# Evolving map matching with markov decision processes

Adrian Wöltche\*

Institute of Information Systems at Hof University

October 12, 2021

#### Abstract

Map matching is about finding the best route of a given track in a road network. This can be useful for many statistical analyses on mobility. With increasing spread of modern cars and mobile devices many tracks are available to match. The difficulty in map matching lies in the geospatial differences between tracks and road networks. Current technologies resolve such differences with Hidden Markov Models and Viterbi algorithm. They majorly vary concerning the used metrics for the probabilities in the models. In this research we improve map matching technology by refining the underlying algorithms, models and metrics. We will introduce Markov Decision Processes with Value Iteration and Q-Learning to the map matching domain and we will compare them to Hidden Markov Models with Viterbi algorithm. Markov Decision Processes allow to use active decisions and rewards, which are not available in previous methods. Combined with improvements concerning the preparation of tracks and the road network, and various technologies for improved processing speed, we will show on a publicly available map matching data set that our approach has a higher overall performance compared to previous map matching technology. We will eventually discuss more possibilities we enable with our approach.

**Keywords** — map matching, geospatial information science, markov decision process, hidden markov models

# 1 Introduction

The world is full of devices (e.g., smartphones, GPS trackers, navigation systems, cars) that are able to track their geospatial positions. There are various reasons for this (e.g., navigation, way-tracking, geolocation). Albeit these devices record their geospatial location the real world location (the actual road network location) is often the main interest.

For mapping the geospatial locations recorded as tracks to the underlying road network map matching can be applied. The difficulty here lies in finding the best matching road edges. One intricacy is the inaccuracy of positioning systems and maps. Sometimes maps are build from aerial imagery, personal knowledge, or GPX traces. Of course, such maps are not guaranteed to be accurate but they still can serve as the underlying road network for map matching.

Most recorded tracks are created with the help of global *satellite based* geospatial positioning systems such as the Global Positioning System  $(GPS)^1$  [54], Galileo<sup>2</sup> [22], GLONASS<sup>3</sup>, or BeiDou<sup>4</sup>. Some geospatial positioning is also done with the help of regional systems such as the Indian IRNSS<sup>5</sup>, or the Japanese QZSS<sup>6</sup>. Sometimes Differential GPS is also used but most GPS tracks rely on smartphone sensors (e.g., Floating Car Data often comes from smartphone apps).

Concerning smartphones, *additional location sources and sensors*, such as wireless hotspot locations, cell provider tower locations, and internal device sensors (i.e., orientation, acceleration sensors, and more) are used for improving accuracy of positioning [16]. The internal sensor positioning often becomes worse

<sup>\*</sup>E-Mail: adrian.woeltche@iisys.de

<sup>&</sup>lt;sup>1</sup>https://www.gps.gov/

<sup>&</sup>lt;sup>2</sup>https://www.gsa.europa.eu/

<sup>&</sup>lt;sup>3</sup>https://www.glonass-iac.ru/en/

<sup>&</sup>lt;sup>4</sup>http://en.beidou.gov.cn/

 $<sup>^{5} \</sup>rm https://www.isro.gov.in/irnss-programme$ 

<sup>&</sup>lt;sup>6</sup>https://qzss.go.jp/en/

the longer no external ground truth sensor data is available. Relying on internal sensor positioning is called dead reckoning [10] and often it is combined with external positioning systems for minimizing the worst-case effects of both [e.g., 1, 62]. In case of smartphones the sampling interval also plays a role in the record accuracy [83] because the optimization of battery usage in todays mobile operating systems often reduces the sampling frequency.

The range we are confronted with measured positions in tracks is from several meters (e.g., GPS, Galileo) up to kilometers (e.g., cell provider station tracking, see [39]). There is technology to improve the geolocation error alone (e.g., Kalman filters, see [13], actor-critic algorithms, see [86]) but the differences between roads and tracks remains, which is why map matching technology is needed.

## 1.1 Map matching

The key idea of map matching is that a measured track lies within the underlying road network. Of course the assumption is that the map is correct, which is not guaranteed. This is why exceptions to this assumption are made in map generation algorithms. In this case we can distinguish between algorithms that are based on individual tracks [e.g., 56, 12, 69, 52], often combined with regular map matching and others relying on statistical aggregation of many tracks, i.e., from Floating Car Data (FCD) [e.g., 43, 68, 67, 88, 90]. These algorithms introduce new roads in case the underlying road network has no sufficient matching solution to the given tracks. As it is not possible to verify whether the road network is in fact sufficient or not, even fully integrated road networks are altered by such algorithms [see 12]. This makes it difficult to blindly trust map generation algorithms though they are currently the only known way to find good matches when the road network is wrong.

Regular map matching that is trusting the correctness of the map can be divided into online and offline map matching. Online map matching occurs while the record of a track is still running (e.g., sliding window from [27, 44, 66]). With using the latest history of recorded positions, this can be used for live navigation or tracking.

Offline map matching is the extension so that the entire record is used. In comparison to live navigation there is no goal that is approximated by the best route towards it [see 28, 51]. Instead the first and last point of the track mark the source and destination. Moreover, not only driven tracks but also walked [7], wheelchaired [60], bicycled [61], swum [6], or flown [17] tracks are matched sometimes. This depends on the tuning of the probabilities and the road network excerpt being used.

The key aspect in optimizing map matching is the definition of 'typical road behavior'. The more it reflects the reality the more realistic and probable the matched routes fit to the given tracks [74].

### 1.2 Current state

Current offline map matching technologies that are state-of-the-art are e.g., Barefoot<sup>7</sup> [47], Fast map matching<sup>8</sup> [79], Map Matching based on GraphHopper<sup>9</sup>, Open Source Routing Machine<sup>10</sup>, Valhalla<sup>11</sup>, and pgMatch<sup>12</sup> [50].

Common in all these established projects is that they are derived from Hidden Markov Models (HMMs) solved with Viterbi algorithm [25] as described in [53], which itself relies on previous work in [35] and [33]. There are sometimes additional map matching algorithms available in these tools. For example in 'Fast map matching' [79] the ST-Match [46] algorithm also exists. Nevertheless, most established tools primarily or solely rely on HMMs and Viterbi algorithms as they mark the state-of-the-art in map matching.

Current research in the field of map matching merges from two complementary main views of the problem. One view is about optimizing processing speed [e.g., 49, 46, 70, 5, 85, 79, 24, 26], the other one is about optimizing accuracy of the output of map matching algorithms [e.g., 33, 35, 57, 53, 30, 74, 81, 82, 7, 48, 40, 15, 31]. More accurate results are usually enabled by using technologies with a higher time complexity. Often, this goes hand-in-hand with technologies for improving the processing speed so that the incremented complexity has less impact on the processing time.

Some processing speed optimizations use caching mechanisms and partial route precomputation [79], others filter unnecessary input data [e.g., 35, 53]. Different ideas, such as ST-Match [46], use less complex

<sup>&</sup>lt;sup>7</sup>https://github.com/bmwcarit/barefoot

<sup>&</sup>lt;sup>8</sup>https://github.com/cyang-kth/fmm

<sup>&</sup>lt;sup>9</sup>https://github.com/graphhopper/map-matching

 $<sup>^{10}</sup>$  https://github.com/Project-OSRM/osrm-backend

<sup>&</sup>lt;sup>11</sup>https://github.com/valhalla/valhalla

 $<sup>^{12} \</sup>rm https://github.com/amillb/pgMapMatch$ 

models for reducing time complexity. Such models are sometimes again improved, for example, in [84] an Interactive-Voting Based Map Matching Algorithm (IVMM) is introduced for improving over ST-Match. Additionally, there are ideas for using parallelization and distribution with cluster computing [e.g., 85, 26] or Graphical Processing Units [GPUs, see 5].

Accuracy improvement mainly depends on the used algorithms and metrics. Early research worked on simple geometric point-to-curve and curve-to-curve matching [e.g., 9, 76]. More recent work used Discrete Fréchet Distance [21] and Free Space Diagrams [3] for fast, up to subquadratic time [2] geometric curve-to-curve matching [11].

Recent research however majorly relies on global max-weight optimization [see 46, 74]. Such statistical approaches can easily deal with outliers by assigning low probabilities to them [53]. Still, hybrid approaches [74] are able to improve further the 'weight function' metrics over solely statistical ones [75].

A recurring conclusion of map matching literature is the lack of accuracy in map matching algorithms, e.g., finding the correct road position especially in complex road scenarios. This was concluded for example in [59] which is an older survey literature before HMMs with their higher matching accuracy became common with [53]. Even today after many more advances in matching accuracy, recent map matching surveys [e.g., 14, 32] still state that map matching performance needs to further improve, concerning e.g., unnecessary detours, matching uncertainty, matching breaks, geometric and topological principles, candidate search based on current vehicle situation, and distributed computing. We see that both processing speed as well as accuracy improvement are still of huge interest in current and future research.

An interesting research direction to HMMs was introduced by [78] based on [45] by empowering Conditional Random Fields (CRFs) for map matching. The idea for using CRFs instead of HMMs comes from [38] which describes the so-called 'label bias' as being a statistical drawback when per-state normalization is used in non-generative models (i.e., when conditional probabilities instead of joint probabilities are used). The authors show that the 'label bias' happens in Maximum-Entropy Markov Models (MEMMs) but clarify that HMMs are not affected because they are generative models with joint probabilities (which is also confirmed in [18] and [29]). Also CRFs don't suffer from 'label bias' and in their comparison of CRFs with MEMMs and HMMs, the authors show that CRFs have a significantly higher accuracy but are affected by a huge processing speed impact compared to HMMs. This research direction seems to have no further current works in it.

One promising research direction left is using reinforcement learning algorithms for map matching. In [89] Inverse Reinforcement Learning (IRL) is used on a Markov Decision Process (MDP) that is modeled after the nodes and edges of the road network with recorded tracks of taxis for learning and predicting route recommendations. This approach is further refined in [63]. Inverse Reinforcement Learning uses expert demonstrations for learning the reward function. This allows the reinforcement learning agent to find optimal policies on similar but new input data. In [23], Markov Decision Processes are combined with multi-agent MDPs for predicting airplane trajectories and demand-capacity balancing. Reinforcement learning can also be used with Actor Critic models as in [86] for reducing the noise on raw GPS tracks. With such sanitized tracks existing map matching technologies can produce more accurate results.

Finally, in [64], History Markov Decision Processes with Q-Learning are used for low-sample map matching. Here the authors first use filtered (outliers removed) high-sample taxi tracks with a time step of 30 s with the incremental map matching algorithm from [11] to generate a label database (history tracks). The high-sample tracks are then thinned out to low-sample tracks with time steps of 1 min, 2 min, 4 min, and 16 min. The authors create a MDP with the state being a candidate position of a GPS measurement and the action being the selection of the next candidate position. The reward is calculated as weighted sum of the percentage of available history tracks between both candidate points (or else the shortest path if no history tracks were found) from all history tracks and the weighted distance between GPS measurement and candidate of the current state. The weights were computed by trying out all values from  $0.1, 0.2, \ldots, 0.9$ . The low-sample tracks are matched by applying Q-Learning on the constructed MDP. Finally the results of the low-sample reinforcement learning match are compared with the results of the filtered high-sample incremental match. This comparison is further compared with IVMM from [84]. A major flaw we see in this approach is that the authors take the results from an incremental matching algorithm as ground truth data for training the reinforcement learning algorithm. This way, it is impossible for the reinforcement learning algorithm to improve over what the incremental algorithm matched.

Still, with the availability of huge amounts of FCD without ground truth routes, reinforcement learning is a very promising technology because reinforcement learning intrinsically has the ability to learn from its own experience instead of labels computed from preceded algorithms. Nevertheless, the reviewed research of route predictions with IRL and MDPs that are based on the results of expert demonstrations or incremental methods is a great way to enhance advanced map matching algorithm results as long as such labels are available or can be generated truthfully.

Our new approach with MDPs and reinforcement learning will be able to find solutions with no existing ground truth data available. We will introduce an evolved MDP that gives a learning agent the possibility to learn an optimal policy without any prior knowledge, similar to the map matching technologies that work on HMMs.

## 1.3 Approach

As [74] demonstrates the superiority of hybrid global max-weight approaches we adhere to this idea. We implement Markov Decision Processes [MDPs, see 8] for replacing Hidden Markov Models to enable an active action selection during the map matching process. This combines the advantages of iterative point-to-curve, and global max-weight algorithms for improving overall performance. Still it extends the tuning possibilities for action selection, state and reward function design. The mixed rewards in MDPs furthermore facilitate the 'weight function design', which has a major impact on map matching accuracy [75]. Furthermore, we expand the usage of reinforcement learning algorithms on map matching, which enables more potential to improve poor accuracy map matching results with reusing previously learned knowledge. We want our algorithms to learn and apply what we call 'typical road behavior'. This behavior is intrinsically contained in FCD but not directly applicable without the further statistical analysis that learning algorithms permit. For this research we will stick to a complex, parametrable reward function defining our current understanding of 'typical road behavior'. Further research may fine tune the weights of the parameters and implement more conditions for improved resemblance to 'typical road behavior'.



Figure 1: Technology Roadmap for our Q-Learning map matching approach. First we import the data of the road network and floating car data we want to process. We build a bidirectional graph from the road network and generate a spatial index (R-Tree) on top of the edges of the graph. When we want to find a match for a selected track, our reinforcement learning algorithm first initializes the environment of the Markov Decision Process (MDP). This executes a candidate search of the selected track with the help of the spatial index. The reinforcement learning algorithm is then able to send actions to the environment. An action selects a next position from a current one. For each action a route (shortest path) is searched within the graph of the road network. The found route is then evaluated with the current action and the reward is given back to the reinforcement learning algorithm. After an optimal policy has been found the matching result for the selected track can be computed and returned.

For our evolved MDP for the reinforcement learning algorithms we will compare Value Iteration [55, 65] with Q-Learning [72, 65]. We will also compare with HMMs and Viterbi algorithm that we implemented with similar metrics to our MDP. We will see that Value Iteration and Viterbi algorithm are able to find theoretical optimal policies and that Q-Learning is still able to find good, up to optimal [73], policies. We will additionally compare our results with the Fast map matching [79] algorithm which is based on HMMs and Viterbi algorithm. Moreover, the Fast map matching open source tool<sup>13</sup> also contains the less complex map matching model ST-Match [46] for large area networks, which makes it interesting to compare to this model as well.

<sup>&</sup>lt;sup>13</sup>https://github.com/cyang-kth/fmm

# 2 Model

Our evolved model is based on Markov Decision Processes [MDPs, see 8] which relies on Markov chains the same way Hidden Markov Models (HMMs) do. The differences are that in HMMs observation and transition probabilities are jointly calculated, whereas in MDPs rewards are given for taking actions in states. Another difference is that HMMs are only partially observable, whereas MDPs are fully controllable by the agent.

When calculating the probabilities or rewards existing data from the recorded track and given road network can be used. There are for example distances and routes [53], bearings [58], trajectories [40], and velocity and timings [46] that can be computed. Typically, routes in the road network are calculated with Dijkstra's shortest path algorithm [19]. From the computed metrics either the probabilities for HMMs are computed or the rewards for the selected actions in a MDP are returned.

## 2.1 Markov decision processes

The idea for the Markov Decision Process (MDP) we use in this research is deduced from the way navigation happens. A navigator selects, depending on its current state, a next road to transition to for eventually reaching its goal with the lowest effort. Our MDP is designed specifically after this approach, as seen in Figure 2.



Figure 2: Markov Decision Process for map matching. Here we present in red a possible state  $s(p, c, p') \in S$ with current measured point  $p = P_n$  with  $p \in P$ , current candidate  $c = C_n^m$  with  $c \in C$ , and next measured point  $p' = P_{n+1}$  with  $p' \in P$ . We also present in green the possible actions  $A_s$  that allow to choose the next candidate  $c' = C_{n+1}^m$  with  $c' \in C$ . So a state s is composed of the current road position (given by p and c) in time step  $t \in T$  and the next measurement p' in time step  $t + 1 \in T$ . With this state a road connection to position c' can be found by choosing an  $a \in A_s$ . With this state representation the reward (violet lines plus chosen green action) can be computed from the distances  $d_{c,p} = |c - p|$  and  $d'_{c',p'}$ between the points p, p' and corresponding candidates c, c', the distance (and other metrics e.g., bearing, vehicle speed, timings) between the points  $d_{p,p'}$ , and the length of the route  $r_{c,c'} = \text{shortest}\_\text{path}(c, c')$ between the candidates  $d_r = |r|$ . This representation is not necessarily bound to the  $t \in T$  being adjacent. It is also possible to skip certain  $t \in T$  so that a state  $s \in S$  with  $A_s$  is composed from the measurement points and candidates of non-adjacent  $t \in T$ , for example  $t_2$  and  $t_5$ .

Concerning the complexity of this model we can compute the amount of possible end-to-end sequences with Equation 1. This shows that the number of possible routes may grow exponentially with the number of points. As such, we use track sanitation and network simplification algorithms for reducing the amount of candidates by removing unnecessary information.

$$n_{seq} = \prod_{i=0}^{t-1} |C_{t_i}| \tag{1}$$

Our state representation moves step by step from road position (measurement point and selected candidate) to road position when taking an action  $A_s$  in  $s \in S$ . This means that we add the selected next road position to the state and remove the current road position for creating the new state  $s' \in S$ . As this enables to concatenate locally optimal succeeding shortest paths for all adjacent candidate pairs (and non-adjacent if skipped time steps were introduced) we are able to eventually reach the global optimum shortest path by applying reinforcement learning algorithms (which itself make use of the Bellman optimality equation, see [65]).

At the first time step the sliding window has gradually to go-in because we already need a reward for choosing the first candidate. So the first states s(p) at  $t_0$  consist only of one position. The states at  $t_1$ then consist of s(p, c, p') as seen in 'Figure 2. Equation 2 does not value this for simplicity reasons and calculates the amount of states  $n_S$  without gradual go-in. We can also have states with actions leading into dead-ends before the final time step was reached. In our MDP we can skip such actions, which means that the actual amount of states  $n_S$  actually can be higher or lower as stated in Equation 2. Still, we can use this calculation later for estimating rough stopping conditions for our training process with Q-Learning.

$$n_S \approx \sum_{i=0}^{n-2} \prod_{j=0}^{1} |S_{i+j}|$$
(2)

### 2.2 Actions and rewards

In our MDPs we use actions for transitioning between succeeding states. An *action* is a choice of how to transition from one state  $s \in S$  into the next state  $s' \in S$ . For each action  $a \in A_s$  followed at time step t in s, a *mixed reward*  $R_{t+1} = R(s, a)$  is returned. We define our reward roughly by the following formula:

$$R(s,a) = -(w_D * |\sum Distance_{Candidates,Points}|$$

$$+w_L * |\sum Length_{Points} - \sum Length_{Routes}|$$

$$+w_B * |\sum BearingDifference_{Points,Routes}|$$

$$+w_A * |\sum AbsoluteDirectionChanges_{Routes}|)$$

$$(3)$$

 $\sum Distance_{Candidates,Points}$  is the sum of the distances (geographical or cartesian, depending on spatial reference system) between each point and assigned candidate of a state.  $\sum Length_{Points}$  is the sum of the distances (or lengths of the lines) between all succeeding points of a state.  $\sum Length_{Routes}$  is the sum of the lengths of the routes (the shortest paths in the road network graph) between all succeeding candidate pairs of a state. The length difference is further multiplied with a route factor that depends on the fraction of both lengths for prioritizing equally long routes. This is not addressed in the formula but can be reviewed in the code. The route factor is also used for dynamic skipping locations when the route is about 10 times larger than the distance between the succeeding points. A similar approach was used in [53] for skipping high error situations.  $\sum BearingDifference_{Points,Routes}$  is the sum of the bearing differences between the bearings of the lines between succeeding points of a state and the average bearings of the corresponding routes. A single bearing difference is always within the range of  $-180^{\circ} < 180^{\circ}$  with negative angles which are angles to the right and positive angles which are angles to the left.  $\sum AbsoluteDirectionChanges_{Routes}$ are the summed absolute direction changes between all succeeding segments of the corresponding routes. Finally, our reward formula has weights for all parts of the mixed reward. We use a default of 1.0 for the distance and absolute direction changes  $w_D$  and  $w_A$  as we empirically found out that they have the most impact for 'typical road behavior'. Still we use 0.1 for the length and bearing differences  $w_L$  and  $w_B$  for making sure that the right direction and length differences are still valued. The fine tuning of these weights can be subject of subsequent research.

We define our reward to be negative for giving punishment for large differences between measured points and road edge candidates. This eventually leads to optimizing against the highest sum of rewards close to negative zero. The summation within each mixed reward is possible because great errors bias single rewards very heavily, whereas the best overall balance is what we seek. Of course this metric can be further tuned with other specifics [see 74]. This can be subject for subsequent research.

Additionally, although our reward  $R_{t+1} = R(s, a)$  always depends on the state  $s \in S$  and the action  $a \in A_s$  taken, it is also possible to use other data that is static during the whole MDP run. It is only important for R being deterministic for every A in every S or else it would not be possible to find an optimal policy with Value Iteration or Q-Learning. For example, we use an accumulated reward of all

remaining point distances in situations where we fail to find a route and have not reached the end of the track yet. This error comes not directly from the state but is static for a specific state during the MDP run and it helps the reinforcement learning algorithm to omit high error situations.

For our HMM the probabilities are similar to our MDP reward. Instead of using absolute values, we need to calculate probabilities (so relative fractions) for the Viterbi algorithm to work on our HMM. The distances are fractions of the current candidate distance to the longest candidate distance of a point. They are the observation probabilities. The lengths are fractions to each other, similar to the route factor above. The bearing differences are relative to 180°. The absolute direction changes are relative to 360°. If a route is around 10 times longer than the distance between the points, its probability is set to 0.01 as no dynamic skips can be used in a static HMM. The Viterbi algorithm needs a precomputed HMM compared to Value Iteration and Q-Learning that compute the state-action pairs of the MDP during their runs. The HMM transition probabilities are also converted to log-probabilities for faster calculation with the Viterbi algorithm.

# 3 Algorithms

After building our models we use an optimized version of Value Iteration [55, 65] which is a dynamic programming algorithm comparable to Viterbi algorithm [25] for solving our MDPs. We compare with Q-Learning [72, 65] which is a temporal-difference reinforcement learning algorithm that improves in an incremental manner. We use an optimized  $\epsilon$ -greedy variant that has some additional conditions for action selection and training stopping that are beyond the scope of this paper but can be reviewed in our code. We also compare with Viterbi algorithm that we implemented on a HMM with the probability metrics described above. Both Value Iteration and Viterbi algorithm share a common idea in finding an optimal global solution in the underlying model by combining locally optimal solutions with globally optimal sequences over an iteratively computed memory. The idea is to combine the current (state) error with the next best (state) error for eventually finding the smallest global (state sequence) error. The Viterbi algorithm is a little bit faster in this case because, as the HMM is a static model precomputed before the algorithm is run, the algorithm can grow the error backwards in one run, so the 'next' error is already computed before being applied to the current error. Value Iteration however is iteratively improved until a threshold is reached, when the global error does not change any further. This is because MDPs have a 'direction' when an action is selected in a specific state and because they are computed on the fly, they cannot be run backwards like a HMM since the rewards simply don't exist when the algorithm starts. Therefore the 'next' error is in fact the error in the next state from the previous episode. When that error does not exist yet, for example in the first episode, an error of zero is assumed. This does not matter as the training continues until the global error does not change any further. Eventually, the global error also grows backwards after many episodes, which stops the process then when the threshold is reached. With respect to this backwards error growing, we swipe through our state space in a similar backwards order, so we calculate the errors from the states in the later track before the errors of the states in the track before. Due to the update formula in Value Iteration computing the value of a state with respect to the value of the succeeding state, this allows to reduce the amount of iterations needed for Value Iteration to converge, which makes our algorithm comparably fast to the Viterbi algorithm. More about this optimization can be reviewed in our code. Still, we see that depending on the model Viterbi algorithm is more an optimization algorithm whereas Value Iteration is more a learning algorithm similar to a real learning algorithm like Q-Learning which also improves incrementally with many episodes.

## 3.1 Preparations and candidate search

Before we can successfully match a track, previous research suggests filtering the track from improbable and unnecessary points. In our case we first remove all spatially duplicate points because such points contain no additional information about the movement of the underlying tracking device. Then we employ Douglas-Peucker algorithm [20] with a setting of 5.0 m (distance around the line) for removing additional points that contain no significant additional information about the movement. This retains curves and direction changes but removes most points within straight lines. For our map matching algorithms, it makes no difference concerning the errors if many small straight segments or a large straight segment are compared, the errors after Equation 3 are similar. However, it makes a difference concerning the processing speed and amount of candidates found in candidate search, so removing unnecessary points is highly recommended.

After that first step, we further reduce the amount of measured positions by using a similar filtering circle as suggested in [53] for sanitizing noisy track parts. There they use a circle of  $r = 2\sigma_z = 2 * 4.07 \,\mathrm{m} =$ 8.14 m because this is the GPS noise they extracted from their measured data. For our work, we use a filtering circle with a rounded fixed radius of 10.0 m as we did not extract any GPS noise from the data. Within that circle we retain the median point from the selected points. This is different to [53] when the circle was only used to remove points around an existing point, in our case we further remove outliers by retaining only the median point. For this it is important to only select points that are directly connected within the filtering circles' radius, in order to avoid removing points that accidentally cross the circle from later parts of the track. The idea of sanitizing noisy track parts is that stochastic models are easily distracted by noise and outliers, so the less noise the easier for a stochastic process to find a true optimal solution. The latter sanitation step naturally makes it more difficult to rely on timestamps of the GPS positions in the used metrics. This is the reason why we omit timestamps completely in our metrics. Not relying on GPS timestamps also makes it possible to match tracks that are given as geographic lines without any time information (for example WKT strings). Only relying on the geographical positions in fact makes our algorithms wider applicable compared to algorithms that rely on timing information in their metrics.

Alternatively to our sanitation approach, other 'heuristic' filtering circle radii, dynamic radii or more advanced trajectory simplification algorithms [e.g., 41, 42, 82] can be applied. The combination and benchmark of alternative sanitation technologies with our approach can be subject of subsequent research.

With the sanitized track we now have all points of P for searchinig possible candidates. A candidate is a road edge that might have been actual part of the track followed in reality. The process of finding these candidates is called *candidate search* [e.g., 35, 53, 27, 75, 15, 79, 24] and is usually also done by applying a search circle around each point, or by using k-nearest search, or by using both. Without candidate search we had to consider all roads of the entire road network for every measured point. Obviously, this would be too much to compute and it is also not reasonable due to GPS accuracy fluctuating only within specific ranges [see 77, 54].

For reducing unnecessary candidates our road network is simplified before candidate search is applied. We simplify by removing all nodes that can be removed without altering the network integrity or shape (lossless simplification). The specific conditions are beyond the scope of this paper but can be reviewed in our code. For example curves can be represented by small segments each having a start and end node in the road network graph. These segments can be concatenated into a single line with its start and end node being the next adjacent crossroad or T-junction. After this lossless simplification, we have less edges and nodes in the road network graph with the edges being more complex lines instead of simple segments. As such, our candidate search finds less edges and our shortest paths algorithm needs to consider fewer nodes. This simplification step improves the processing speed significantly.

Considering the radius of the search circle, if we have GPS points, we can for example use the findings from the GPS Performance Analysis Report [see 77, Table 7-2], which states a maximum horizontal error of various international GPS sites of 27.98 m in 99.99% of cases. This is obviously true for GPS devices but for example smartphones sometimes have larger errors due to battery optimization. It is also possible to derivate the radius from search radii used in previous research, e.g., 15 m [24], 30 m [75], 50 m [27], 200 m [35, 53], or 100 m, 200 m, and 300 m [79]. Moreover, it is also possible to use dynamic radii [15] depending on the distances between succeeding measurement points. In a MDP as we suggest it, it is also possible to expand the search radius at run-time when specific conditions are fulfilled, such as huge length differences or exceptional bad rewards for all  $s \in S$  at  $t \in T$  in comparison to other time steps. This dramatically increases processing times, as the already converged algorithms then need to find a new global optimum. Therefore we disabled this feature, it can be the topic of subsequent research. For this work, we make only one candidate search and we use a dynamic radius approach with a minimum radius of 200 m. If no points are found within that radius, we double the radius until we find points or the limit of 10000 m is reached, in which case we skip the point. In our tests, this never happened. The radius is also increased up to the minimum half distance between succeeding points, for example when three points each lie 500 m next to each other, the radius around the points is increased to 250 m so that in low sample situations our map matching algorithm has more possibilities to find the best route. The minimum radius makes sure that in dense areas with points very near to each other, no probably important candidates are left, especially when the points have a larger noise or are farther apart from the roads, e.g., when alternative side roads are taken that are not part of the underlying road network.

In our MDP each candidate is an edge intersecting our search radius around a point. Each nearest point for every candidate edge can be perpendicular to the edge or can be an end point of it. After we found all these candidates for each point of our track, we use an additional processing that we call candidate adoption. In the first step, we exchange all candidates between succeeding points, so each point adopts the candidates from its previous and next point, except the ones that were already within its own list. This leads to each road position being mapped to at least two points (in the start and end of the track) up to three points for all remaining points inbetween. Next, we make an additional index based search around each point with the radius of the farthest candidate distance and adopt all candidates that we found in this circle not being already in our list. In dense areas with many points remaining, even after our track sanitation, this adopts candidates from points further apart and allows for more possibilities to navigate within difficult dense situations. Our candidate space C grows very large after these steps but it allows our optimization algorithms to choose from a lot of possibilities. In our tests, we found out that the candidate adoption dramatically improves matching accuracy. Many noisy points can then point to the same candidate position which effectively eliminates round travels and forward-backward (zig-zag) routes. Further investigation of how these methods impact the map matching process can be subject of subsequent research as our code base allows for additional configuration of these processing steps.

#### **3.2** Performance adjustments

Our code contains more adjustments that specifically improve the processing speed, which in turn allowed us to increase our model complexity, for example by using candidate adoption.

Concerning the network, we use a custom adjacency list graph with fat-initialized and simplified edges. During the import and simplification, we use lists for fast modification. We then convert our graph into vectors for fast and constant time access during Dijkstra (routing) algorithm. Our edges contain all segments, segment lengths, segment bearings and the accumulated length, edge bearing, and direction changes, as well as absolute direction changes. This is also true for our imported tracks before and after sanitation.

The fat-initialization of the tracks is used for fast track sanitation with a custom Douglas-Peucker [20] algorithm, for fast map matching reward calculation, and for fast line comparison for error calculation between matching results and ground truth routes. Concerning the edges, the fat-initialization is also used for fast subline extraction during candidate search.

For each candidate search, we use an R-tree on the segments of the network with each segment pointing to its edge. This allows for fast retrieval of references to candidate edges. We then calculate the closest point to each edge and extract two sublines from start to closest point and from closest point to end. These sublines are saved with each candidate. We also save all nodes of all edges with each candidate.

When we want to compute a route from one candidate position to the next candidate position, we start a single-source Dijkstra algorithm on the target node of the edge of the current candidate to all nodes of the next candidate. When all nodes were visited, meaning we have the shortest paths to all given nodes, we early-stop the Dijkstra algorithm with a custom visitor as we have no need to search the entire graph from that point on. All the results are saved in a hash map accessible by ids of the nodes. This way we can retrieve the other found results without the need to run the Dijkstra algorithm again during the following iterations when we want to reward the alternative routes from the source candidate to the target nodes of the next candidate. This saves a lot of Dijkstra algorithm queries which is known to have a huge impact in map matching performance. For example in Fast map matching [79], for reducing the impact of the Dijkstra algorithm, a precomputed routing table (UBODT) is created before the map matching is applied. Our approach does not precompute an UBODT for the whole graph but dynamically precomputes and caches only the candidate pair routes presumably needed for matching a specific track. When multiple tracks are matched on the same graph, this route cache speeds up following tracks if they lie within the same bounds of the previous tracks.

The routes we receive from the Dijkstra algorithm only contain references to the fat edges of the network graph. Since we can only route between nodes of the graph, but eventually we want routes between two candidate positions, we have to combine the routes of the graph with the precomputed subline extracts of the candidates. For this, we can simply concatenate the references of the source subline, the route between the target of the source subline and the source of the target subline, and the target subline. Then we only have to fat-initialize the new length and directions from the already fat-initialized references but we don't need to copy any fat objects or recalculate any lengths or inner directions, only the directions between the three lines. One exception is the bearing for the whole line which always needs to be recalculated, but we introduced a cache so that we can retrieve bearings between any two points that were already calculated before. We do the same with lengths between any two points, just in case, because when we skip points in our MDP, new lengths might need to be calculated between points that were not directly connected before. Our code is able to calculate the lengths and angles in geographic

spatial reference systems (SRSs) as well as in cartesian SRSs. As the calculations in a geographic SRS rely on computationally heavy calculations, in our case Andoyer [4], but also the more expensive Vincenty [71] and Karney [34] are available, the caching and fat-initialization saves a lot of computational resources when accessing the same distances and bearings multiple times. This also makes our map matching in the geographic SRS much faster in comparison to the cartesian SRS, because expensive bearing and distance calculations are minimized by caches, fat-initializations and references. With the computed fat-initialized route, calculating the reward is very fast as no new geographical calculations need to be done.

For our MDP und HMM models, we use index mapping on vectors instead of hash maps for faster access of rewards from previous episodes for specific state-action pairs and we use vectors for transition and observation probabilities. Both vectors and hash maps have an access time of O(1) but index mapping was about eight times faster in our tests because no hash needs to be calculcated for finding a position, a position can directly be accessed by the already available index. Our code base still contains both algorithms with hash map and with index mapping, which can be interesting for reviewing, but in our benchmarks we only use the much faster index mapping technology.

## 4 Results and analysis

We validate our approach with the map matching data set [36] from [37] available under CC BY-SA 4.0<sup>14</sup> at https://zenodo.org/record/57731. This data set specifically created for benchmarking map matching algorithms consists of 100 tracks from the OpenStreetMap Planet GPX dataset<sup>15</sup> available under CC BY-SA 2.0<sup>16</sup>, 100 corresponding road network extracts from OpenStreetMap Planet OSM<sup>17</sup> available under ODbL<sup>18</sup> from 100 different locations around the world, and hand corrected matches for each track. By matching the given tracks against the given road network with our algorithms, we can compare our matching results to the provided hand matches and calculate the matching error.

The data set is given in the GPS spatial reference system WGS 84 (EPSG:4326). For our implementations, we generally stay within this SRS. For the FMM algorithms, we convert the data into the WGS 84 / Pseudo-Mercator (EPSG:3857) SRS. WGS 84 / Pseudo-Mercator is a reprojected (cartesian) SRS used by Google Maps, OpenStreetMap, and others. We need to use a reprojected SRS with FMM because it internally uses the much faster cartesian distance calculations and does not support the more precise geographical distance calculations as our algorithms do. For the sake of comparison, we also benchmarked the data set in the reprojected SRS with Value Iteration. We will later see that the error is similar good. Still, exact error difference examinations can be subject of subsequent research, our code allows for comparing both geographical and cartesian SRSs.

The hand matches are given as a list of edges of the road network. Unfortunately, this leads to some discrepancies between the tracks, our matches and the hand matches. Often, tracks do not start directly at the beginning or ending of an edge but somewhere in-between. For example, when starting a movement from a parking lot aside of the road, the track does not begin at the start node of the corresponding edge but somewhere inbetween. As such, it is practically impossible for our algorithms to find a match that is perfectly fitting the given hand match (so has an error of exactly zero).

We can compute the error fraction of our map matching result to the hand match with the following equation from [53] with  $d_s$  being the distances erroneously subtracted,  $d_a$  being the distances erroneously added, and  $d_c$  being the correct distance:

$$e = (d_s + d_a)/d_c \tag{4}$$

For computing the error fraction, we compare the lines directly with each other. This enables to extract exactly the mismatching parts, even when the hand match contains the entire edge (by id) and our results start somewhere in the middle of the edge. Because our results consist of floating points (geographical points) our comparison algorithm has a complex sanitation and adaption process in the first step. This process adapts the lines to each other in nearby locations for eliminating the floating point differences and splits the lines when they divide into different directions for exact extracting of similar and different parts. This of course lets very tiny differences vanish but significant map matching errors only happen when different roads are taken, tiny differences between close points thus can be neglected.

 $<sup>^{14} \</sup>rm https://creativecommons.org/licenses/by-sa/4.0/legalcode$ 

<sup>&</sup>lt;sup>15</sup>https://planet.openstreetmap.org/gps/

 $<sup>^{16}</sup>$  https://creativecommons.org/licenses/by-sa/2.0/

 $<sup>^{17}</sup>$  https://planet.openstreetmap.org/

<sup>&</sup>lt;sup>18</sup>https://opendatacommons.org/licenses/odbl/

Our comparison algorithm in the second step then can work on binary equal positions and can exactly detect similar and different parts, even in roundabouts, zig-zag movements, reverse movements, and multi lines with gaps. After extracting all erroneously added and subtracted parts of the hand match, we can calculate the error after Equation 4.

For both tools that we compared, we used the default program parameters for the models. We assume that the authors of Fast map matching (FMM and ST-Match) know best which default parameters lead to fine results. For our tool, we also set default parameters that we found good working on a large variety of tracks. Our defaults, as long as they are not already explained in this paper, can be reviewed in our code by running our program without any parameters or with the '--help' parameter. Fast map matching in the version we benchmarked by default used an UBODT upper bound distance of 3000 m, a candidate search radius of 300 m and it retained the 8 k-nearest candidates within the found candidates for each track point. FMM calculated the distance error for its HMM observation probabilities relative to 50 m meter (called GPS error). The transition probabilities were calculated from the length fractions of the distance between point pairs and the length of the related found route, similar to our length metric part of our mixed reward. There were no bearings or direction changes used for the probabilities in FMM as far as we understood the program.

Our tests were all run on a Dual Socket AMD EPYC 7742 system with 2x 64 CPU-Cores and 1024 GB DDR4-3200 RAM on local RAIZ-Z1 SSD storage. We ran all tests in sequence so that the timing measurements were not impacted by any bandwidth limits or schedulers. For the memory and timing measurements we used /usr/bin/time. All tests were run on Ubuntu 20.04 LTS and both C++ programs (ours and FMM/ST-Match) were compiled locally with GCC 9.3.



Error fraction comparison benchmark

Figure 3: *Benchmark* results for *accuracy* – We compare on our MDP Value Iteration, Q-Learning and we also employ Value Iteration on reprojected coordinates for faster distance and bearing calculations. On our HMM we employ a Viterbi algorithm. We compare our results with FMM which is also based on HMM and Viterbi algorithm and ST-Match from the same tool suite. We can see that FMM and ST-Match fail to find results for all tracks of the map matching data set. This happened especially when the tracks had large gaps or the UBODT had no entries for a point pair. Our algorithms can handle this and find results for all tracks of the map matching data set. An error fraction > 1.0 means that there were more wrongly added or missed matches than the hand match is long. We can see that our algorithms have generally significantly fewer matching errors than FMM and ST-Match and always stay below an error fraction of 1.0. In the results with high matching errors, our Value Iteration on MDP approach also outperforms our Viterbi on HMM implementation significantly. Q-Learning is mostly worse than both Value Iteration and Viterbi but still is able to find good results in comparison with FMM and ST-Match.

algorithm	$\operatorname{count}$	min	$Q_{0.01}$	$Q_{0.10}$	$Q_{0.25}$	$Q_{0.50}$	$Q_{0.75}$	$Q_{0.90}$	$Q_{0.99}$	max	mean
Value Iteration	100	0.0004	0.0004	0.0015	0.0053	0.0130	0.0283	0.0600	0.2194	0.3142	0.0259
VI Reprojected	100	0.0004	0.0004	0.0013	0.0052	0.0105	0.0296	0.0630	0.2186	0.2392	0.0265
Viterbi	100	0.0000	0.0002	0.0016	0.0056	0.0120	0.0333	0.0837	0.4637	0.5120	0.0404
Q-Learning	100	0.0005	0.0009	0.0022	0.0062	0.0140	0.0347	0.0738	0.7793	0.8850	0.0459
FMM	90	0.0005	0.0007	0.0037	0.0086	0.0172	0.0663	0.1904	1.0605	1.4227	0.0976
ST-Match	94	0.0005	0.0008	0.0031	0.0071	0.0271	0.0796	0.3680	2.1069	2.8196	0.1690

Table 1: *Benchmark* results for *accuracy* – Here are some detailed error fraction metrics from Figure 3. We can see that Value Iteration on our MDP is on average (mean) much better than every other algorithm and model. We can also see that using a reprojected SRS (VI = Value Iteration) leads to similar good matching errors as using a geographic SRS even though cartesian distance and bearing calculations are less exact than geographic ones.



Figure 4: *Benchmark* results for *duration* – We see that ST-Match is the overall fastest algorithm and takes less than a second in more than 70% of the cases. However, the result errors were not so favorable on average compared with our algorithms and models. FMM only works after an UBODT has been generated. We can see that the UBODT generation takes the longest time and even then, FMM is significantly slower on some tracks than our algorithms. This is because we did not only measure the matching time but also the time it needed to import the UBODT. For our algorithms, we also not only measured the matching time but also the network import and sanitation times of networks and tracks. After all, when matching one track, the whole running time is important. Our Value Iteration algorithm is nearly as fast as our Viterbi algorithm, which behaves exactly as expected. Our Value iteration algorithm on the reprojected SRS is significantly faster in all cases than our Value Iteration on the geographic SRS. Q-Learning is often slower than our Value Iteration algorithm due to the many episodes needed until convergence.

algorithm	min	$Q_{0.01}$	$Q_{0.10}$	$Q_{0.25}$	$Q_{0.50}$	$Q_{0.75}$	$Q_{0.90}$	$Q_{0.99}$	max	mean
Value Iteration	0.23	0.36	0.90	1.90	5.03	10.88	16.35	44.87	104.98	8.30
VI Reprojected	0.06	0.09	0.23	0.57	1.76	3.98	7.29	22.72	37.93	3.24
Viterbi	0.23	0.36	0.89	1.83	4.74	9.63	15.75	44.63	83.17	7.56
Q-Learning	0.23	0.33	0.94	2.02	4.37	12.77	27.34	84.41	375.83	13.69
UBODT-GEN	0.12	0.12	0.96	3.94	19.13	50.58	167.67	337.43	351.62	51.20
$\operatorname{FMM}$	0.09	0.09	0.49	1.75	7.49	20.11	143.16	618.68	1822.28	55.46
ST-Match	0.08	0.11	0.20	0.29	0.51	0.97	1.62	35.46	475.59	5.95

Table 2: Benchmark results for duration – Here are some detailed duration measurements in seconds to Figure 4. We can see than on average our Value Iteration (=VI) on the reprojected SRS is much faster than ST-Match (which also computes on the reprojected SRS). This comes mainly from one track where ST-Match needs nearly  $\approx 8 \text{ min}$  to match whereas our algorithm on reprojected SRS only needs  $\approx 0.63 \text{ min}$  maximum. We can also see that Fast map matching (FMM) needs much more time than all our algorithms. This is mainly because for every track an UBODT needs to be generated and imported for the whole corresponding network graph though only a fraction of routes is really needed. We can also see that the Viterbi algorithm on HMM is only slightly faster compared to our Value Iteration on MDP algorithm, as expected, but this comes with a larger matching error. We can see that Q-Learning is not really fast in comparison, which is the expected behavior of a learning algorithm. Still it is not extremely slow, so more tuning in the action selection process can further improve the speed.

algorithm	$\min$	$Q_{0.01}$	$Q_{0.10}$	$Q_{0.25}$	$Q_{0.50}$	$Q_{0.75}$	$Q_{0.90}$	$Q_{0.99}$	max	mean
Value Iteration	32.90	39.04	64.33	107.81	231.16	383.28	582.52	1200.44	3154.63	303.29
VI Reprojected	35.12	38.30	51.58	79.11	145.56	240.06	335.20	625.66	1318.00	188.98
Viterbi	32.63	37.23	51.38	82.89	150.35	251.31	368.11	590.16	1073.63	186.33
Q-Learning	33.43	38.37	61.27	106.79	208.95	371.18	544.62	1203.46	3326.86	286.36
UBODT-GEN	53.62	53.98	54.59	55.93	59.16	69.04	89.91	134.16	149.76	66.82
UBODT filesize	1.41	1.51	16.88	72.69	338.15	903.66	2729.45	5137.22	5303.60	873.58
FMM	57.32	57.83	84.00	179.17	573.22	1439.52	3562.72	6628.64	6682.41	1241.42
ST-Match	55.29	55.59	57.32	59.51	64.90	75.21	104.86	149.44	156.62	73.13

Table 3: *Benchmark* results for *memory consumption* – These are some details of the memory measurements in MegaBytes from Figure 5. For the UBODT filesize we used the filesize from the filesystem. We can see that on average, Q-Learning needs less memory than Value Iteration because it visits less states of the MDP whereas Value Iteration visits all states. Still the difference is not much and the matching error of Value Iteration is significantly better.

algorithm	min	$Q_{0.01}$	$Q_{0.10}$	$Q_{0.25}$	$Q_{0.50}$	$Q_{0.75}$	$Q_{0.90}$	$Q_{0.99}$	max	mean
Value Iteration	6664	36226	233483	514140	1733568	3596893	6943929	23123159	58768564	3386264
VI Reprojected	6866	25147	128589	270955	754477	1594903	3049655	9204441	21261622	1470803
Viterbi	5240	33796	224442	502229	1743849	3586903	6879679	23035197	58389964	3363861
Q-Learning	6280	35640	233161	495833	1662620	3552087	6850338	22726828	57627268	3279703
$\mathbf{FMM}$	16704	22977	37933	57920	92096	178672	273453	1028020	1235968	158177
ST-Match	16704	22977	37933	57920	92096	178672	273453	1028020	1235968	158177

Table 4: Benchmark results for candidate combinations – These are some details of the candidate combinations from Figure 6. We can see that our geographic MDP and HMM algorithms on average use  $\approx 20$  times more candidate combinations than FMM and ST-Match. Though this is a huge amount of additional candidate combinations to examine in comparison, our best algorithm (Value Iteration on MDP) on average only needs  $\approx 2$  times more processing time and  $\approx 4$  times more memory than the fastest algorithm ST-Match while finding matching results with  $\approx 6$  times less errors (for all tracks, compared to ST-Match which fails on six tracks). Concerning the one track where ST-Match needed  $\approx 10$  min and comparing the maximum candidate combinations with our Value Iteration and Viterbi algorithms, we can see that our algorithms are more robust in edge cases concerning the processing speed.



#### Memory consumption (resident set size) comparison

Figure 5: *Benchmark* results for *memory consumption* – Here we can see how much Random Access Memory (RAM) each algorithm needed for matching. For the UBODT we also plotted the file size on the file system. This shows why FMM needs much more memory in comparison to all other algorithms, because it needs to import the UBODT before it can match. We are sure that with better chosen program parameters FMM becomes faster and less resource hungry, but we compared all tools with default parameters in our experiments. ST-Match mostly needs very few memory due to its simpler model design and because it needs no UBODT. Our algorithms all need most of the time more memory than ST-Match but always less than FMM because we only compute the routes that we need. From our algorithms, our Q-Learning and Value Iteration on MDP algorithms need most memory in comparison. Using the Viterbi algorithm and HMM model needs significantly less memory but has a higher matching error. Using a reprojected SRS also needs less memory, because less candidate combinations are examined (see Figure 6).



Figure 6: Benchmark results for candidate combinations – Here we can see how many pairs of candidates each algorithm examined during matching. Each candidate combination is equal to computing a reward after Equation 3. Our algorithms wrote out the exact number of combinations by returning the size of the internal state-action cache used in index mapping. For FMM and ST-Match, we used the following formula  $(8 * 8) * (n_{points} - 1)$  with  $n_{points}$  being the points of the track. Each point had 8 k-nearest candidates, so between each pair of candidates, 8 \* 8 combinations had to be examined. We can see that our algorithms generally examine much more candidate combinations with our default program parameters, which is due to our candidate adoption and search circle without k-nearest. This has a processing speed and memory penalty on our algorithms compared to ST-Match and the map matching phase of FMM without UBODT generation and import. But concerning the comparably huge amount of candidate combinations our algorithms examine, the additional memory and time consumption we need is impressively small, due to our performance adjustments. We can also see that our Q-Learning, Viterbi and Value Iteration algorithm all examine comparable amounts of candidate combinations. Our reprojected algorithm examines much less candidate combinations, which explains why it is faster and needs less memory. The reason for this is that a search circle with a radius of 200 m in a cartesian SRS is smaller than a circle with a radius of 200 m in a geographic SRS, so less candidates are found and need to be examined. Even though our reprojected algorithm uses less candidates, it finds similar good results (see Table 1). Further tuning of our default parameters for increased processing speed while maintaining similar results can be subject of subsequent research. Also comparing FMM and ST-Match with higher than default k-nearest setting remains of interest but is out of scope in this work.

# 5 Discussion and conclusion

We were able to show that map matching with MDPs and Value Iteration is a more robust solution compared to HMMs with Viterbi algorithm concerning its matching error especially in difficult map matching situations. Though our MDPs are a little bit slower to compute than HMMs, they are still very competing due to their much lower error fractions. On average, our MDPs with Value Iteration has a matching error of  $\approx 2.59\%$  compared to our HMMs with Viterbi algorithm with an average matching error of  $\approx 4.04\%$ , which is  $\approx 35.9\%$  more accurate while being only  $\approx 9.7\%$  slower.

We showed that on our MDP it is also possible to employ learning algorithms such as Q-Learning, although they need more tuning for increased performance. Also, more advanced reinforcement learning algorithms, for example 'n-step' bootstrapping [65] can further improve the performance. This remains subject of subsequent research. Still, we have shown that HMMs with Viterbi algorithm also can benefit from more complex metrics such as direction changes compared to HMMs with Viterbi algorithm without such complex metrics, such as the implementations in FMM. Most of our improvements (e.g., our candidate adoption, index mapping, simplification, sanitation improvements, and others) can be used in both HMM methods as well as our new MDP methods.

We have seen that an understanding of 'typical road behavior' can be implemented by using absolute reward metrics as in Equation 3. The benchmark of different weights from 0.0 to 1.0 or more against each other to see how much impact each fraction of our metrics formula has on the matching error can be part of subsequent research.

Moreover, it is possible to combine artificial neural networks with our reinforcement learning algorithms, for example trajectory pattern analysis can further reduce the amount of candidates [80]. Similarly, it is possible to rely on the history of the measurements for selecting next candidates [87].

We believe our work contains many ideas and details on how to improve map matching technology. Empowering Markov Decision Processes for map matching in fact shows a promising and flexible way of reducing remaining matching errors while still being competitive concerning their computational speed.

## Data and codes availability statement

The map matching data set [36] from [37] is available under CC BY-SA  $4.0^{19}$  at https://zenodo.org/record/57731

Our code is available as C++ code (CMake project) under AGPL 3.0<sup>20</sup> license at https://github.com/iisys-hof/map-matching-2

Please review the provided Dockerfile for instructions on how to compile and run the software.

## **Disclosure statement**

There are no competing interests.

# Funding

This work was supported by the 'Bundesministerium für Verkehr und digitale Infrastruktur' in the research projects 'Mobilität Digital Hochfranken' and 'Shuttle Modellregion Oberfranken'.

# References

- E. Abbott and D. Powell. Land-vehicle navigation using GPS. Proceedings of the IEEE, 87(1):145–162, 1 1999.
- [2] Pankaj K. Agarwal, Rinat Ben Avraham, Haim Kaplan, and Micha Sharir. Computing the Discrete Fréchet Distance in Subquadratic Time. CoRR, 4 2012.
- [3] Helmut Alt, Alon Efrat, Günter Rote, and Carola Wenk. Matching planar maps. Journal of Algorithms, 49(2):262–283, 11 2003.

<sup>&</sup>lt;sup>19</sup>https://creativecommons.org/licenses/by-sa/4.0/legalcode

 $<sup>^{20} \</sup>rm https://www.gnu.org/licenses/agpl-3.0.en.html$ 

- [4] Henri Andoyer. Formule Donnant la Longueur de la Géodésique Joignant 2 Points de L'ellipsoïde Donnés Par Leurs Coordonnées Géographiques. Notices Scientifiques, 34(1):77–81, 4 1932.
- [5] Markus Auer, Hubert Rehborn, Sven-Eric Molzahn, and Klaus Bogenberger. Boosting performance of map matching algorithms by parallelization on graphics processors. In 2017 IEEE Intelligent Vehicles Symposium (IV), pages 462–467. IEEE, 6 2017.
- [6] Ali Esref Aytac, Orhan Aksoy, and Yusuf Sinan Akgul. Ship positioning by matching radar images and map data. In 2014 22nd Signal Processing and Communications Applications Conference (SIU), pages 1423–1426. IEEE, 4 2014.
- [7] Yoonsik Bang, Jiyoung Kim, and Kiyun Yu. An improved map-matching technique based on the fréchet distance approach for pedestrian navigation services. *Sensors*, 16(10):1768–1786, 10 2016.
- [8] Richard Bellman. A markovian decision process. Journal of Mathematics and Mechanics, 6(5):679–68, 4 1957.
- [9] David Bernstein and Alain L. Kornhauser. An introduction to map matching for personal navigation assistants, 8 1996.
- [10] Nathaniel Bowditch. The American Practical Navigator. National Imagery and Mapping Agency, Bethesda, Maryland, 1 1995.
- [11] 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, pages 853–864, Trondheim, Norway, 8 2005. VLDB Endowment.
- [12] Benedikt Budig. An algorithm for map matching on incomplete road databases. Master's thesis, Julius-Maximilians-Universität Würzburg, Würzburg, 2 2012.
- [13] Francois Caron, Emmanuel Duflos, Denis Pomorski, and Philippe Vanheeghe. GPS/IMU data fusion using multisensor kalman filtering: introduction of contextual aspects. *Information Fusion*, 7(2):221–230, 6 2006.
- [14] Pingfu Chao, Yehong Xu, Wen Hua, and Xiaofang Zhou. A survey on map-matching algorithms, 10 2019.
- [15] Mingliang Che, Yingli Wang, Chi Zhang, and Xinliang Cao. An enhanced hidden markov map matching model for floating car data. *Sensors (Basel)*, 18(6):1758–1777, 5 2018.
- [16] P. Dabove, G. Ghinamo, and A. M. Lingua. Inertial sensors for smartphones navigation. SpringerPlus, 4(1):834–852, 12 2015.
- [17] Anthony J. DeGregoria. Gravity Gradiometry and Map Matching: An Aid to Aircraft Inertial Navigation Systems. Air Force Institute of Technology, Ohio, 10 2010.
- [18] Thomas G. Dietterich. Machine learning for sequential data: A review. In Lecture Notes in Computer Science, pages 15–30. Springer Berlin Heidelberg, Berlin, Heidelberg, 1 2002.
- [19] Edsger Wybe Dijkstra. A note on two problems in connexion with graphs. In Numerische Mathematik, volume 1, pages 269–271. Springer, 1 1959.
- [20] David H. Douglas and Thomas K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2):112–122, 10 1973.
- [21] Thomas Eiter and Heikki Mannila. Computing discrete frechet distance, 5 1994.
- [22] European Global Navigation Satellite Systems Agency. European GNSS (galileo) open service definition document, 5 2019.
- [23] Esther Calvo Fernández, J. Cordero, G. Vouros, N. Pelekis, Theocharis Kravaris, H. Georgiou, G. Fuchs, Enrique Casado, P. Costas, and S. Ayhan. DART: A machine-learning approach to trajectory prediction and demand-capacity balancing. In *Seventh SESAR Innovation Days*, SESAR, 11 2017.

- [24] David Fiedler, Michal Čáp, Jan Nykl, Pavol Žilecký, and Martin Schaefer. Map Matching Algorithm for Large-scale Datasets. CoRR, 9 2019.
- [25] G.D. Forney. The viterbi algorithm. Proceedings of the IEEE, 61(3):268–278, 3 1973.
- [26] Matteo Francia, Enrico Gallinucci, and Federico Vitali. Map-matching on big data: a distributed and efficient algorithm with a hidden markov model. In 2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), pages 1238–1243. IEEE, 5 2019.
- [27] C.Y. Goh, J. Dauwels, N. Mitrovic, M. T. Asif, A. Oran, and P. Jaillet. Online map-matching based on hidden markov model for real-time traffic sensing applications. In 2012 15th International IEEE Conference on Intelligent Transportation Systems, pages 776–781. IEEE, 9 2012.
- [28] Joshua S. Greenfeld. Matching GPS observations to locations on a digital map. In Transportation Research Board 81st Annual Meeting, Washington D.C., 1 2002.
- [29] Awni Hannun. The label bias problem, 11 2019.
- [30] Noriaki Hirosue. Map matching with hidden markov model on sampled road network. In Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012), pages 2242–2245. IEEE, 11 2012.
- [31] Yigong Hu and Binbin Lu. A hidden markov model-based map matching algorithm for low sampling rate trajectory data. *IEEE Access*, 7:178235–178245, 12 2019.
- [32] Zhengfeng Huang, Shaojie Qiao, Nan Han, Chang-an Yuan, Xuejiang Song, and Yueqiang Xiao. Survey on vehicle map matching techniques. *CAAI Transactions on Intelligence Technology*, 6 2020.
- [33] Britta Hummel. Map matching for vehicle guidance. Dynamic and mobile GIS. Ed.: J. Drummond, 10:157–168, 1 2006.
- [34] Charles F. F. Karney. Algorithms for geodesics. Journal of Geodesy, 87(1):43–55, 1 2013.
- [35] John Krumm, Eric Horvitz, and Julie Letchner. Map matching with travel time constraints. In SAE World Congress & Exhibition, Warrendale, PA, United States, 4 2007. SAE International.
- [36] Matej Kubicka, Arben Cela, Philippe Moulin, Hugues Mounier, and S. I. Niculescu. Dataset for testing and training map-matching methods, 7 2016.
- [37] Matej Kubicka, Arben Cela, Philippe Moulin, Hugues Mounier, and S.I. Niculescu. Dataset for testing and training of map-matching algorithms. In 2015 IEEE Intelligent Vehicles Symposium (IV). IEEE, 6 2015.
- [38] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 282–289, San Francisco, CA, USA, 6 2001. Morgan Kaufmann Publishers Inc.
- [39] B. D. S. Lakmali and Dileeka Dias. Database correlation for GSM location in outdoor & indoor environments. In 2008 4th International Conference on Information and Automation for Sustainability, pages 42–47. IEEE, 12 2008.
- [40] Bijun Li, Yuan Guo, Jian Zhou, and Yi Cai. A data correction algorithm for low-frequency floating car data. Sensors, 18(11):3639–3656, 10 2018.
- [41] Hengfeng Li, Lars Kulik, and Kotagiri Ramamohanarao. Spatio-temporal trajectory simplification for inferring travel paths. In *Proceedings of the 22nd ACM SIGSPATIAL International Conference* on Advances in Geographic Information Systems, SIGSPATIAL '14, pages 63–72, New York, NY, USA, 11 2014. Association for Computing Machinery.
- [42] Hengfeng Li, Lars Kulik, and Kotagiri Ramamohanarao. Robust inferences of travel paths from GPS trajectories. International Journal of Geographical Information Science, 29(12):2194–2222, 12 2015.

- [43] Jun Li, Qiming Qin, Chao Xie, and Yue Zhao. Integrated use of spatial and semantic relationships for extracting road networks from floating car data. *International Journal of Applied Earth Observation* and Geoinformation, 19:238–247, 10 2012.
- [44] Biwei Liang, Tengjiao Wang, Shun Li, Wei Chen, Hongyan Li, and Kai Lei. Online learning for accurate real-time map matching. In James Bailey, Latifur Khan, Takashi Washio, Gill Dobbie, Joshua Zhexue Huang, and Ruili Wang, editors, Advances in Knowledge Discovery and Data Mining, pages 67–78, Cham, 4 2016. Springer International Publishing.
- [45] Lin Liao, Dieter Fox, and Henry Kautz. Extracting places and activities from GPS traces using hierarchical conditional random fields. *The International Journal of Robotics Research*, 26(1):119–134, 1 2007.
- [46] 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, pages 352–361, New York, NY, USA, 11 2009. Association for Computing Machinery.
- [47] Thomas Louail, Maxime Lenormand, Oliva García Cantú, Miguel Picornell, Ricardo Herranz, Enrique Frias-Martinez, José J. Ramasco, and Marc Barthelemy. From mobile phone data to the spatial structure of cities. *Scientific Reports*, 4(1):5276–5287, 6 2014.
- [48] An Luo, Shenghua Chen, and Bin Xv. Enhanced map-matching algorithm with a hidden markov model for mobile phone positioning. *ISPRS International Journal of Geo-Information*, 6(11):327–344, 10 2017.
- [49] F. Marchal, J. Hackney, and K. W. Axhausen. Efficient map matching of large global positioning system data sets: Tests on speed-monitoring experiment in zürich. *Transportation Research Record*, 1935(1):93–100, 1 2005.
- [50] Adam Millard-Ball, Robert C. Hampshire, and Rachel R. Weinberger. Map-matching poor-quality GPS data in urban environments: the pgmapmatch package. *Transportation Planning and Technology*, 42(6):539–553, 5 2019.
- [51] Koichi Miyashita, Tsutomu Terada, and Shojiro Nishio. A map matching algorithm for car navigation systems that predict user destination. In 22nd International Conference on Advanced Information Networking and Applications - Workshops (aina workshops 2008), pages 1551–1556. IEEE, 3 2008.
- [52] James Murphy, Yuanyuan Pao, and Albert Yuen. Map matching when the map is wrong: Efficient vehicle tracking on- and off-road for map learning, 9 2019.
- [53] Paul Newson and John Krumm. Hidden markov map matching through noise and sparseness. In Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, GIS '09, pages 336–343, New York, NY, USA, 11 2009. Association for Computing Machinery.
- [54] Office of the Department of Defense. Global positioning system standard positioning service performance standard, 4 2020.
- [55] Elena Pashenkova, Irina Rish, and Rina Dechter. Value iteration and policy iteration algorithms for markov decision problem, 4 1996.
- [56] Francisco Câmara Pereira, Hugo Costa, and Nuno Martinho Pereira. An off-line map-matching algorithm for incomplete map databases. *European Transport Research Review*, 1(3):107–124, 9 2009.
- [57] Oliver Pink and Britta Hummel. A statistical approach to map matching using road network geometry, topology and vehicular motion constraints. In 2008 11th International IEEE Conference on Intelligent Transportation Systems, pages 862–867. IEEE, 10 2008.
- [58] Mohammed Quddus and Simon Washington. Shortest path and vehicle trajectory aided mapmatching for low frequency GPS data. Transportation Research Part C: Emerging Technologies, 55:328–339, 6 2015.

- [59] Mohammed A. Quddus, Washington Y. Ochieng, and Robert B. Noland. 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, 10 2007.
- [60] Ming Ren. Advanced Map Matching Technologies and Techniques for Pedestria/Wheelchair Navigation. PhD thesis, University of Pittsburg, Pittsburgh, 3 2012.
- [61] Joerg Schweizer, Federico Rupi, and Silvia Bernardi. Map-matching algorithm applied to bicycle global positioning system traces in bologna. *IET Intelligent Transport Systems*, 10(4):244–250, 6 2016.
- [62] I. Skog and P. Handel. In-car positioning and navigation technologies a survey. IEEE Transactions on Intelligent Transportation Systems, 10(1):4–21, 2 2009.
- [63] Aaron J. Snoswell, Surya P. N. Singh, and Nan Ye. Revisiting Maximum Entropy Inverse Reinforcement Learning: New Perspectives and Algorithms. In 2020 IEEE Symposium Series on Computational Intelligence (SSCI). IEEE, 12 2020.
- [64] Wenbin Sun and Ting Xiong. A low-sampling-rate trajectory matching algorithm in combination of history trajectory and reinforcement learning. Acta Geodaetica et Cartographica Sinica, 45(11):1328, 1 2016.
- [65] Richard S. Sutton and Andrew G. Barto. Reinforcement Learning: An Introduction (Second Edition). Adaptive computation and machine learning series. The MIT Press, Cambridge, Massachusetts, 1 2018.
- [66] Shun Taguchi, Satoshi Koide, and Takayoshi Yoshimura. Online map matching with route prediction. IEEE Transactions on Intelligent Transportation Systems, 20(1):338–347, 3 2018.
- [67] Luliang Tang, Xue Yang, Zhen Dong, and Qingquan Li. CLRIC: Collecting lane-based road information via crowdsourcing. *IEEE Transactions on Intelligent Transportation Systems*, 17(9):2552–2562, 3 2016.
- [68] Luliang Tang, Xue Yang, Zihan Kan, and Qingquan Li. Lane-level road information mining from vehicle GPS trajectories based on naïve bayesian classification. *ISPRS International Journal of Geo-Information*, 4(4):2660–2680, 11 2015.
- [69] Fernando Torre, David Pitchford, Phil Brown, and Loren Terveen. Matching GPS traces to (possibly) incomplete map data: Bridging map building and map matching. In Proceedings of the 20th International Conference on Advances in Geographic Information Systems, SIGSPATIAL '12, pages 546–549, New York, NY, USA, 11 2012. Association for Computing Machinery.
- [70] Nagendra R. Velaga, Mohammed A. Quddus, and Abigail L. Bristow. Improving the performance of a topological map-matching algorithm through error detection and correction. *Journal of Intelligent Transportation Systems*, 16(3):147–158, 7 2012.
- [71] T. Vincenty. Direct and Inverse Solutions of Geodesics on the Ellipsoid with Application of Nested Equations. Survey Review, 23(176):88–93, 4 1975.
- [72] Christopher John Cornish Hellaby Watkins. Learning from Delayed Rewards. PhD thesis, King's College, 5 1989.
- [73] Christopher John Cornish Hellaby Watkins and Peter Dayan. Q-learning. Technical Report 3-4, Machine Learning, 5 1992.
- [74] Hong Wei, Yin Wang, George Forman, and Yanmin Zhu. Map matching: Comparison of approaches using sparse and noisy data. In *Proceedings of the 21st ACM SIGSPATIAL International Conference* on Advances in Geographic Information Systems, SIGSPATIAL '13, pages 444–447, New York, NY, USA, 11 2013. Association for Computing Machinery.
- [75] Hong Wei, Yin Wang, George Forman, Yanmin Zhu, and Haibing Guan. Fast viterbi map matching with tunable weight functions. In *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '12, pages 613–616, New York, NY, USA, 11 2012. Association for Computing Machinery.

- [76] Christopher E White, David Bernstein, and Alain L Kornhauser. Some map matching algorithms for personal navigation assistants. *Transportation Research Part C: Emerging Technologies*, 8(1-6):91–108, 2 2000.
- [77] William J. Hughes Technical Center. Global positioning system (GPS) standard positioning service (SPS) performance analysis report. Technical report, Atlantic City International Airport, Washington, DC, 1 2017.
- [78] Ming Xu, Yiman Du, Jianping Wu, and Yang Zhou. Map matching based on conditional random fields and route preference mining for uncertain trajectories. *Mathematical Problems in Engineering*, 2015:1–13, 9 2015.
- [79] Can Yang and Győző Gidófalvi. Fast map matching, an algorithm integrating hidden markov model with precomputation. International Journal of Geographical Information Science, 32(3):547–570, 3 2018.
- [80] Can Yang and Győző Gidófalvi. Detecting regional dominant movement patterns in trajectory data with a convolutional neural network. *International Journal of Geographical Information Science*, 34(5):996–1021, 12 2019.
- [81] Haiqiang Yang, Shaowu Cheng, Huifu Jiang, and Shi An. An enhanced weight-based topological map matching algorithm for intricate urban road network. *Proceedia - Social and Behavioral Sciences*, 96:1670–1678, 11 2013.
- [82] Yifang Yin, Rajiv Ratn Shah, and Roger Zimmermann. A general feature-based map matching framework with trajectory simplification. In *Proceedings of the 7th ACM SIGSPATIAL International Workshop on GeoStreaming*, IWGS '16, pages 1–10, New York, NY, USA, 10 2016. Association for Computing Machinery.
- [83] Mohamed A. Yousef and Mustafa K. Ragheb. Effect of recording interval on GPS accuracy. Journal of Engineering Sciences, 42(5):1215–1231, 9 2014.
- [84] Jing Yuan, Yu Zheng, Chengyang Zhang, Xing Xie, and Guang-Zhong Sun. An interactive-voting based map matching algorithm. In 2010 Eleventh International Conference on Mobile Data Management, pages 43–52. IEEE, 5 2010.
- [85] Ayman Zeidan, Eemil Lagerspetz, Kai Zhao, Petteri Nurmi, Sasu Tarkoma, and Huy T. Vo. Geomatch: Efficient large-scale map matching on apache spark. In 2018 IEEE International Conference on Big Data (Big Data), pages 384–391. IEEE, 12 2018.
- [86] Ethan Zhang and Neda Masoud. Increasing GPS localization accuracy with reinforcement learning. IEEE Transactions on Intelligent Transportation Systems, pages 1–12, 2 2020.
- [87] Kai Zhao, Jie Feng, Zhao Xu, Tong Xia, Lin Chen, Funing Sun, Diansheng Guo, Depeng Jin, and Yong Li. DeepMM: Deep learning based map matching with data augmentation. In *Proceedings* of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, SIGSPATIAL '19, pages 452–455, New York, NY, USA, 11 2019. Association for Computing Machinery.
- [88] Ke Zheng and Dunyao Zhu. A novel clustering algorithm of extracting road network from low-frequency floating car data. *Cluster Computing*, 22(5):12659–12668, 1 2018.
- [89] Brian D. Ziebart, Andrew Maas, J. Andrew Bagnell, and Anind K. Dey. Maximum entropy inverse reinforcement learning. In *Proceedings of the 23rd National Conference on Artificial Intelligence*, volume Volume 3 of AAAI '08, pages 1433–1438. AAAI Press, 7 2008.
- [90] Xinyan Zou, Zhixiang Fang, Haoyu Zhong, Zhongheng Wu, and Xiongyan Liu. Inertia mutation energy model to extract roads by crowdsourcing trajectories. *IEEE Access*, 7:186393–186408, 12 2019.