2015 IEEE 18th International Conference on Intelligent Transportation Systems

Fast Hidden Markov Model Map-Matching for Sparse and Noisy Trajectories

Hannes Koller, Peter Widhalm, Melitta Dragaschnig, Anita Graser
Austrian Institute of Technology
Giefinggasse 2
1210 Vienna
Email: first.last@ait.ac.at

***Abstract*—The problem of map-matching sparse and noisy GPS trajectories to road networks has gained increasing importance in recent years. A common state-of-the-art solution to this problem relies on a Hidden Markov Model (HMM) to identify the most plausible road sequence for a given trajectory. While this approach has been shown to work well on sparse and noisy data, the algorithm has a high computational complexity and becomes slow when working with large trajectories and extended search radii. We propose an optimization to the original approach which significantly reduces the number of state transitions that need to be evaluated in order to identify the correct solution. In experiments with publicly available benchmark data, the proposed optimization yields nearly identical map-matching results as the original algorithm, but reduces the algorithm runtime by up to 45%. We demonstrate that the effects of our optimization become more pronounced when dealing with larger problem spaces and indicate how our approach can be combined with other recent optimizations to further reduce the overall algorithm runtime.**

***Keywords*—Data Analysis, GPS Processing, Map-Matching, Hidden Markov Model, Optimization**

I. Introduction

With the advent of modern vehicles, smart phones and navigation systems, a rich data pool of GPS trajectories is gradually becoming available as an exciting new data source for intelligent transportation systems and related services. As GPS data is often prone to inaccuracies and measurement errors, the first step in trajectory analysis usually involves a map-matching algorithm that finds the most likely route for a given sequence of GPS measurements. In our work we focus on *off-line analysis*, where the entire GPS trajectory is known beforehand. In this context, map-matching can formally be defined as the problem of matching an entire GPS trajectory (a sequence of GPS positions $z_0, z_1, ..., z_n$) onto a road network graph. The output is a sequence of road links $r_0, r_1, ..., r_n$ which best corresponds to the input trajectory.

Existing map-matching approaches can be classified into three categories: *Geometric methods* such as [1] perform matching based on the geometrical properties of the GPS trajectory. *Topology-based methods* utilize additional information about the road network (such as link connectivity) to improve map-matching performance [2]. *Statistical methods* make use of probability models to identify the most likely road sequence for a GPS trajectory. Statistical approaches based on Hidden Markov Models (HMM) and related methods have successfully been applied to map-matching problems ([6], [7], [8], [9]). We base our contribution on the work of Newson and Krumm [3] which has been shown to work well on sparse and noisy data.

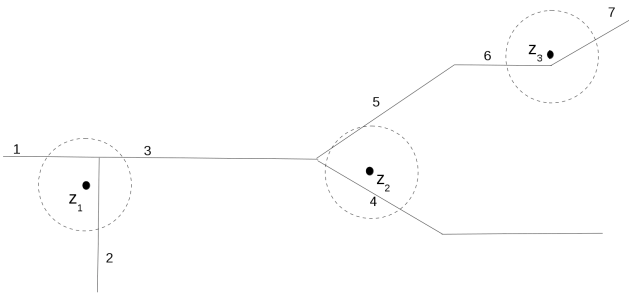

Fig. 1. An example problem. Position z_1 could be matched to the Links 1,2 or 3. Position z_2 could be matched to Links 4 or 5 and Position z_3 could be matched to the Links 6 or 7.

The Hidden Markov Model introduced in [3] attempts to balance the trade-off between (i) the probability of the candidate roads suggested by a single GPS measurement and (ii) the feasibility of the path between different candidate roads. The algorithm can be summarized as follows:

1) For each GPS position z_i, a set of candidate roads $\{r_i^0, r_i^1, ..., r_i^n\}$ is determined, i.e. a set of all roads within a certain search-radius, for example 200m, around z_i.
2) For each candidate road r_i^j, a *measurement probability* is calculated which reflects how likely it is for a vehicle on road r_i^j to emit a GPS measurement having position z_i. The probability depends on GPS error characteristics and decreases with increasing distance between road and GPS position.
3) For each candidate road r_i^j of a GPS position z_i, the *transition probability* to all candidate roads $\{r_{i+1}^0, r_{i+1}^1, ..., r_{i+1}^m\}$ of the next GPS position z_{i+1} is calculated.The transition probability is an exponential function of the difference between the route length and the great circle distance between z_i and z_{i+1}.The transition probability calculation therefore requires a shortest path routing between each pair of candidate roads, which is a computationally expensive operation.
4) The Viterbi dynamic programming algorithm [12] is applied to determine the map-matching solution by selecting the sequence of candidate roads which

978-1-4673-6596-3/15 $31.00 © 2015 IEEE
DOI 10.1109/ITSC.2015.411

yields the overall best explanation for the observed GPS trajectory.

As noted in [5], the *run-time bottleneck* of the approach are the shortest-path calculations that have to be performed between every combination of candidate roads of two consecutive GPS positions. Assuming there are n candidate roads for GPS position z_i and m candidate roads for GPS position z_{i+1} then nm shortest paths need to be calculated to obtain all transition probabilities between these two GPS positions. When dealing with long, noisy GPS trajectories where a large number of candidate roads must be considered in order to obtain good map-matching results, the number of shortest path routings becomes intractable.

Improving the run-time of [3] has already been the subject of several recent publications. In [4] multi-threading is used to parallelize the computation of measurement and transition probabilities. The authors of [5] adapt the transition probability calculation such, that the paths from a candidate road to all of its successors can be determined with a single execution of Dijkstra's algorithm [13], thus reducing the number of required shortest-path routings from nm to n.

In our work, we describe a different approach to improve the run-time performance for HMM-based map-matching algorithms. Our suggested method replaces the Viterbi algorithm with a Bidirectional Dijkstra algorithm [10] and employs lazy evaluation to reduce the costly calculation of transition probabilities which are not required to determine the optimal solution. Our contribution is orthogonal to the other optimization approaches, because we employ a different solving algorithm to find the solution. Our method decreases the number of transition probabilities which need to be evaluated in order to identify the optimal solution. The presented approach is however compatible with the other, previously mentioned optimizations and can help to further improve the algorithm performance.

The remainder of the paper is structured as follows: In section II we describe our optimization of the original algorithm. We show how we construct a search graph from the problem space of the HMM and describe how to transform the HMM probabilities into cost functions compatible with Dijkstra's algorithm. In section III we evaluate the quality and performance of our optimized algorithm using a publicly available benchmark dataset and compare our results to the results reported in [3]. Section IV concludes with a summary of our findings and indicates directions for further improvements of the presented algorithm.

II. Algorithm

As our algorithm is based on the approach of Newson and Krumm [3], the performed steps are similar. However, as opposed to the original method we do not apply the Viterbi algorithm to find the most likely sequence of candidates given the GPS positions $z_0, z_1, ..., z_n$. Instead, we find the solution by minimizing the *path costs* in a search graph using a bidirectional version of Dijkstra's algorithm [10]. This algorithm follows a *greedy* approach for finding a minimal cost path and evaluates the costs of a node and its outgoing edges only when it arrives at this node during search. This way the computation of transition costs, which involve computationally expensive routing operations, do not have to be performed immediately but can instead be delayed until they are actually needed. In most cases, the bidirectional Dijkstra algorithm only needs to visit a fraction of all nodes before the minimum cost path is found [11]. Thus, a large percentage of costly routing operations can be avoided. In real-world scenarios, where a trip can consist of thousands of GPS positions (and therefore millions of edges might be present in the graph), this optimization can lead to substantial performance improvements.

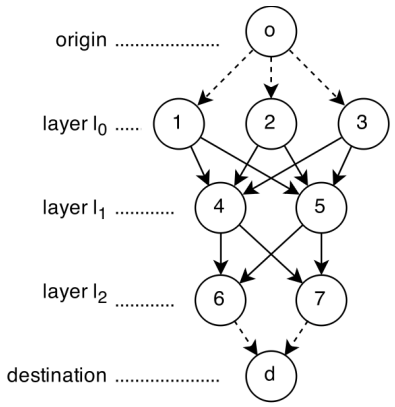

Fig. 2. Search graph for the example problem in Figure 1. Nodes in layer l_i correspond to the candidate roads of position z_i. Origin and destination nodes, and dashed lines complete the graph. Solid lines represent edges, where a vehicle routing operation is required to obtain the edge cost (transition probability).

A. Constructing a search graph

The search graph is constructed from an entire GPS trajectory. First we insert an origin node o. Then, for each position z_i, we insert a layer l_i in the search graph and identify a set of candidate roads (i.e. all roads in a search radius d around position z_i). For each candidate road r_i^j of position z_i, a corresponding node is created in layer l_i and assigned with a *cost* representing the distance between the GPS position z_i and candidate r_i^j. Next, a set of edges is created connecting all nodes of a layer l_i to all nodes of the next layer l_{i+1}. These edges are assigned with *costs* for a transition from candidate r_i^j in layer l_i to r_{i+1}^k in layer l_{i+1}, which are defined based on the route length and great circle distance between the GPS positions z_i and z_{i+1}. Note that the edge costs are not calculated when the search graph is constructed, the corresponding cost function is evaluated only when the costs are actually needed (lazy evaluation). Finally a destination node d is added and connected to all nodes of the previous layer.

An example is given in Fig. 1, which shows a small trip with 3 GPS positions $\{z_0, z_1, z_2\}$. The resulting search graph for this example problem is shown in Fig. 2: layer l_0, which represents position z_0, contains three nodes representing the roads $r_0^1 = 1$, $r_0^2 = 2$ and $r_0^3 = 3$. Similarly, layers l_1 and $l_2$contain nodes representing the candidate roads for position z_1 resp. z_2.

Note that the constructed search graph is analogous to the HMM decoding performed by the Viterbi Algorithm. The difference is that the measurement and transition probabilities

have been replaced by *costs* assigned to nodes and edges, respectively. In the original algorithm, the Viterbi Algorithm is used to find the map-matching solution by identifying the state sequence $R = (r_0, r_1, ..., r_n)$ which *maximizes* the joint probability $p(R, Z)$. Dijkstra's algorithm, in contrast, attempts to *minimize* the path cost. In the following section we therefore demonstrate how the measurement and transition probabilities can be transformed into *costs* in such a way, that the joint probability $p(R, Z)$ is expressed as a *path cost* that can be minimized with the Bidirectional Dijkstra algorithm.

B. Transforming probabilities into costs

A HMM models the joint distribution $p(R, Z)$ of a state sequence $R = (r_0, r_1, ..., r_n)$ given measurements $Z = (z_0, z_1, ..., z_n)$ as

$$p(R, Z) = p(z_0|r_0)p(r_0) \prod_{i=1..n} p(z_i|r_i)p(r_i|r_{i-1}), \quad (1)$$

where $p(z_i|r_i)$ are measurement probabilities (conditional probability distribution of measurements given states), $p(r_i|r_{i-1})$ are transition probabilities and $p(r_0)$ are initial state probabilities. In [3] the GPS errors are assumed to follow a zero-mean Gaussian distribution, and accordingly the measurement probabilities are defined as

$$p(z_i|r_i) = \frac{1}{\sqrt{2\pi}\sigma_z} e^{-0.5(\frac{\delta(z_i,x_i)}{\sigma_z})^2}, \quad (2)$$

where x_i is the point on the road segment r_i nearest to z_i, and $\delta(z_i, x_i)$ is the great circle distance between GPS measurement z_i and x_i. The transition probabilities were assumed to follow an exponential distribution

$$p(r_i|r_{i-1}) = \beta e^{-\beta|\delta(z_{i-1},z_i)-\phi(x_{i-1},x_i)|}. \quad (3)$$

Here $\phi(x_{i-1}, x_i)$ is the the driving distance along the shortest route from x_{i-1} to x_i, and β is a parameter chosen to best fit empirical data. The most likely state sequence - i.e. sequence of road segments R - is calculated by maximizing $p(R, Z)$, which is equivalent to minimizing

$$\begin{aligned} -\log p(R, Z) = &-\log p(z_0|r_0) - \log p(r_0) + \\ &\sum_{i=1..n} -\log p(z_i|r_i) - \log p(r_i|r_{i-1}), \end{aligned} \quad (4)$$

where

$$-\log p(z_i|r_i) = \log(\sqrt{2\pi}\sigma_z) + 0.5(\frac{\delta(z_i, x_i)}{\sigma_z})^2 \quad (5)$$

and

$$-\log p(r_i|r_{i-1}) = -\log(\beta) + \beta|\delta(z_{i-1}, z_i) - \phi(x_{i-1}, x_i)|. \quad (6)$$

When minimizing $-\log p(R, Z)$ all additive constants can be dropped as they are not relevant for the result. The initial state probabilities $p(r_0)$ are assumed to be uniformly distributed and can be dropped as well, and we minimize

$$\begin{aligned} C = &0.5(\frac{\delta(z_0, x_0)}{\sigma_z})^2 + \\ &\sum_{i=1..n} 0.5(\frac{\delta(z_i, x_i)}{\sigma_z})^2 + \beta|\delta(z_{i-1}, z_i) - \phi(x_{i-1}, x_i)| \end{aligned} \quad (7)$$

By constructing a search graph with node costs

$$n_i^k = 0.5(\frac{\delta(z_i, x_i^k)}{\sigma_z})^2 \quad (8)$$

for node k in layer i and edge costs

$$e_i^{jk} = \beta|\delta(z_{i-1}, z_i) - \phi(x_{i-1}^j, x_i^k)|. \quad (9)$$

for the edge connecting node j in layer $i-1$ to node k in layer i, we can use the Bidirectional Dijkstra algorithm to find a path (ie. a sequence $R = (r_0, r_1, ..., r_n)$) minimizing the path cost C and thereby maximizing $p(R, Z)$.

As Dijkstra's algorithm generally operates only on weighted edges, the node costs $n(v)$ have to be merged with the edge costs $e(u, v)$ when building the search graph. The edge weight $w(u, v)$ between graph nodes u and v is thus defined as $w(u, v) = e(u, v) + n(v)$.

C. Adapted cost functions

The previous subsection has shown that the probabilities of the original algorithm can be transformed into equivalent cost functions suitable for use with Dijkstra's algorithm. It is however interesting to note that in the original formalism the parameter β of equation 9, which controls the influence of transition probabilities versus measurement probabilities, is difficult to calibrate. This is apparent by the fact that the absolute magnitude of the transition cost (as defined in equation 9) is influenced by the length of the great circle distance between two GPS positions. As a result, costs for equally plausible routes will vary greatly, depending on the sampling interval and the speed at which the tracked vehicle was travelling. This makes it hard to define a β which consistently balances transition and observation costs, especially in real-life situations where the algorithm has to deal with incomplete trajectories and varying vehicle speeds. In order to allow for easier calibration, we have made adjustments to the cost functions. In our implementation we define the node costs as

$$n_i^k = \delta(z_i, x_i^k) \quad (10)$$

Edge costs for the edge connecting node j in layer $i-1$ to node k in layer i are based on the *ratio* (instead of the difference) between great circle distance and route distance

$$c_i^{jk} = \frac{\phi(x_{i-1}^j, x_i^k)}{\delta(z_{i-1}, z_i)} \quad (11)$$

and are defined as

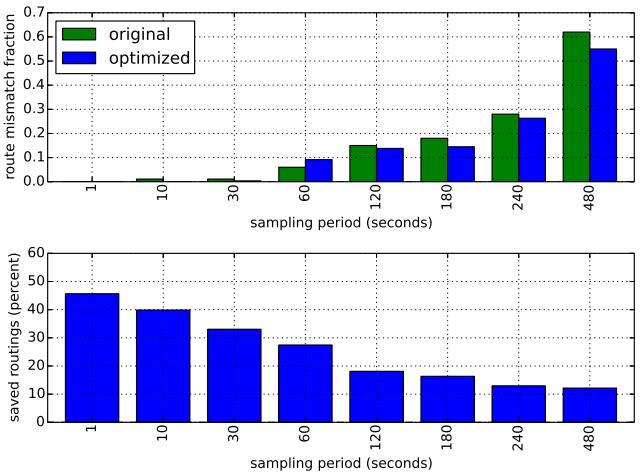

Fig. 3. *Top:* Comparison of route mismatch fraction between our optimized version (d=50) and the values reported for the original algorithm in [3]. As no exact numbers are given in [3], we use our best estimates based on the bar plot in Figure 7 of the article. *Bottom:* Percentage of routing operations which were avoided by our optimized approach

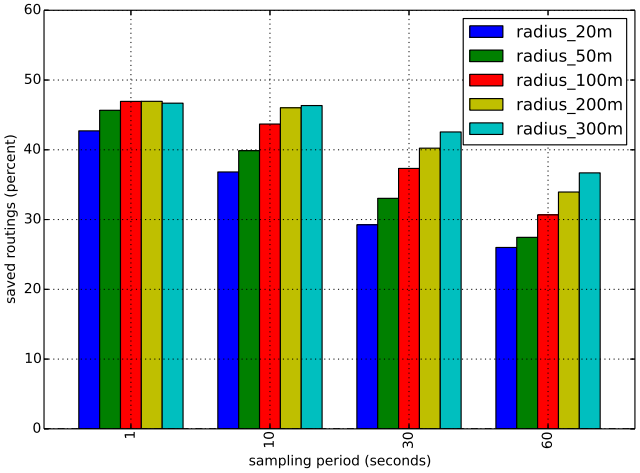

Fig. 4. Percentage of saved routings with different search radii and sampling intervals. When the search radius grows larger, a higher percentage of unnecessary routings can be avoided, especially when dealing with sparse trajectories.

$$e_i^{jk} = \begin{cases} \beta c_i^{jk} & \text{if } c_i^{jk} \leq t \\ \infty & \text{if } c_i^{jk} > t \end{cases} \tag{12}$$

where t is the maximal ratio between great circle distance and route distance which can be considered plausible. The calibration parameter β can be calculated intuitively as

$$\beta = \frac{\alpha}{1-\alpha}\frac{d}{t} \tag{13}$$

where α is a value between 0 and 1 which controls the tradeoff between transition and observation weights (higher α values put more weight on route plausibilty) and d is the search radius used to identify candidate roads around the observations z_i. Here, the role of d and t is to normalize the node and edge costs, respectively, in order to assure similar value ranges. In our experiments the following settings have been used: $\alpha = 0.65$, $t = 5.0$, and d was varied between 20 and 300 meters.

III. Experimental results

To evaluate our improvements, we use the benchmark data of [3] which was made publicly available. The data consists of 7531 GPS positions obtained during an 80km car trip in the Seattle area. The evaluation is based on a manually created ground truth, which was provided in the form of the correct link sequence travelled by the car. We have applied the same preprocessing steps to the GPS data as described in [3], where GPS positions with low confidence were removed from the data set.

A. Quality

We evaluate the quality of the results using the *route mismatched fraction* as proposed in [3]. The reported error is defined as

$$\frac{d_- + d_+}{d_0}$$

where d_0 is the length of the correct route, d_- is the length of all road links which were missed during map matching and d_+ is the length of all road links which were erroneously found during map matching.

Figure 3 (top) shows that for a one-second sampling period the algorithm is able to reconstruct the trajectory perfectly (route mismatch fraction is zero). This is identical with the results reported in [3]. With longer sampling periods the error of our implementation remains very similar to the results presented in [3]: Up to about 30 seconds, the route mismatch fraction is very low. With longer sampling periods, where more information about the original trajectory is lost, the performance of both algorithm versions degrades considerably (with a sampling period of 480 seconds only 17 of the 7531 positions are used as input).

B. Performance

Compared to the original algorithm, our solution maintains almost the same map-matching quality while at the same time requiring fewer routing operations to obtain the result. The map-matcher presented in [3] uses the Viterbi algorithm, which needs to evaluate all transition probabilities to find a solution. As the algorithm needs to evaluate all transition probabilities, it needs to perform 100% of the shortest path routings defined by the edges in the search graph. Our algorithm in contrast is able to avoid a substantial amount of these operations: Figure 3 (bottom) shows that for a sampling period of one second, our approach is able to avoid 45% of the routing operations (only 100703 of the 185333 edge weights needed to be evaluated). As the routing operation is the most expensive calculation step in the map-matching algorithm, this amounts to a substantial runtime improvement. With longer sampling periods the percentage of avoided routing operations decreases,

because the problem space has to be searched more thoroughly before the optimal solution can be identified. However with a 30 second sampling period (where the quality of the map-matching result is still very good) our optimization is still able to avoid 33% of the routings.

The quality of the map-matching algorithm is strongly dependent on the search radius d which is used to identify candidate roads around every GPS position. When dealing with sparse and noisy trajectories, a larger search radius typically improves the map-matching result because it increases the likelihood that the optimal road is included in the set of candidate roads. Larger search radii however also increase the size of the overall problem space: the number of nodes in each layer of the search tree increases and as a result the number of transition probabilities which need to be evaluated by the Viterbi algorithm grows larger. Figure 4 shows that the effects of our optimization become more pronounced under these conditions: with a search radius of 300m and a sampling interval of 1 second, the Viterbi algorithm would need to perform about 27.4 million routing operations. By contrast, our algorithm needed to perform only 14.6 million routing operations to determine the correct solution, thus avoiding over 46% of the operations. With increasing sampling interval the effect of the optimization on different search radii becomes more obvious: at a 60 second sampling interval our optimization can avoid 36% of the routing operations for the 300m search radius, but only 26% of the routing operations for the 20m search radius. The reason is that our approach evaluates more plausible paths through the search tree first and terminates once the best path has been identified. The problem space induced by the 300m search radius contains many state transitions related to possible, but unlikely paths. The evaluation of the related transition probabilities can often be avoided by our algorithm. This amounts to greater savings when the problem space includes a greater number of (possible but unlikely) candidate roads.

Based on our experiments, we therefore conclude that our optimization is especially advantageous when analyzing sparse and noisy trajectories, i.e. where an extended search radius is most useful.

IV. Conclusion and Outlook

In the last few years, map-matching has become increasingly important for evaluating road traffic and driving behaviour. While high quality map-matching algorithms exist, they can be slow when applied to large data sets. We have proposed an optimization to reduce the running time of a state-of-the-art map-matching algorithm which is based on a Hidden Markov Model [3]. We suggested an improvement which replaces the Viterbi algorithm with a Bidirectional Dijkstra and employs lazy evaluation to reduce the number of costly route calculations which are necessary to determine the optimal map-matching solution for a trajectory.

We have evaluated the quality and performance of our proposed method based on a publicly available data set. Our test results show that our suggested solution can avoid up to 45% of the costly routing operations and has no negative effect on the quality of the map-matching result. Future work will also include a detailed analysis of the effect of increased GPS noise and extreme maneuvers (such as sharp turnings) on run-time savings and map-matching quality.

Several other researchers have suggested different methods for optimizing the run time of the original map-matching algorithm [4] [5]. Since these suggestions optimize different aspects of the map-matching algorithm, they should be compatible with our approach. Future work will focus on further decreasing the algorithm run-time by combining our approach with these techniques. Another promising direction for further improvement of the optimization proposed in this paper is to replace the Bidirectional Dijkstra with other search algorithms such as the A^*-algorithm. This algorithm employs a heuristic to estimate the cost from the current node to the target node in the search graph, and with a suitable heuristic this could further reduce the average number of routing operations in many real-world scenarios.

V. Acknowledgements

This work was supported by the European Commission under TEAM, a large-scale integrated project part of the Seventh Framework Programme for research, technological development and demonstration [Grant Agreement NO.318621].

References

[1] Sotiris Brakatsoulas, Dieter Pfoser, Randall Salas, and Carola Wenk. On map-matching vehicle tracking data. In Proceedings of the 31st international conference on Very large data bases (VLDB '05). VLDB Endowment 853-864, 2005.

[2] N.R. Velaga, M.A. Quddus and A.L.Bristow Developing an enhanced weight-based topological map-matching algorithm for intelligent transport systems Transportation Research Part C: Emerging Technologies, 17 (6), pp.672-683, 2009

[3] Paul Newson and John Krumm. Hidden markov map matching through noise and sparseness. In *In Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 336–343, 2009.

[4] R.Song, W. Lu, W.Sun. Quick Map Matching Using Multi-Core CPUs. ACM SIGSPATIAL GIS, 2012

[5] Hong Wei, Yin Wang, George Foreman Fast viterbi mapmatching with tunable weight functions ACM SIGSPATIAL GIS, 2012

[6] P.Lamb, S.Thiebaux. Avoiding explicit mapmatching in vehicle location. In Proceedings of the 6th World Conference on Intelligent Transportation Systems, 1999

[7] B.Hummel. Map matching for vehicle guidance. In Dynamic and Mobile GIS: Investigating Space and Time. J.Drummond and R.Billen, Editors. CRC Press: Florida, 2006

[8] Yin Lou, Chengyang Zhang, Yu Zheng, Xing Xie, Wei Wang, and Yan Huang. Map-matching for low-sampling-rate GPS trajectories. In Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS '09). ACM, New York, NY, USA, 352-361. DOI=10.1145/1653771.1653820, 2009

[9] C. Y. Goh, J. Dauwels, N. Mitrovic, M. T. Asif, A. Oran, and P. Jaillet. Online map-matching based on hidden markov model for for real-time traffic sensing applications In Proceedings of the 15th International IEEE Conference on Intelligent Transportation Systems (ITSC),Anchorage, AK, DOI=10.1109/ITSC.2012.6338627, 2012

[10] Ira S. Pohl. Bi-directional search. *Machine Intelligence*, 6:127–140, 1971.

[11] T.A.J. Nicholson Finding the shortest route between two points in a network The Computer Journal, Vol. 9, Nr. 3,S. 275-280, 1966.

[12] A. Viterbi Error bounds for convolutional codes and an asymptotically optimum decoding algorithm IEEE Trans. Information Theory, vol. 13, no. 2, Apr. 1967, pp. 260-269.

[13] E. W. Dijkstra A note on two problems in connection with graphs Numer. Math., 1:269271, 1959