FAST ALGORITHMS FOR MINIMUM ERROR MAP-MATCHINGS

Renaud Chicoisne, Universidad de Chile, renaud.chicoisne@gmail.com
Daniel Espinoza, Universidad de Chile, daespino@gmail.com
Fernando Ordoñez, Universidad de Chile, fordon@dii.uchile.cl

ABSTRACT

GPS data of vehicles travelling on road networks can be used to estimate travel times. This requires the identification of the corresponding paths in the network. We developed algorithms identifying a path minimizing a special distance with a GPS trajectory even in presence of cycles. Mild assumptions over the GPS precision and the use of heuristic steps allow fast map matchings. We compared these algorithms over 30000 real and generated GPS samples on a grid graph and two real life networks. We show that on average, our algorithm and the heuristic return in minutes paths that have respectively 92.5% and 89% of the trajectory in a corridor of one meter around them.

Keywords: Map matching, Travel time estimation

1 INTRODUCTION

Many real world problems use weighted networks as a basis and thus need a graph representation of the real object we are working on. For example when scheduling police patrols, computing routes with Personal Navigation Assistants or solving any routing problem. This project has been developed as part of a research effort with the Santiago fire department which aimed to improve the dispatching response. In our case, we wanted to solve a shortest path problem in a real road network and we chose to approximate travel times over every edge of the network with static durations representing the congestion of the streets. Given a trajectory of georeferenced points and a network representation, our objective is to find the path of the map associated to this trajectory. As every real data set, it comes with significant measurement errors and then makes the path search non trivial.

In the literature there is several families of algorithms designed to solve this map matching problem. The first and most intuitive ones are the constructive algorithms that successively build path when iterating over the trajectory's points like in White et al., 2000 designed heuristics that find subgraphs locally close to the GPS trajectory, without any guarantee that the returned subgraph is a path. Quddus et al., 2003, Marchal et al., 2004, Li et al., 2005 and Zhang et al., 2005 are doing a semi global optimization linking locally close path parts between themselves and show results on a few controlled instances. In Yang et al., 2005 the authors improve the accuracy of the previous methods with bigger neighbourhoods and then the effectiveness of the algorithms when the GPS sampling rate is low. Joshi, 2002 and Wu et al., 2007 find heuristical map matchings by non-exhaustively enumerate path candidates and compare them with original metrics. In Brakatsoulas et al., 2005 and Wei et al., 2013 they aim to find the path minimizing the Fréchet distance with the trajectory using dynamic programming. Newson and Krumm, 2009, Lou et al., 2009, Bierlaire et al., 2013, Westgate et al., 2013 and Jenelius and Koutsopoulos, 2013 designed algorithms finding high-likelihood roads taken by a trajectory supposing that the error of measurement of the velocity data follows a normal distribution. For more information of the different types of map matching algorithms, see Quddus et al., 2007 and Wei et al., 2013.

The novelty of our projection algorithm resides in the fact that it finds the path of the network that minimizes the distance with the data trajectory in a theoretically and computationally fast time. Further, when facing long trajectories or when needing fast responses we designed a heuristic that outruns by two orders of magnitude the optimal method at a small cost of the quality of the solution.

We first present an algorithm that finds a path -that can contain loops- that minimizes some distance with the GPS trajectory. We then show its theoretical complexity and some ways to speed it up. Second, we present an heuristic that can cope with very fast map matching needs. And finally, we compare in the last section the effectiveness and speed of both algorithms.

Let define some notations that will be useful in this paper: let $G=(V,E)$ be a network representation, with $|E|=m$ and $|V|=n$, $d\in\mathbb{R}_+^m$ represents the lengths of the network's edges and $(p_k)_{k\in\{1,\dots,q\}}$ is a set of ordered points of the trajectory.

2 AN OPTIMAL ALGORITHM

Let $R=(r_j)_{j\in\{1,\dots,|R|\}}\subseteq E$ the real path of the trajectory and $C=(c_i)_{i\in\{1,\dots,|C|\}}\subseteq E$ the path found with some algorithm. In order to check if the algorithm is efficiently identifying the correct subgraph of each trajectory, we defined several error measures, first the maximum real error:

$$\mathrm{RE}(C) := \max_{i \in \{1,\dots,|C|\}} \min_{j \in \{1,\dots,|R|\}} d(c_i, r_j)$$

That represents the maximum distance between an edge of the path found and the real path the trajectory comes from. And the total ordered error:

$$\mathrm{OE}(C) := \sum_{k=1}^{q} d(p_k, c_{e(k)})$$

With:

$$e(k) := \operatorname{argmin}\{d(p_k, c_i) : i \geq e(k-1)\}$$
$$e(0) := 1$$

That represents the greatest distance between a point and the edges of the path C following the edge associated with the anterior point. We can notice that this error measure penalizes a lot the paths cutting tight turns for example.

We will now present a way to find the optimal path for the ordered error measure. Let define the directed graph $\bar{G} = (\bar{V}, \bar{E})$ such that $\bar{V} := \{s\} \cup V_1 \cup V_2 \cup \dots \cup V_q \cup \{t\}$ with $V_r := \{v_r^e, \forall e \in E\}$ for each point p_r of the cloud. Then, the edge set of $\bar{G}$ is $\bar{E} := E_1 \cup E_2 \cup \dots \cup E_q \cup E_{q+1}$ with:

$$E_1 := \{(s, v_1^e), \forall e \in E\}$$
$$E_r = \bigcup_{(i,j) \in E} \left\{ \left(v_{r-1}^{(i,j)}, v_r^{(i,j)}\right), \left(\left(v_{r-1}^{(i,j)}, v_r^{(j,k)}\right)_{k \in \delta^-(j)}\right) \right\}, \forall r \in \{2,\dots,q\}$$
$$E_1 := \{(s, v_1^e), \forall e \in E\}$$

With the following weights:

$$w_{(s, v_1^e)} = d_{p_1}^e, \forall e \in E$$
$$w_{(v_{r-1}^{(i,j)}, v_r^{(i,j)})} = d_{p_r}^{(i,j)}, \forall r \in \{2,\dots,q\}, \forall (i,j) \in E$$
$$w_{(v_{r-1}^{(i,j)}, v_r^{(j,k)})} = d_{p_r}^{(j,k)}, \forall r \in \{2,\dots,q\}, \forall (i,j) \in E, \forall k \in \delta^-(j)$$
$$w_{(v_q^e, t)} = 0, \forall e \in E$$

With $d_p^e = d(e, p)$ the euclidian distance between point p and the closest point of edge e.

In the following we show an example of graph G in figure 1 and its associated graph $\bar{G}$ in figure 2.

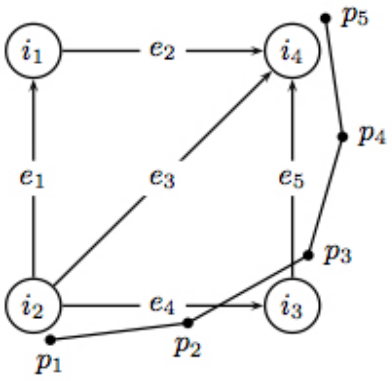

Figure 1: Example of graph with trajectory

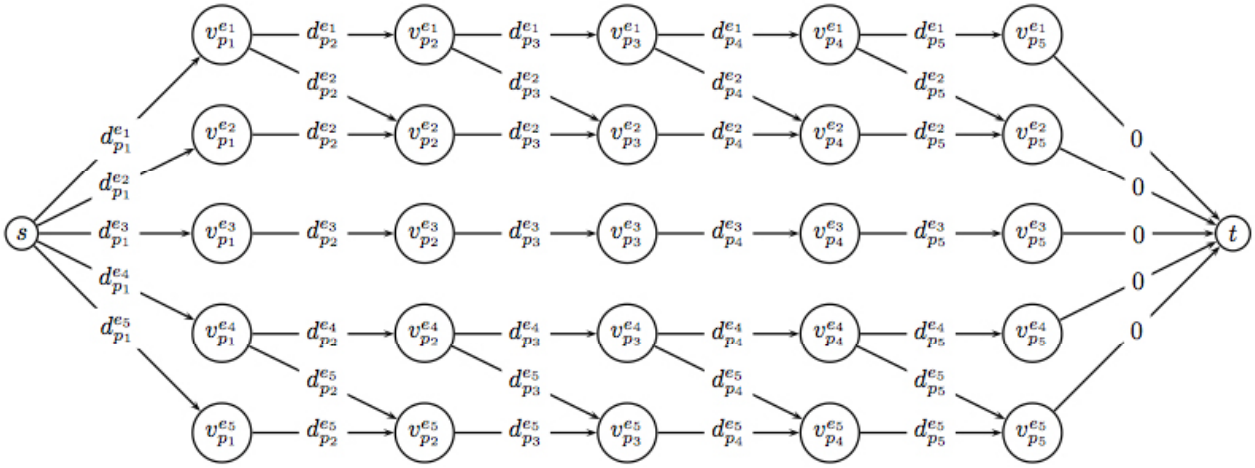

Figure 2: $\bar{G}$ associated to G

Let prove that we can extract a path that minimizes the total ordered distance with the trajectory of points from every shortest *st*-path $C=(c_k)_{k\in\{1,\dots,q\}}$ in the graph $\bar{G}$. First, we can see that a *st*-path in $\bar{G}$ has exactly $q+1$ edges, the last one being dummy. The *k*-th edge c_k associates the *k*-th trajectory point p_k with some edge e_k of the graph, which costs the distance between p_k and e_k. By construction, we know that e_k is an outgoing edge of e_{k-1} or e_{k-1} itself. Consequently, $(e_1,\dots,e_k)$ defines a path in the original network *G*, and does not allow an association with an edge already associated that is not going out form e_{k-1}. Putting everything together, a shortest *st*-path in $\bar{G}$ is giving the association path-trajectory that has the minimum oriented error.

An important observation about this algorithm is that it allows to backtrack and form cycles.This can be a good feature when we know that the trajectory loops at some point, but in the opposite case we must forbid the algorithm to do so when the GPS error induces a local backtrack. In this goal, we used a double priority heap instead of the classical one. When running Dijkstra, we first optimize with respect to the lengths of the network $\bar{G}$, and second, we force it to choose the path with minimum real path length.

Now let compute this last algorithm's theoretical complexity. Noticing that $\bar{G}$ is acyclic, we can find a *st*-shortest path in $O(\bar{V}+\bar{E})$ time with a breadth first search or a depth first search. And we know that:

$$|V_r|=m,\ \forall\, r\in\{1,\dots,q\}\Rightarrow|\bar{V}|=2+qm$$

$$|E_1|=|E_{q+1}|=m$$

$$|E_r|=\sum_{(i,j)\in E}\left[1+|\delta^-(j)|\right]=m+\sum_{i\in V}\delta^+(i)\delta^-(i),\ \forall\, r\in\{2,\dots,q\}$$

$$|\bar{E}|=2\mathrm{m}+(q-1)\left(m+\sum_{i\in V}\delta^+(i)\delta^-(i)\right)=(q+1)m+(q-1)\sum_{i\in V}\delta^+(i)\delta^-(i)$$

Putting everything together the complexity $C(\bar{G})$ of finding a shortest *st*-path over graph $\bar{G}$ is:

$$C(\bar{G})=O(|\bar{V}|+|\bar{E}|)=qm\min\{D_G^+,D_G^-\}$$

With $D_G^+=\max_{i\in V}\delta^+(i)$ and $D_G^-=\max_{i\in V}\delta^-(i)$ respectively the indegree and outdegree of *G*. In the worst case we then have: $C(\bar{G})=O(qm^2)$.

The computationally heavy part of the previous algorithm is the construction of the derivated network $\bar{G}$. We observed that if we have some upper bound δ over the maximum error between the real path and its trajectory, we can build the derivated network $\bar{G}$ with the edges of *E* in a corridor of width δ around the trajectory. Further, we can remark that the only weights we need during the execution of our algorithm are the ones we are looking during the shortest path algorithm's loop. Consequently, we must compute the weight of an edge of $\bar{E}$ only when we are looking at an outgoing edge *e* of the current node with the lowest label. Doing so allows us to considerably speed-up the algorithm in the practice, although it does not change the theoretical worst-case complexity.

3 MAP MATCHING HEURISTIC

In this section we present a heuristic that constructs in an original way a subgraph of the trajectory. The main idea of our method is to lower the length of the edges that are close to the trajectory and then execute a shortest path algorithm between the first point of the trajectory and its last one. This way we ensure that we have a path, to the difference of the state of the art algorithms. Given a trajectory, we iterate over all the points that belong to it. For each of these points, say p_k , we find the edges that are within a radius *R>0* of p_k and we lower their length by the distance between p_k and p_k . Once we iterated over the whole line, we look for two sets of nodes: the set $S:=\{v\in V: d(p_0,v)\leqslant R\}$ of the nodes closest to the first point p_0 and the set $T:=\{v\in V: d(p_q,v)\leqslant R\}$ of the nodes closest to the last point p_q . Then we add two more nodes *s* and *t*, the edges *(s,v)* of weight $w_{(s,v)}=d(p_0,v),\ \forall\, v\in S$ and the edges *(v,t)* of weight $w_{(v,t)}=d(p_q,v),\ \forall\, v\in T$ and we finally execute a shortest path algorithm between *s* and *t*. Note that when we compute the modified weights we keep the positive part of the length: this will avoid situations of negative cycles for Dijkstra algorithm. In order to have a more precise identification of the path's edges, we prefered to penalize the edges' length with small quantities

several times rather than penalize a lot once. In this aim we densified the cloud of points we disposed of, imposing that no point could be separated from its successor or predecessor further than some distance $d_{\max}$. We can see the principle pseudo-code in algorithm 1, where `dijkstra`(*G,w,p,q*) a routine that computes a shortest path inside graph *G=(V,E)* with weights $w \in \mathbb{R}^{|E|}$ between point *p*'s closest node in *G* and *q* closest node in *G* and *SP* is a subset of nodes corresponding to the path followed by the trajectory (Output of the algorithm).

Algorithm 1 heuristic

Require: $G = (V, E)$, $P = (p_k)_{k\in\{1,\dots,q\}}$, R, $d \in \mathbb{R}^E$, $d_{\max}$
- $w \leftarrow d$, $p_o \leftarrow p_1$
- **for** $k = 1, \dots, q-1$ **do**
  - $n_k \leftarrow \lfloor d(p_k, p_{k+1})/d_{\max} \rfloor$
  - **for** $l = 0, \dots, n_k$ **do**
    - $p \leftarrow p_k + (p_{k+1} - p_k) \cdot l \frac{d_{\max}}{d(p_k, p_{k+1})}$
    - **for** $e : d(p, e) \leqslant R$ **do**
      - $w_e \leftarrow (w_e - d(p_o, p))_+$
    - $p_o \leftarrow p$
- **for** $e : d(p_q, e) \leqslant R$ **do**
  - $w_e \leftarrow (w_e - d(p_o, p_q))_+$
- $S \leftarrow \{v \in V : d(p_0, v) \leqslant R\}$
- $T \leftarrow \{v \in V : d(p_q, v) \leqslant R\}$
- $\bar{G} = (V \cup \{s, t\}, E \cup \{(s, v)_{v\in S}, (v, t)_{v\in T}\})$
- **for** $v \in S$ **do**
  - $w_{(s,v)} \leftarrow d(p_0, v)$
- **for** $v \in T$ **do**
  - $w_{(v,t)} \leftarrow d(p_q, v)$
- $SP \leftarrow$ `dijkstra`(G, w, s, t)
- **return** SP

Now we will show that this heuristic has a nice theoretical complexity. First, there is at most $Q := \frac{1}{d_{\max}} \sum_{k=1}^{q-1} d(p_k, p_{k+1})$ points belonging to the dense data. The complexity of the entire projection framework can be detailed as follows: for each of the (at most) *Q* points of the dense trajectory, find its closest edges has O(*m*) complexity, giving a total complexity for all the dense points of O(*Qm*). Dijkstra's algorithm using binary heap over the graph with modified weights has a complexity in O(*m* + *n* log *n*). Putting everything together, the global complexity of the projection algorithm for each trajectory is in O(*Qm*+*n* log *n*). Which is fast in an algorithmic sense, but in practice can be quite slow because we have to compute the distance between every pair point-edge of the graph. Then, in order to speed up the algorithm we can use the same trick we used to improve the last algorithm. Indeed, the only weights we need during the execution of Dijkstra's algorithm are the outcoming edges of the minimum label nodes. Consequently we can modify the shortest path algorithm in the same way and compute dynamically the distances between trajectory points and edges. The pseudo algorithm is presented in algorithm 2, where $(\bar{p}_k)_{k\in\{1,\dots,Q\}}$ is the densified trajectory, *H* is a binary heap, `insert_heap` (H, v, π) inserts the node index *v* with value π into the heap *H*, `change_val` (H, v, π) changes the value of the node index *v* inside the heap *H* to the value π , `get_root` (H) returns the node index of the root node of the heap *H* and `delete_root` (H) deletes the root node of heap *H*.

Algorithm 2 heuristic_On_the_fly

Require: $G = (V, E)$, $P = (\bar{p}_k)_{k\in\{1,\dots,Q\}}$, R, $d \in \mathbb{R}^E$, $d_{\max}$

```
w ← d
T ← {v ∈ V : d(p_q, v) ⩽ R}
H ← ∅
for v ∈ V do
  if d(p_0, v) ⩽ R then
    insert_heap(H, v, d(p_0, v))
while H ≠ ∅ do
  i ←get_root(H)
  delete_root(H)
  if i ∈ T then
    return i
  for j ∈ δ+(i) do
    for k = 1, ..., Q − 1 do
      if d(p̄_k, (i, j)) ⩽ R then
        w_ij ← [w_ij − d(p_k, p_{k+1})]_+
    if π_j > π_i + w_ij then
      π_j ← π_i + w_ij
      father_j ← i
      if j ∉ H then
        insert_heap(H, j, π_j)
      else
        change_val(H, j, π_j)
return T not reached.
```

However, in some situations the algorithm can identify a path that does not correspond to the original itinerary. When the dense trajectory passes through a very dense street zone and has longer segments than the edges in the neighbourhood, the algorithm can choose to pass trough closer edges although they are not the right ones (see figure 3) where the dotted path is the real one and the dashed path the subgraph found by the algorithm). Note that it is a data-induced problem.

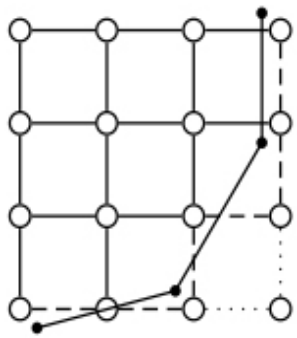

Figure 3: Sparse trajectory Vs. dense graph.

Another common case of partial failure of the algorithm is the presence of tight turns in the trajectory: due to the proximity between the forward and the backward parts, the edges connecting them in the entire graph will have near-zero weights. In some cases those connecting edges will have lower values than the real path edges and consequently, the algorithm will cut off the entire turn, passing directly at the end of it (See figure 4).

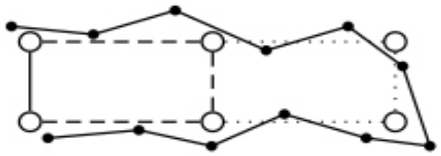

Figure 4: Tight turn type errors.

The optimal algorithm for oriented error avoids this type of mismatch.

4 EXPERIMENTAL RESULTS

We ran our two algorithms over two real networks: Santiago de Chile (*n=330000, m=660000*) and Seattle, WA (*n=420000, m=860000*) and a artificial grid network of *100*x*100* nodes with edges of length *100* meters. We had real GPS trajectories for the particular case of Santiago de Chile and we generated artificial trajectories for each network in order to have more computational results. For each network we generated *100* different paths. The generated paths are shortest paths between two nodes uniformly drawn in a box of the densest zone of its network with respect to an aleatory uniform weight. Consequently, the family of trajectories we generated in this paper have an minimal expected number of edges. Second, we sampled and noised with gaussian perturbation each of these paths with sampling steps $s \in [10,500]$ meters and Gaussian errors $e \sim N(\mu = 0, \sigma)$ such that $\sigma \in [1,50]$. The parameters we used for the algorithms were the following: two consecutive points of the trajectory are closer than $d_{\max} = 20$ meters and the radius parameter used in both algorithms is $R = \delta = 500$ meters. This work was coded entirely in C programming language and run over an Intel Core 2 Duo 2.4 GHz with 4Go 1067 MHz DDR3 RAM.

In table 1 we can see that the geometric execution time of the optimal algorithm is two orders of magnitude greater than the heuristic one. As expected, the mean oriented error is way lower for the paths found by the optimal algorithm than those computed by the heuristic. Further, this result goes as well for the real maximum error measure, meaning that the map matching of the optimal algorithm is better than for the heuristic. Our computational results show that both algorithm's real error measures depends of the sampling step in an almost linear way, and that the standard deviation of the Gaussian noise we applied to it has a weak influence up to $\sigma \leqslant 20$ meters which is in the range of modern GPS sensitivity.

Table 1: General statistics

Algorithm	Geometric Exec Time[s 10^{-6}]	Mean *RE(C)* [m]	Mean O*E(C)* [m]
Optimal	3115034	179,1	79,2
Heuristic	56585	228,5	113,5

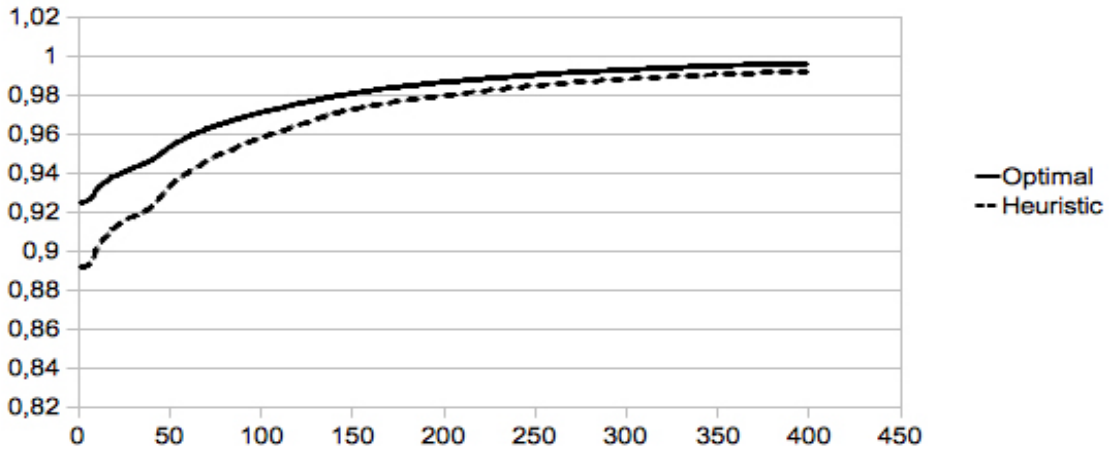

Figure 5: Proportion of path edges Vs. distance to real path

In figure 5 we can see that 92.5% of the path' edges found by the optimal algorithm are at less than 1 meter away from the real path, and 89% for the heuristic.

5 CONCLUSIONS

In this paper, we defined an oriented measure of error that takes in account the ordering of the path when computing its discrepancy level with the trajectory. We developed an algorithm that finds in an optimal way the path minimizing the oriented error measure and added some algorithmic steps speeding it up when information about the measurement error of the GPS signal is known. In the case of dynamic map matching, this algorithm can be prohibitive in terms of computational time, so we constructed a heuristic that runs two orders of magnitude faster at a low cost of the fidelity of the path it founds.

References

M. Bierlaire, J. Chen, and J. Newman (2013) A probabilistic map matching method for smartphone GPS data. **Transportation Research Part C: Emerging Technologies**, 26:78–98.

S. Brakatsoulas, D. Pfoser, R. Salas, and C. Wenk (2005) On map-matching vehicle tracking data. **Proceedings of the 31st international conference on Very large data bases**, pages 853–864.

E. Jenelius and H.N. Koutsopoulos (2013) Travel time estimation for urban road networks using low frequency probe vehicle data. **Transportation Research Part B: Methodological**, 53:64–81.

R.R. Joshi (2002) Novel metrics for map-matching in in-vehicle navigation systems. **Intelligent Vehicle Symposium, 2002. IEEE**, 1:36–43.

X. Li, H. Lin, and Y. Zhao (2005) A connectivity based map matching algorithm. **Asian Journal of Geoinformatics**, 5(3):69–76.

Y. Lou, C. Zhang, Y. Zheng, X. Xie, W. Wang, and Y. Huang (2009) Map-matching for low-sampling-rate GPS trajectories. **Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems**, pages 352–361.

F. Marchal, J. Hackney, and K.W. Axhausen (2004) Efficient map-matching of large GPS data sets-tests on a speed monitoring experiment in zurich. **Arbeitsbericht Verkehrs-und Raumplanung**, 244.

P. Newson and J. Krumm (2009) Hidden Markov map matching through noise and sparseness. **Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems**, pages 336–343.

O.A. Nielsen and M.V. Sørensen (2008) The akta road pricing experiment in copenhagen. **Road Pricing, the Economy and the Environment**, pages 93–109.

M.A. Quddus, W.Y. Ochieng, L.N. Zhao, and R.B. Noland (2003) A general map matching algorithm for transport telematics applications. **GPS solutions**, 7(3):157–167.

M.A. Quddus, W.Y. Ochieng, and R.B. Noland (2006) Integrity of map-matching algorithms. **Transportation Research Part C: Emerging Technologies**, 14(4):283–302.

M.A. Quddus, W.Y. Ochieng, and R.B. Noland (2007) Current map-matching algorithms for transport applications: State-of-the art and future research directions. **Transportation Research Part C: Emerging Technologies**, 15(5):312–328.

H. Wei, Y. Wang, G. Forman, and Y. Zhu (2013) Map matching by Fréchet distance and global weight optimization. **Working Paper**.

B.S. Westgate, D.B. Woodard, D.S. Matteson, and S.G. Henderson (2013) Large-network travel time distribution estimation, with application to ambulance fleet management. **PhD Thesis**.

C.E. White, D. Bernstein, and A.L. Kornhauser (2000) Some map matching algorithms for personal navigation assistants. **Transportation Research Part C: Emerging Technologies**, 8(1):91–108.

D. Wu, T. Zhu, W. Lv, and X. Gao (2007) A heuristic map-matching algorithm by using vector-based recognition. **Computing in the Global Information Technology, 2007. ICCGI 2007. International Multi-Conference on**, pages 18–18.

J. Yang, S.P. Kang, and K. Chon (2005) The map matching algorithm of GPS data with relatively long polling time intervals. **Journal of the Eastern Asia Society for Transportation Studies**, 6:2561–2573.

M. Zhang, W. Shi, and L. Meng (2005) A generic matching algorithm for line networks of different resolutions. **Workshop of ICA Commission on Generalization and Multiple Representation Computering Faculty of A Coruña University-Campus de Elviña, Spain.**