Map-matching poor-quality GPS data in urban environments: The pgMapMatch package

Preprint of a paper published in *Transportation Planning and Technology,* 2019

The published version is at https://doi.org/10.1080/03081060.2019.1622249

The code repository is at https://github.com/amillb/pgMapMatch

Adam Millard-Ball
Environmental Studies Department, University of California, Santa Cruz
adammb@ucsc.edu

Robert C. Hampshire
Transportation Research Institute, University of Michigan
hamp@umich.edu

Rachel R. Weinberger
Weinberger & Associates
rrw@pobox.com

Abstract

Global Positioning System (GPS) data have become ubiquitous in many areas of transportation planning and research. The usefulness of GPS data often depends on the points being matched to the true sequence of edges on the underlying street network—a process known as "map matching." This paper presents a new map-matching algorithm that is designed for use with poor-quality GPS traces in urban environments, where drivers may circle for parking and GPS quality may be affected by underground parking and tall buildings. The paper is accompanied by open-source Python code that is designed to work with a PostGIS spatial database. In a test dataset that includes many poor-quality traces, our new algorithm accurately matched about one-third more traces than a widely available alternative. Our algorithm also provides a "match score" that evaluates the likelihood that the match for an individual trace is correct, reducing the need for manual inspection.

Key words: Global Positioning Systems (GPS); map-matching; Geographic Information Systems

1. Introduction

Global Positioning System (GPS) data have become ubiquitous in many areas of transportation planning and academic research in recent years. Household travel surveys often include a GPS-equipped subsample, providing analysts with the path of each trip as well as data on demographics, trip purpose and the origin and destination (Bricka & Bhat 2006; Bricka et al. 2012). Commercial resellers maintain GPS datasets collected from cellphones and navigation systems. Smartphone apps and GPS receivers also make it simple for users to collect their own traces.

The usefulness of GPS data often depends on the points being matched to the underlying street network—a process known as "map matching." Analysis of congestion, parking availability, land-use types or other characteristics of the route relies on knowledge of the streets being traveled, not just the locations of individual GPS vertices. Moreover, the length of the GPS trace may differ from the actual route taken. The GPS trace could be longer (because of noise in the location data) or shorter (because GPS vertices are connected with straight lines, rather than following the street network), which in turn affects estimates of travel distances and speeds.

Formally, the map-matching problem is to identify the true path, consisting of a sequence of street edges (i.e. blocks), given a sequence of GPS coordinates. Several different algorithms, discussed in detail in the following section, have been developed to map-match GPS data. However, there are three major shortcomings of existing approaches. First, some map-matching algorithms assume that the user takes the fastest route, and does not circle or deviate. This is an appropriate assumption in some contexts, but not in others, for example when drivers circle or "cruise" in search of a parking space (Millard-Ball et al. 2014; Hampshire et al. 2016). Second, published algorithms often leave the implementation to the user. Computer code is usually not made publicly available, at least in a form that is compatible with standard GIS tools. Third, when map-matching large datasets where the matches cannot be individually inspected, published algorithms do not provide a way to evaluate the likelihood that the match is correct. Global measures such as R^2 can be helpful, but do not allow the accuracy of individual matches to be assessed, while integrity monitoring systems such as Li et al. (2013) rely on augmenting the GPS signal with supplemental data.

This paper presents a new map-matching algorithm that is designed for use with poor-quality, heterogeneous GPS traces in urban environments, where drivers may circle for parking and GPS quality may be affected by underground parking, tall buildings and similar obstacles. The algorithm considers the distance between GPS locations and candidate streets (a geometric component); the ratio of GPS path length to candidate map-matched path lengths (a topological component); and the implied speed of a map-matched path (temporal component). In combination, these components provide a robust way to deal with inaccuracies in the underlying GPS trace. We consider poor-quality data to be typified by sparse data (e.g. 30 seconds or more between points), noisy measurements, unrealistic movements, and spatial inaccuracies of 50m or more that would lead an algorithm based on only one of these components astray. Examples of the types of poor-quality traces used in this paper are shown in Figure 3 and Figure 5.

The paper is accompanied by open-source Python code that is designed to work in a GIS environment, specifically with a PostGIS spatial database. Users can use this code to immediately apply the algorithm to match their own traces. Moreover, we provide a new method to assess the likelihood that a good match has been made, enabling the automated analysis of noisy GPS traces without the need for manual inspection of the matches. Thus, our major contribution is to make easy-to-use map-matching tools available to the GIS community, using two common tools in GIS analysis—Python and PostGIS.

We begin with a brief review of map-matching algorithms and their implementations. We then discuss our own algorithm, before presenting the results of a test on two GPS datasets—one collected by the authors, and one from a commercial provider. We compare the speed and accuracy of our algorithm to a freely available alternative, GraphHopper. Finally, we introduce a method for automated estimation of the quality of the match, that can help identify matches that should be discarded or warrant inspection by the analyst.

2. Review of Map-Matching Algorithms

We focus here on post-processing algorithms that are applied to previously collected datasets, rather than real-time algorithms that are useful in navigation and vehicle location applications. (For discussions of real-time approaches, see, for example, Quddus et al. 2007; Velaga et al. 2009.) Real-time approaches, while accounting for the vast majority of the published literature, are less relevant for the discussion here because they are primarily concerned with identifying *current* position, rather than the integrity of the route as a whole. For example, a real-time algorithm can jump to a new position without the constraint of ensuring a feasible route from the previous position (Van Dijk & De Jong 2017). This review is not intended to be comprehensive, but rather to identify some of the main approaches that have been used in the literature, and some of their strengths and weaknesses.

Two principal typologies have been developed to help understand the differences between competing map-matching algorithms. One distinction is between local algorithms, which work incrementally and find the best match for individual points or groups of points; and global algorithms, which attempt to match the entire trace at once. In one global approach, Brakatsoulas et al. (2005) use measures of curve similarity to find the best match. Another distinction is between (i) geometric algorithms, which rely on the distance between GPS points and candidate edges in the road network, and sometimes also incorporate heading information; and (ii) topological algorithms, which take into account the connectivity of the road network.

In practice, the most successful algorithms blend characteristics from local and global and geometric and topological approaches. Typically, a likelihood or scoring function, which is comprised of both geometric and topological components, is applied to candidate paths. The geometric score can be simply the inverse distance of each GPS point from the candidate path (e.g. Camargo et al. 2017). A more complex geometric approach generates match probabilities as a nonlinear function of distance, based on a structural model of GPS error (e.g. Newson & Krumm 2009). The topological score, meanwhile, can be based on the difference between the network distance along the candidate path, and the Euclidean (or great circle) distance between adjacent GPS points (Lou et al. 2009; Newson & Krumm 2009). Alternatively, the topological score could be calculated based on a routing algorithm such as Dijkstra (Wei et al. 2012; for a related approach, see Van Dijk & De Jong 2017). For a useful compilation of different scoring or probability functions, see Wei et al. (2012).

The tradeoff between local and global approaches can be resolved through retaining multiple path candidates as each point in the trace is processed incrementally, before choosing the best-scoring candidate in the final step. Such a "Multiple Hypothesis Technique" avoids locking in on a single solution that may be rendered less likely by subsequent points on the trace, and in effect allows later GPS points to inform the matching score of previously processed points. One example of an algorithm based on the Multiple Hypothesis Technique is provided by Marchal et al. (2005) and refined by Schuessler and Axhausen (2009); this algorithm has been used to map-match large datasets such as the California Household Travel Survey.

A more formal variant of the Multiple Hypothesis Technique uses a Hidden Markov Matrix (HMM), under which the states of the matrix represent edges in the road network. State probabilities (equivalent to a geometric scoring function) indicate the likelihood of observing a given GPS point conditional on the vehicle being in a given state (i.e., on a given edge). Transition probabilities (equivalent to a topological scoring function) indicate the probabilities of moving from one state to another, i.e. moving between different edges. The Viterbi algorithm can then be applied to find the optimal path through the matrix (Newson & Krumm 2009; Wei et al. 2012). A refinement by Li et al. (2015), meanwhile, generates spatial-linear clusters of GPS points, rather than treating each point independently; a HMM model is then used to identify the optimal combination of the clusters.

Typically, a map-matching paper will describe its algorithm, provide evidence as to its speed and accuracy, and sometimes compare its performance against other map-matching algorithms. Comparison is hampered, however, in several ways. First, each algorithm has its own strengths and idiosyncrasies. For example, the algorithm by Schuessler and Axhausen (2009) explicitly disallows circling (i.e. repeating any sequence of links) and U-turns, which may increase the overall quality and speed of the matching, but makes it impossible to explore behaviors such as cruising for parking. Camargo et al. (2017), meanwhile, design their algorithm for interurban freight trips, and implicitly disallow circling through assuming the driver takes the shortest path using the set of edges that are within a given distance from the set of GPS vertices. In the context of real-time map-matching, Velaga et al. (2009) note that different scoring functions are required in different contexts, such as urban, suburban and rural areas.

The second obstacle to comparisons is that many papers present their algorithm without accompanying implementation code, or provide code in a form that cannot easily be adapted to other geographic contexts or computer platforms. While some researchers have provided standardized test traces for purposes of comparison (e.g. Newson & Krumm 2009), it is not immediately evident how to apply the results to different types of GPS data, such as traces with lower spatial accuracy or trips that involve circling for parking. From the perspective of the user, meanwhile, the application of the algorithm to their own datasets often requires substantial programming knowledge to adapt or even run the provided code, which is typically written in Java. Alternatively, users must develop their own implementation from scratch. One main exception is the GraphHopper package.[1] GraphHopper is written in Java and based on the Newson and Krumm algorithm (2009), and can be run as a command-line tool without any programming knowledge. Another exception is the Camargo et al. (2017) open-source algorithm.

[1] GraphHopper can be downloaded at https://github.com/graphhopper/map-matching. In the analysis below, we use version 0.8.2, which was the most recent version as of July 2017.

3. The pgMapMatch Algorithm

In this section, we present our implementation of the pgMapMatch map-matching algorithm. Formally, we wish to identify the true path, i.e. sequence of edges on the street network, $R = [r_1, r_2, r_3, \dots, r_N]$ given a set of GPS points $Z = [z_1, z_2, z_3, \dots, z_T]$. An edge in this instance is a directed link in the street network between two nodes (normally intersections).

Our approach uses a three-part quasi-likelihood function to rank candidate paths, comprising the following components:

- A geometric component, based on the distance between each vertex on the GPS trace and the corresponding vertex on each candidate path.
- A topological component, based on the ratio of segment lengths of the GPS trace and candidate path (a segment is defined as the path between two vertices on the GPS trace).
- A temporal component, based on the implied speed of each segment of the candidate path.

The geometric and topological components are similar to those employed elsewhere in the literature (e.g. Velaga et al. 2009; Wei et al. 2012). In terms of the algorithm, our main innovation is the addition of a temporal component, which takes into account the feasibility of moving between candidate vertices in a given amount of time. For example, a candidate path is unlikely if it implies moving at a speed of 100 km h^{-1} on streets that have a speed limit of 50 km h^{-1}. With a few exceptions (notably Lou et al. 2009), this temporal component is rarely found in the map-matching literature.

We do not consider data on the quality of the positioning fix (e.g. horizontal dilution of precision or number of satellites) nor the heading, which may improve the quality of the match, but are not consistently available in publicly available GPS datasets.

3.1 Geometric likelihood

The geometric likelihood is based on a normally distributed function of the distance of each point to candidate edges (Eq. 1), as in Newson and Krumm (2009). Each point represents a vertex on the GPS trace, as shown in Figure 1. The intuition here is that the most-likely path will lie close to each GPS vertex. The normal distribution accounts for errors due to imprecision in the GPS coordinates. It also accounts for the width of the street, given that the path (represented by the street centerline) will be a short distance from a true GPS location within the street right-of-way. The left-hand panel of Figure 2 shows the shape of the likelihood function:

$$p^G(z_t|r_i) = f^G(|\|z_t - g(r_i, z_t)\||) \qquad \text{(Eq. 1)}$$

Where: $f^G(x) \sim N(0, \sigma_z)$, and $|\|z_t - g(r_i, z_t)\||$ denotes the absolute distance between observed point z_t and the closest point to z_t on the candidate edge r_i.

In common with the other parameters introduced subsequently, we estimate σ_z through an informal iterative process, examining the likelihoods of successful and unsuccessful matches. (Due to the heterogeneity of our sample of GPS traces, the distances between the GPS vertex and map-matched position do not follow a regular distribution.) We do not attempt to optimize more formally given (i) computational limitations; (ii) the likelihood that the optimum value of σ_z is unique to our dataset, and in fact σ_z is a user-defined parameter in our open-source code; and (iii) that optimizing for σ_z could only improve the quality of our matches, and thus our evaluation underestimates the potential performance of the algorithm.

3.2 Topological likelihood

The topological likelihood is based on the ratio of the length of the GPS trace to the length of the candidate path. It is computed on a segment-by-segment basis, where a segment is defined as the path between two vertices on the GPS trace. The intuition is that the most-likely path will have a similar length to the GPS trace. The probability function is a Students' *t*-distribution with 20 degrees of freedom, as shown in Eq. 2 and the center panel of Figure 2. The *t*-distribution rather than a normal distribution is used because of its fatter tails, useful in accounting for situations where a one-way system precludes a direct route between two points.

Formally, the length of a path is given as:

$$p^T(z_{t,} z_{t-1}|r_i, r_j) = f^T\left(\max\left(0, \frac{||z_{t,} z_{t-1}||}{D\left(g(r_i, z_t), g(r_j, z_{t-1})\right)} - 1\right)\right) \qquad \text{(Eq. 2)}$$

Where $f^T(x) \sim t(0, \sigma_t, 20)$, and $D\left(g(r_i, z_t), g(r_j, z_{t-1})\right)$ gives the network distance of the candidate path segment.

If no candidate path is found between the two candidate edges, p^T is set to zero. As before, the $g()$ function denotes the closest point on the candidate edge to the GPS point. The network distance includes the portion of r_j that remains to be traveled, the portion of r_i that will be traveled on this segment, and the length of any intermediate edges. In some cases, r_i and r_j refer to the same edge. Where a path includes intermediate edges (i.e., when $i \neq j+1$), we use the Dijkstra algorithm as implemented in the pgrouting package[2] to compute the shortest route between the two candidate edges. This last step is similar to the "connected subset" procedure (Van Dijk & De Jong 2017), which uses the Dijkstra algorithm to infer the path between consecutive points.

3.3 Temporal likelihood

The temporal likelihood is based on the implied speed of the candidate path, as a fraction of the posted speed limit, computed on a segment-by-segment basis. For example, if a 20m segment of the path is traversed in one second, the implied speed is 20m per second or 72 km h^{-1}. On a street with a speed limit of 50 km h^{-1}, this is 1.44 times the speed limit. The temporal likelihood helps to reduce the chances of a match that involves improbably high speeds.

In cases where the implied speed is less than the speed limit, the probability function follows an exponential distribution. This likelihood function rewards shorter movements, but the shape of the function means that there is little difference in likelihood between a speed of 0.8 times the limit and 0.9 times the limit. Once the speed increases beyond the limit, however, we assume that higher speeds become much less likely, as reflected in the shift to a normal distribution in Eq. 3 and shown in the right-hand panel of Figure 2. As with the topological likelihood, we use the Dijkstra algorithm to compute the length of each candidate path segment in cases where intermediate edges are involved. As discussed below, even if speed limit data are not available, they can be inferred from the street classification (e.g. freeways vs primary vs secondary routes).

Formally, the temporal likelihood is calculated as:

$$p^R\left(z_{t,}z_{t-,1}, s_{t,}s_{t-1} | r_i, r_j\right) = f^R\left(\frac{S\left(g(r_i,z_t), g(r_j,z_{t-1})\right)}{s_t - s_{t-1}}\right) \qquad \text{(Eq. 3)}$$

Where $f^R(x) \sim Exponential(\lambda)$ if $x \leq 1$ else $f^R(x) \sim N(1, \sigma_R) + c$, c is a constant so the two functions are equal at $x = 1$, $S\left(g(r_i, z_t), g(r_j, z_{t-1})\right)$ indicates the seconds to traverse, at the speed limit, the road network from the closest point on candidate edge r_i to edge r_j, and $s_{t,}s_{t-1}$ give the timestamps (in seconds) of vertices s_t and s_{t-1}.

[2] pgRouting is a cross-platform, open-source extension to the PostGIS and PostgreSQL geospatial database. It can be downloaded at http://pgrouting.org. In this paper, we use version 2.4.1.

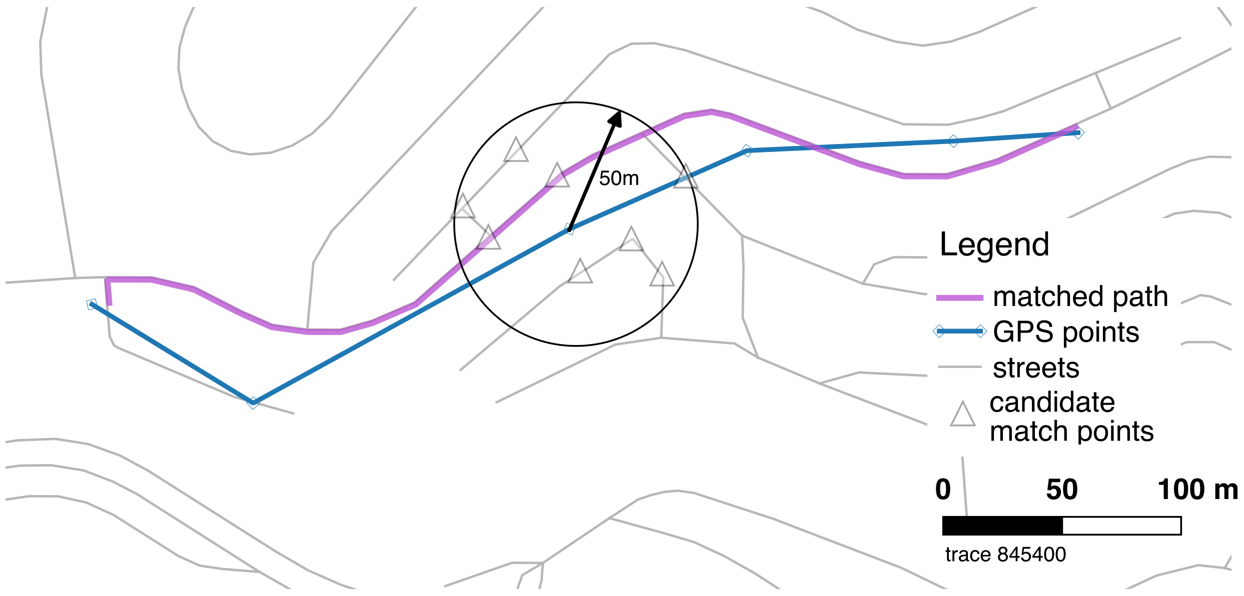

Figure 1 **Example Trace with Candidate Match Points**

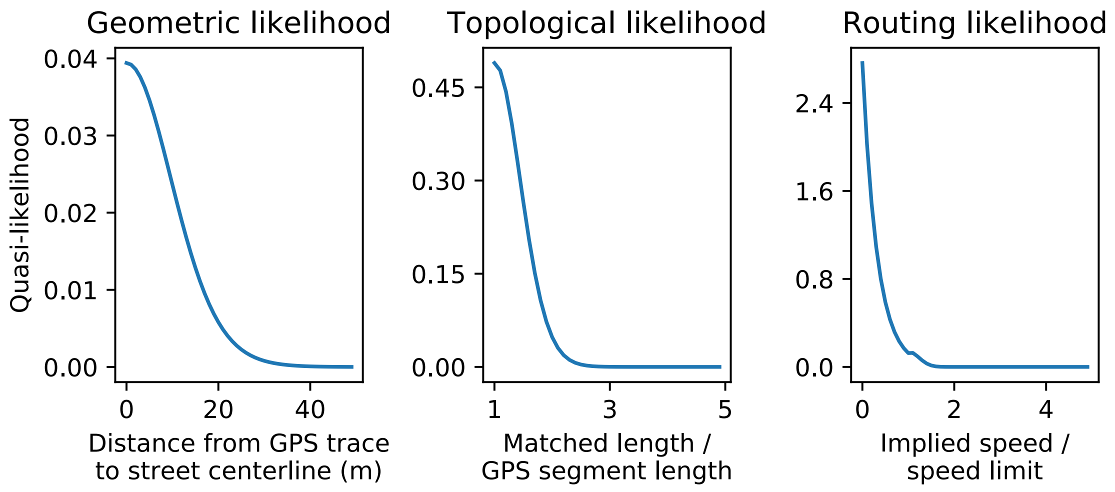

Figure 2 **Shape of Likelihood Functions**

3.4 Combined likelihood

The quasi-likelihood of each candidate path is calculated as a weighted product of the geometric, topological and temporal likelihoods (Eq. 4), and the highest-likelihood path is selected as the best match. We consider all possible paths comprised of edges within 50m of each GPS point (70m on freeways, to account for their wider right-of-way given that the distance is measured from the street centerline). The 50m radius captures almost all true edges, while providing a balance between inclusivity and computational time which roughly increases with the square of the radius.

$$p(\boldsymbol{z},\boldsymbol{s}|\boldsymbol{r}) = p^G(z_1|r_i)\prod_{t=2}^{N} p^G(z_t|r_j)p^T(z_{t,}z_{t-1}|r_k,r_l)^{w^T}\, p^R(z_{t,}z_{t-,1},s_{t,}s_{t-1}|r_m,r_n)^{w^R} \quad \text{(Eq. 4)}$$

Where: terms are defined in Eqs 1, 2 and 3

Suppose that each GPS point z_t has k_t edges within 50m, and thus $2k_t$ directed edges (given that travel can occur in either direction). Then the number of candidate paths for a trace with N points is given by the product of the options at each timestep: $\prod_{t=1}^{N} 2k_t$. For example, a one-minute trace with one-second resolution and 10 edges within 50m of each point yields 20^{60} candidate paths.

We make this problem tractable through estimating the most-likely path via a Hidden Markov Model using the Viterbi algorithm, as in (Newson & Krumm 2009). The geometric likelihood provides the probability of observing a HMM state, and the product of the topological and temporal likelihoods provides the probability of moving between candidate states.

3.5 Complicating factors

One particular complication is posed by U-turns, represented by a repeated edge with opposite directions of travel. It is necessary to allow for U-turns, given that they are a reality, particularly in urban environments (Figure 3 Panel A provides one example). However, in most cases, a "U-turn" simply reflects imprecision or noise in the GPS signal, causing a point to be closer to a perpendicular street than to the path of travel. Therefore, our algorithm penalizes U-turns through the temporal likelihood function by adding a penalty of half the time to traverse the median edge in the street network.

Another complication is posed by errant points in the GPS dataset (an example is shown in Figure 5). Some, but not all, of these points can be removed by pre-filtering the dataset. Our algorithm allows a point to be skipped at a penalty equivalent to traversing a segment at three times the speed limit, using the temporal likelihood function.

4. Assessing Performance

4.1 Data sources

We assess the performance of our algorithm against two sets of GPS traces. The first (hereafter called the "author dataset") was collected by the authors in San Francisco, California, and aims to replicate the search for parking in dense urban neighborhoods. It is discussed further in Karlin-Resnick et al. (2016). We selected eight San Francisco neighborhoods and randomly selected three addresses in each neighborhood. We then sent drivers into the field with instructions to find a parking place as near to that destination as they could. Drivers were equipped with Dash Command, an iPhone app that could log their position thus creating a GPS trace of each trip. The value in this dataset is twofold. First, the introduction of circuity due to parking search provides the challenge we sought to address. Second, the dataset is well understood making it a good control.

The second (hereafter called the "commercial dataset") was purchased from a commercial vendor, and also consists of traces with trip ends in San Francisco. Pre-cleaning involved removing vertices with an implied speed of more than 120 km h^{-1}, and eliminating traces with fewer than three vertices. Descriptive statistics are shown in Table 1.

The heterogeneity of the commercial sample in particular provides a challenging test set for any algorithm. First, the traces are derived from a range of devices, including in-vehicle navigation devices and smartphones, but there is no metadata that indicates the type of device used. Second, the resolution of the traces (i.e., the time between adjacent GPS vertices) varies from 1 second to 288 seconds, and varies within as well as between traces. Third, San Francisco provides a challenging physical environment because of the prevalence of hills, tall buildings, tunnels, and interior or underground off-street parking, all of which compromise the accuracy of a GPS signal. Fourth, in the case of the commercial dataset, a stratified sample was used, comprised of 192 traces that preliminary work indicated were hard to match or included circuitous paths, and a further 26 traces selected at random from the full commercial dataset.

In both cases, the street network data are from OpenStreetMap, and were imported to a PostGIS database using the osm2po package. Speed limit data is inferred by osm2po if not explicit in the OpenStreetMap data, based on the street classification (e.g. primary, secondary, residential). The conversion between street classification and inferred speed limit is configurable by the user. Most of the streets in our empirical test-case have inferred speed limits, and thus the performance reported here would likely be better if actual speed-limit data are present.

Table 1 Trace Characteristics

	Commercial dataset	Author dataset
Number of traces	218	71
Resolution (mean seconds between points)		
Mean	12.6	1.1
Range	1-189	1-4.3
Resolution (maximum seconds between points)		
Mean	26.2	4.3
Range	1-288	1-89
Length of trace (m)		
Mean	1164	715
Range	408-28611	250-3111

4.2 Match performance

The accuracy of pgMapMatch can be assessed by comparing the sequence of edges in the matched path to that in the true path. For the author dataset, the true path was known. For the commercial dataset, the true path was inferred visually from the GPS trace, where this was possible. A match was considered unsuccessful if even a single edge was missing from the matched path or incorrectly inferred, regardless of how well the remainder of the trace was matched.

In the author dataset, all but one of the 71 traces were successfully matched. In the commercial dataset, 180 of the 218 traces were successfully matched, 26 matches were unsuccessful, and the true path could not be inferred in 12 cases, giving a success rate of 87%. Figure 3 shows examples of a successfully matched trace (panel A), an unsuccessful match (panel B), and a match where the true path could not be inferred (panel C).

The success rate itself is not meaningful, since it depends on the quality of the GPS dataset, which in this case was selected to consist of hard-to-match traces. Thus, a low success rate is to be expected. A more useful comparison is to alternative software tools. The open-source map-matching package GraphHopper, described earlier, had a success rate of 65% on the same dataset, using an identical GPS tolerance of 50m. GraphHopper successfully matched 3 traces that our algorithm did not, and failed to match 50 of our successes. In general, these failed attempts are for traces with noise in the GPS signal, where the GraphHopper algorithm introduces erroneous loops around the block. Because our algorithm takes into account the temporal likelihood, we can avoid introducing these erroneous loops which usually involve unrealistic speeds. GraphHopper is based on the Newson and Krumm algorithm (2009); while more sophisticated algorithms are available, the relevant comparison for purposes of this paper is to a tool that has accompanying, ideally open-source, software that can be used by research and planning practitioners.

Using a 2008 Mac Pro desktop computer with 16GB RAM, the algorithm took a mean of 14 seconds to match each trace; this average is skewed by some complex traces that took 10 minutes or more to match. The median trace was matched in 5.2 seconds, compared to 0.8 seconds for GraphHopper. Our speed can be improved by resampling high-resolution traces from, for example, one second to five seconds, with no or minimal loss in accuracy. However, if speed is the primary concern, our algorithm, which prioritizes accuracy over speed, may not be the best choice.

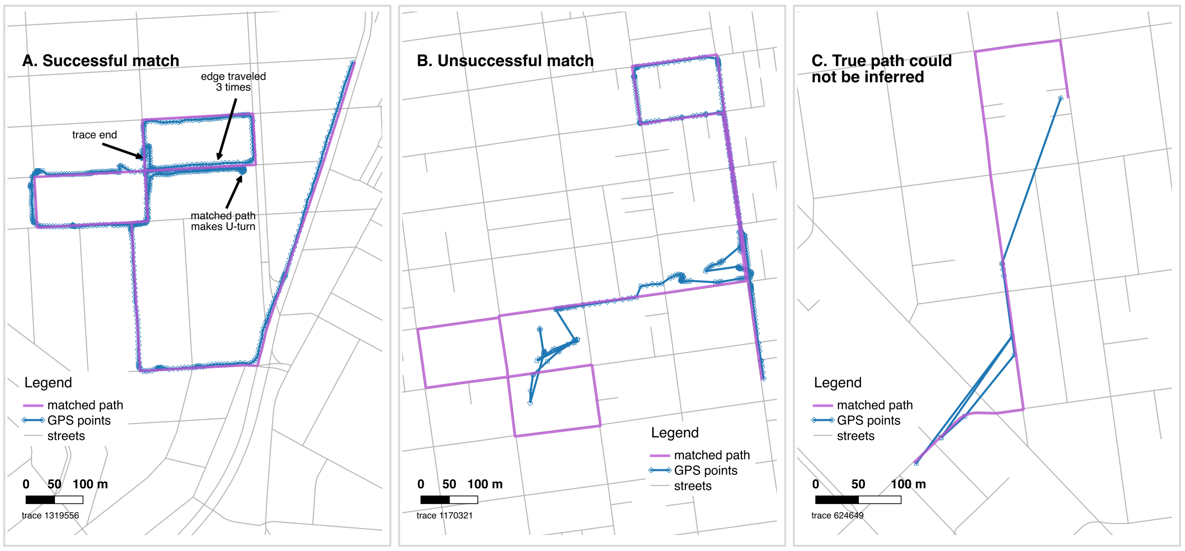

Figure 3 Examples of Matched Paths

4.3 Predicting accuracy

As discussed in the introduction, the inevitable presence of inaccurate matches poses a key challenge with map-matching of large datasets, where manual inspection of the results is infeasible. Geocoding of address data represents an analogous problem, where a geocoded location may be returned if there is uncertainty due to a misspelled address or multiple potential matches. In the case of geocoding, algorithms typically return a match score or similar indication of the likelihood of success (Karimi et al. 2004). In this subsection, we demonstrate how an analogous match score can be quantified for map matching of GPS data.

We predict the accuracy of the matches for the commercial dataset only, given that almost all of the traces in the author dataset are successfully matched. Our approach relies on a logistic regression model that estimates the probability of a successful match as a function of a set of predictor variables. Our candidate predictor variables are as follows:

- Log likelihood for each point on the most-likely path, computed according to Eqs 1-3. We have six variables in this category, corresponding to the mean and minimum for each of the geometric, topological and temporal likelihood functions.
- Ratio of the length of the GPS trace to the length of the most-likely path, and also its inverse. A ratio close to one is indicative of a successful match. This variable can be seen as a global version of the topological likelihood, but computed over the trace as a whole rather than on a segment-by-segment basis.
- Fréchet distance between the GPS trace and the most-likely path. Fréchet distance is a measure of curve similarity, and can be imagined as the minimum leash length required for a dog to walk on the GPS trace while the human walks on the most-likely path. Fréchet distance has been occasionally used in map-matching algorithms (Brakatsoulas et al. 2005); here, we employ it as a post-estimation quality measure.
- Resolution of the trace, in terms of the mean and maximum number of seconds between GPS points.

Table 2 shows the results of three regression specifications. Model 1 includes all the variables, including non-linear terms. Model 2 includes only the variable (Fréchet distance) that has a large and statistically significant effect in Model 1. Model 3 adds a measure of the temporal likelihood, which is not captured by Fréchet distance.

In all cases, we restrict the sample to traces with a mean temporal resolution of 30 seconds or less. We estimate each model on 70% of the sample, and hold back the remainder for validation purposes. Table 2 indicates the proportion of successful matches for each model in both the estimation and validation samples. A threshold probability of 0.7 is used to convert the model's predictions (which are expressed as the probability of a good match) to a binary success or fail prediction. A lower threshold probability will yield more false positives, i.e. matches that are predicted to be successful but are not in reality, while a higher threshold probability will yield more false negatives.

Overall, the models perform similarly as measured by AIC and predictive accuracy. However, in Model 1, most coefficients are estimated very imprecisely, as indicated by the number of large yet insignificant coefficients, likely because of collinearity. We choose Model 3 as our preferred specification because it is the most parsimonious model that incorporates all three dimensions: distance and topology by the Fréchet distance, and temporal by the minimum log likelihood. Intuitively, a successful match will have a matched path that lies close to the original GPS trace, and no segments with an implied speed well above the speed limit. Figure 4 indicates that Model 3 also discriminates well between successful and unsuccessful matches.

Closer inspection of both the false positives and false negatives is revealing. Figure 5 shows two false positives (incorrect matches) in the left and center panels, and one false negative (a correct match) in the right panel. In the left panel, the GPS trace backtracks on the initial edge and then heads to the top portion of the figure, following a gap in the sequence of GPS vertices, but the U-turn and backtracking is not identified by the matching algorithm. In the center panel, the opposite problem arises, with noise in the GPS signal causing the algorithm to erroneously introduce two U-turns. In the right panel, the trace is successfully matched, but the circuitous route (due to one-way streets) and the low temporal resolution of the trace means that the Fréchet distance is large. In general, similar issues—U-turns, GPS signal noise and low resolution—can explain the other false negatives and false positives.

We compute match scores for the traces in the author dataset, and find that the scores provide an excellent indication of the quality of the matches. The one trace that did not match correctly has a suitably low score (0.05). Setting the threshold probability at 0.7 yields two false negatives, with match scores of 0.61 and 0.68. The remaining traces in the author dataset all have scores greater than 0.85, with most traces having scores greater than 0.95.

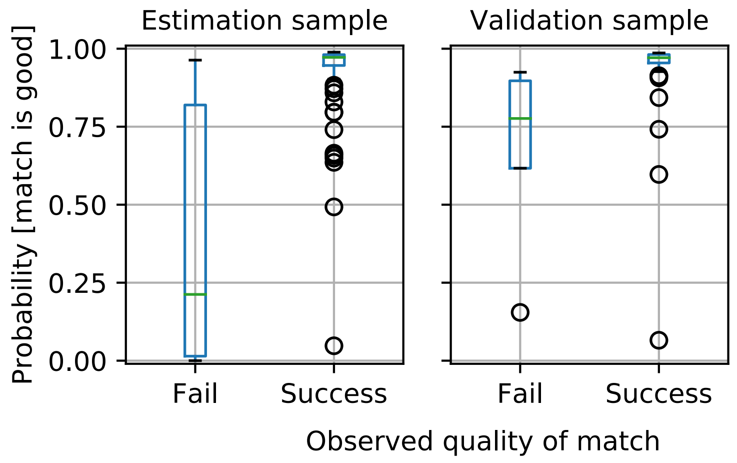

Figure 4 Predicted Match Probabilities

Table 2 Regression Estimates

	Model 1	Model 2	Model 3
Intercept	-0.102 (10.453)	4.375*** (0.667)	4.529*** (0.710)
Fréchet distance	-0.0483*** (0.0161)	-0.0463*** (0.00952)	-0.0447*** (0.00978)
Distance log likelihood (mean)	-2.035 (3.030)		
Distance log likelihood (minimum)	-0.00914 (0.309)		
Topological log likelihood (mean)	-1.913 (2.583)		
Topological log likelihood (minimum)	0.0766 (0.0470)		
Temporal log likelihood (mean)	0.0382 (0.166)		
Temporal log likelihood (minimum)	-0.000688 (0.00330)		0.00201** (0.00121)
Length ratio of matched path: trace	2.629 (7.021)		
Length ratio of trace: matched path	0.220 (3.789)		
Seconds between points (mean)	-0.0415 (0.0769)		
Seconds between points (maximum)	-0.00832 (0.0302)		
N (estimation sample)	135	135	135
AIC	69.11	61.59	61.07
Estimation sample:			
% predicted correctly	96%	90%	91%
False positives	5	7	6
False negatives	1	6	6
Validation sample:			
% predicted correctly	84%	91%	91%
False positives	4	3	3
False negatives	5	2	2

*Standard errors are in parentheses. *p<0.10 **p<0.05 ***p<0.01*

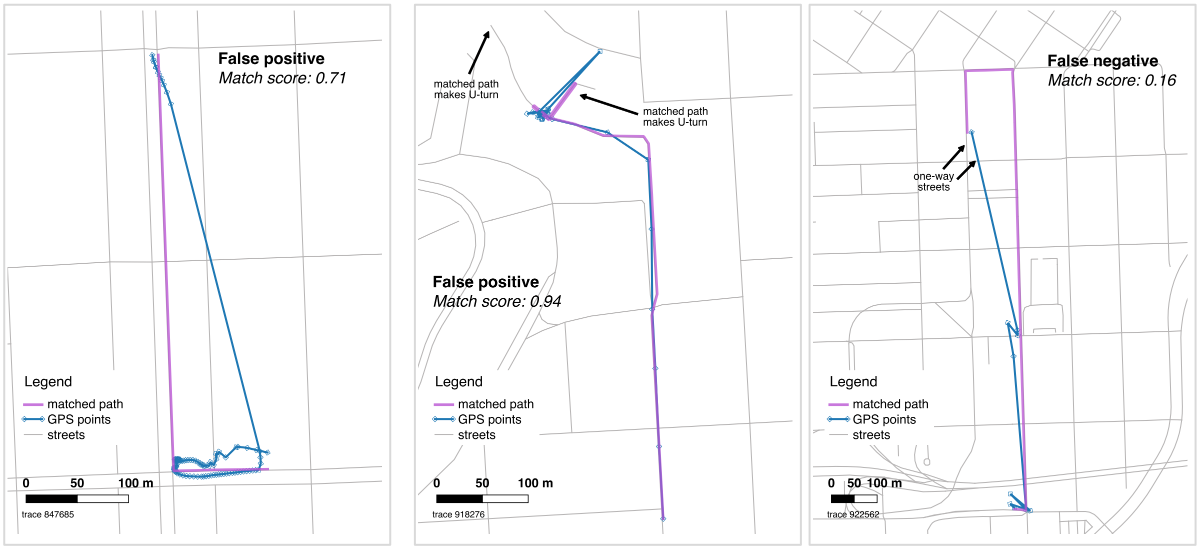

Figure 5 False Positives and a False Negative

5. User Implementation

Our Python code is available under an open-source license at www.github.com/amillb/pgMapMatch. The code enables a user to match a GPX file or a PostGIS table of GPS traces, using a simple command-line function. No Python knowledge is required unless the user wishes to adapt the code, for example to use custom likelihood functions. A configuration file allows parameters such as the distance radius within with candidate edges are considered, and the standard deviations of the likelihood functions, to be altered. The matching function returns:

- A sequence of IDs for each edge in the most-likely path
- The geometry of the most-likely path
- The probability that the match is correct, based on the logistic specification in Model 3 shown above, or another model estimated by the user

The key dependencies are the PostgreSQL database application, with PostGIS and pgrouting installed. Full instructions are provided on the GitHub site.

6. Conclusion

At first glance, map-matching of GPS data appears to be a simple problem. Visual inspection of a GPS trace usually enables a human to identify the route that was followed. However, translating this human intuition into algorithmic form—necessary if large-scale travel survey and similar datasets are to be matched—is complex. An algorithm may need to consider the distance between vertices on the GPS trace, and the feasibility of a route in terms of implied speed and topological similarity. The algorithm should also identify any points that may need to be dropped due to GPS error. Further complications are introduced by missing streets from the street-network dataset, or where a vehicle leaves the street network to travel through a parking lot or gasoline station.

While the published literature includes a range of map-matching algorithms, the number of software tools available to analysts is small. To our knowledge, none are provided as open-source code in Python—the programming language of choice for GIS analysis. In this paper, we provide a practical tool that enhances existing map-matching algorithms, particularly for poor-quality, heterogeneous traces in urban environments, which are characterized by noisy GPS signals, U-turns and circling.

Erroneous map-matching of GPS traces can bias the analysis of travel survey data, for example through suggesting that drivers circle for parking when they do not, or that they take longer routes to their destinations. While no algorithm is likely to match GPS data with 100% accuracy, our software tool mitigates this problem by introducing a regression-based match score, which indicates the probability that a trace is successfully matched. The analyst can then manually inspect traces below a certain probability threshold, or discard them altogether.

Further improvements to our algorithm could focus on faster performance, and also on identifying when points should be dropped from the GPS trace, particularly in the context of erratic signals when a vehicle enters an off-street parking facility. By providing our code under an open-source license, others can build off of the work that we present here.

Acknowledgements

This work was partially funded by Small Business Innovation Research grant DTRT5715C10013. We are indebted to Allen Greenberg for his thoughtful comments in support of our work and to Dave Perlmutter and Joshua Karlin-Resnick who were integral to accomplishing the research. All errors are ours alone.

References

Brakatsoulas, S., Pfoser, D., Salas, R. & Wenk, C., 2005. On Map-Matching Vehicle Tracking Data. In *Proceedings of the 31st International Conference on Very Large Data Bases*, pp. 853–864.

Bricka, S.G. & Bhat, C.R., 2006. Comparative Analysis of Global Positioning System–Based and Travel Survey–Based Data. *Transportation Research Record*, 1972, pp.9–20.

Bricka, S.G., Sen, S., Paleti, R. & Bhat, C.R., 2012. An analysis of the factors influencing differences in survey-reported and GPS-recorded trips. *Transportation Research Part C: Emerging Technologies*, 21(1), pp.67–88.

Camargo, P., Hong, S. & Livshits, V., 2017. Expanding the Uses of Truck GPS Data in Freight Modeling and Planning Activities. *Transportation Research Record* 2646, pp.68–76.

Van Dijk, J. & De Jong, T., 2017. Post-processing GPS-tracks in reconstructing travelled routes in a GIS-environment: network subset selection and attribute adjustment. *Annals of GIS*, 23(3), pp.203–217.

Hampshire, R., Jordon, D., Akinbola, O., Richardson, K., Weinberger, R., Millard-Ball, A. & Karlin-Resnick, J., 2016. Analysis of Parking Search Behavior with Video from Naturalistic Driving. *Transportation Research Record: Journal of the Transportation Research Board*, 2543, pp.152–158.

Karimi, H.A., Durcik, M. & Rasdorf, W., 2004. Evaluation of uncertainties associated with geocoding techniques. *Computer-Aided Civil and Infrastructure Engineering*, 19(3), pp.170–185.

Karlin-Resnick, J., Weinberger, R., Millard-Ball, A., Hampshire, R., Dykstra, T. & Lucas, K., 2016. Parking search caused congestion: what we learn from a controlled GPS example with applications to big data. Paper presented at TRB Annual Meeting, January 2016.

Li, H., Kulik, L. & Ramamohanarao, K., 2015. Robust inferences of travel paths from GPS trajectories. *International Journal of Geographical Information Science*, 29(12), pp.2194–2222.

Li, L., Quddus, M. & Zhao, L., 2013. High accuracy tightly-coupled integrity monitoring algorithm for map-matching. *Transportation Research Part C: Emerging Technologies*, 36, pp.13–26.

Lou, Y., Zhang, C., Zheng, Y., Xie, X., Wang, W. & Huang, Y., 2009. Map-matching for low-sampling-rate GPS trajectories. In *Proceedings of the 17th ACM SIGSPATIAL - GIS '09*.

Marchal, F., Hackney, J. & Axhausen, K., 2005. Efficient Map Matching of Large Global Positioning System Data Sets: Tests on Speed-Monitoring Experiment in Zürich. *Transportation Research Record*, 1935, pp.93–100.

Millard-Ball, A., Weinberger, R. & Hampshire, R., 2014. Is the curb 80% full or 20% empty? Assessing the impacts of San Francisco's parking experiment. *Transportation Research Part A: Policy and Practice*, 63, pp.76–92.

Newson, P. & Krumm, J., 2009. Hidden Markov map matching through noise and sparseness. In *Proceedings of the 17th ACM SIGSPATIAL - GIS '09*.

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

Schuessler, N. & Axhausen, K.W., 2009. *Map-matching of GPS traces on high-resolution navigation networks using the Multiple Hypothesis Technique (MHT)*, Working Paper, ETH Zurich. http://www.baug.ethz.ch/ivt/ivt/vpl/publications/reports/ab568.pdf.

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

Wei, H., Wang, Y., Forman, G., Zhu, Y. & Guan, H., 2012. Fast Viterbi Map Matching with Tunable Weight Functions. In *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*. Redondo Beach, CA, pp. 613–616.