediction might not only be used on street data, but for example to trace IPs. So a possible alternative input would be the sequence of the last tunneled proxy-servers. For that it remains to see how well concepts like reach apply to such communication networks (at first sight they do).

References

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

[2] H. Bast, S. Funke, and D. Matijevic. *Ultrafast shortest-path queries via transit nodes*, volume 74 of *DIMACS Series on Disrecte Mathematics and Theoretical Computer Science*, pages 175–192. AMS, Providence, RI, 2009.

[3] V. Batz, D. Delling, P. Sanders, and C. Vetter. Time-dependent contraction hierarchies. In *Proc. 11th Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 97–105. SIAM, 2009.

[4] S. Brakatsoulas, D. Pfoser, R. Salas, and C. Wenk. On map-matching vehicle tracking data. In *Proceedings 31st International Conference on Very Large Data Bases (VLDB)*, pages 853–864. ACM, 2005.

[5] R. Geisberger, P. Sanders, D. Schultes, and D. Delling. Contraction hierarchies: faster and simpler hierarchical routing in road networks. In *Proceedings of the 7th international conference on Experimental algorithms*, WEA'08, pages 319–333, Berlin, Heidelberg, 2008. Springer-Verlag.

[6] A. V. Goldberg, H. Kaplan, and R. F. Werneck. Reach for a â^— : Efficient point-to-point shortest path algorithms. In *In: Workshop on Algorithm Engineering and Experiments*, pages 129–143, 2006.

[7] R. Gutman. Reach-based routing: A new approach to shortest path algorithms optimized for road networks. In *Proc. 6th Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 100–111. SIAM, 2004.

[8] A. Leonhardi, C. Nicu, and K. Rothermel. A map-based dead-reckoning protocol for updating location information. In *IPDPS*, 2002.

[9] Y. Lou, C. Zhang, Y. Zheng, X. Xie, W. Wang, and Y. Huang. Map-matching for low-sampling-rate GPS trajectories. In *Proc. ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM SIGSPATIAL GIS)*, pages 544–545. ACM, 2009.

[10] F. Pereira, H. Costa, and N. Pereira. An off-line map-matching algorithm for incomplete map databases. *European Transport Research Review*, 1:107–124, 2009.

[11] M. Quddus, W. Ochieng, and R. Noland. Current map-matching algorithms for transport applications: state-of-the art and future research directions. *Transportation Research Part C: Emerging Technologies*, 15:312 – 328, 2007.

[12] P. Sanders and D. Schultes. Highway hierarchies hasten exact shortest path queries. In *13th European Symposium on Algorithms (ESA'05)*, pages 568–579, 2005.

[13] H. Yanagisawa. An offline map matching via integer programming. In *Proc. 20th International Conference on Pattern Recognition (ICPR)*, pages 4206–4209. IEEE, 2010.

[14] H. Yin and O. Wolfson. A weight-based map matching method in moving objects databases. In *Proc. 16th International Conference on Scientific and Statistical Database Management (SSDBM)*, pages 437–438. IEEE, 2004.