Unambiguous, Real-Time and Accurate Map Matching for Multiple Sensing Sources

Mohamed Amine Falek*, Cristel Pelsser†, Antoine Gallais†, Sébastien Julien* and Fabrice Théoleyre†
* ICube lab, CNRS/ University of Strasbourg, 67400 Illkirch, France - Email: {lastname}@unistra.fr
† T&S Group, 4 rue de Dublin, 67300 Schiltigheim, France - Email: {a.falek,s.julien}@technologyandstrategy.com

***Abstract*—Smart Cities need real time information to improve the efficiency of their transportation systems. In particular, crowd sensing may help to identify the current speed for each street, the congested areas, etc. In this context, map matching techniques are required to map a sequence of GPS waypoints into a set of streets on a common map. Unfortunately, most map matching approaches are probabilistic. We propose rather an unambiguous algorithm, able to identify all the possible paths that match a given sequence of waypoints. We need an unambiguous identification for each waypoints set. For instance, the actual speed should be assigned to the correct set of streets, without error. To identify all the possible streets, we construct the set of candidates iteratively. We identify all the edge candidates around each waypoint, and reconstruct all the possible sub-routes that connect them. We then verify a set of constraints, to eliminate impossible routes. The road segments common to all computed routes form an unambiguous match. We evaluate the matching ratio of our technique on real city maps (London, Paris and Luxembourg). We also validate our approach with a real GPS trace in Seattle.**

***Index Terms*—Smart Cities; unambiguous map matching; crowd sensing; GPS traces; merging heterogeneous datasets**

I. Introduction

The concept of Smart City refers to modern cities relying on ICT for increased efficiency [1]. A myriad of deployed devices allow cities to perform traffic light management, traffic congestion avoidance and smart transportation. Based on the produced data, the smart city services can exploit a real time road map (e.g. speed, congestion level). In particular, dynamic route planning needs real time traffic conditions to guide vehicles and users through the optimal route [2]. Yet, deploying thousands of traffic sensors is particularly expensive.

Participatory sensing [3] relies on individual bodies to collect a large dataset of measurements. Unfortunately, each participant provides independent sequences of measurements, which have to be merged to construct a consistent, real time view of the whole road or street network. A map matching algorithm aims to map all these distinct traces onto a common base map, so that we create a global dataset from a set of individual traces [4]. Openstreetmap[1] is commonly used as a base map [5].

Map matching algorithms exploit an ordered sequence of waypoints (geographical coordinates) obtained with a cellular or GPS positioning system. Thus, each point is riddled with measurement errors. Typically, GPS-enabled smartphones provide an accuracy of 4.9 m in ideal conditions (open sky) [6]. Most map matching algorithms identify the most probable path corresponding to a given individual sequence. However, trajectory planning requires a precise mapping of congestion information on the map for highly accurate travel time estimation.

Because acquiring a GPS location is very energy consuming, participatory sensing shall benefit from a low sampling rate GPS trace. However, reducing this sampling rate negatively impacts the accuracy [7], thus jeopardizing the identification of the most probable route for a given trace. For a dense map, multiple *similar* paths may exist between two data points, and no argument can fairly differentiate the actual path from its alternative.

In this paper, we propose an **unambiguous map matching**, to identify all the possible paths corresponding to a sequence of waypoints. Indeed, a probabilistic method is to our mind insufficient: identifying the wrong route implies that a bias is created. For instance, an urban planner may need to count the actual number of vehicles for a specific street. Here we choose not to resolve ambiguities. We rather not consider segments between waypoints that we can not match definitely.

To the best of our knowledge, we propose the first unambiguous map matching method. The contributions of this paper are:

1) we propose a map matching algorithm for sparse traces, enabling real-time mapping with a low computation time;
2) instead of a probabilistic method, we adopt here an unambiguous approach: we consider that the map matching has partially failed when several path candidates are obtained, and we thus identify the subroute which was used *for sure* by the trace ;
3) we thoroughly evaluate our solution, on emulated GPS traces on the London, Paris and Luxembourg maps. We always identify the initial path (no false negative), while reaching a small set of candidates (false positive), with a reasonable sampling period (under 50s).
4) we illustrate the performance of our algorithm on real GPS traces. We show its robustness to increasing sampling rates in the face of real-life GPS measurement errors.

II. Related Work

Map matching is the process of identifying the route taken by an entity, by only analyzing a sequence of its geolocated measurements. Typically, a user or a device activates its GPS,

[1] http://www.openstreetmap.org

and takes a measurement periodically to reconstruct its path (i.e. on or off-line). Map matching is extensively used for crowd-sensing applications in smart cities. For instance, Li *et al.* [8] infer traffic conditions after using map matching on a large dataset of GPS traces. In particular, the authors extract a repetitive pattern, where traffic peaks are mechanically highly probable during the peak hours.

A. Trajectory completion

Inferring road maps from sparse GPS traces has received much attention in the literature. Liu *et al.* [9] compared the accuracy of map inference from highly instrumented (and expensive) measurements and from large-scale low-cost GPS traces. They confirmed that sampling intervals should be kept short to provide high accuracy.

Li *et al.* [10] extract first a skeleton from the data points, and construct a junction network to unify all the trajectories. This way, each individual trajectory can be mapped to the same, common base map. We do not have the same objective in this paper: we have a common base map and our objective is rather to map individual traces to an existing map, serving as a ground-truth.

B. Map matching

Existing map matching approaches are classified into the geometric, topological, probabilistic, and advanced methods [11].

Geometric solutions only consider geometric properties (e.g. Euclidean distances) while topological approaches also consider the cartography / connectivity information. Yuan *et al.* [12] construct a candidate graph, with all the edges close to the GPS measures. Weights in the graph are chosen proportional to their Euclidean distance from the data point. A voting scheme allows the algorithm to identify the most probable candidate. They evaluated the accuracy of their technique on the traces of 1,000 taxis in Beijing, which have heterogeneous sampling rates (i.e. up to a few minutes), reaching a maximum of 73% correct matching for sampling intervals of 3.5 minutes.

Probabilistic methods identify an *error region*, derived from the GPS inaccuracy. Then, the device is mapped to the roads present in this error region. Szwed *et al.* [13] use a hidden Markov model to identify the most probable mapped path when using multiple sensors (e.g. GPS, Wi-Fi). Expectedly, from the experiments conducted for 20 GPS traces only (modified by introduction of noise and/or downsampled), the longer and more noisy the trace (i.e. as anticipated for real-world scenarios), the less reliable the algorithm. Hu *et al.* [14] also use the speed and direction information to prune the set of path candidates.

Chen *et al.* [15] analyzed a multimodal dataset, with a collection of different sensors (GPS, Bluetooth, accelerometer). They propose to use map matching to infer the mode of transportation, in order to split a trip into a set of individual mono-modal paths. Even cellular positioning data are used with a hidden Markov Model for map matching [16], with a much higher location inaccuracy.

In this paper, we rather aim to identify **all** the paths which respect the time delay constraint, and which may be candidate for the given set of GPS data points. Our proposal enters in the class of probabilistic methods, targeting high accuracy even at low sampling rates.

TABLE I: Notation used in the paper

Symbol	Signification
$\mathcal{E}_{gps}$	maximum GPS error for a data measurement
σ_{gps}	standard deviation of the GPS error (Gaussian distribution)
Δ_{speed}	security coefficient of speed excess authorized for the trace
$Z = \{Z_i\}_{i\in[0,\lvert Z\rvert]}$	GPS trace, with $\lvert Z\rvert$ data points
$tail(e)$ / $head(e)$	tail (respectively head) of the edge e
$C(e)$	cost (in seconds) to join $tail(e)$ and $head(e)$
$Cand(Z_i)$	set of candidate edges for the measurement point Z_i
$Outgoing(Z_i)$	outgoing edge candidates for the measurement point Z_i
$Incoming(Z_i)$	incoming edge candidates for the measurement point Z_i
$\mathcal{R} = \{e_i\}_{i\in[0,k]}$	a valid route from the edge e_0 to e_k
$subRoute(e_i, e_j)$	valid subroute between the candidate edges e_i and e_j
$D(Z_i)$	disk centered on Z_i with a radius $\mathcal{E}_{gps}$
δ_{cell}	size of a cell in the grid in degrees of latitude and longitude

III. Models and Assumptions

We analyze the case of a vehicle equipped with a GPS measuring device and moving on a given road network. The sampling frequency of the GPS can be adjusted to generate these measurements (i.e. latitude and longitude coordinates) at a constant rate T. A GPS trace is a sequence of such measurements with T seconds intervals from the start until the end of the journey. We use the term *true route* to refer to the real route used by the vehicle during its journey. Our goal is to reconstruct the *true route* from the issued trace.

Let us model the road network with a directed graph $G(V, E)$, V denoting the set of vertices, and E the set of edges. Each road intersection corresponds to a vertex in the graph. Each road segment is an edge between the corresponding vertices. To each vertex is associated a geographical coordinate, and a road segment may be subdivided into small segments to preserve the geographical *shape* of the road. The weight of an edge is the time (in seconds) elapsed to join the two corresponding vertices, at the speed limit of the road section. We call a couple of two consecutive data points, the tail and the head vertices, respectively. Besides, we denote by `path` a sequence of connected vertices and edges in the map.

We use here Openstreetmap as base map, since this opendata project provides a free access to a huge and accurate collection of road networks.

IV. Map Matching Algorithm

Our objective is to map a `trace` to a set of *valid routes* that are proven to be feasible for the journey. Then, we extract the set S of all shared edges among valid routes. We prove that $\forall e \in S,\ e \in true\ route$. We use the notation described in

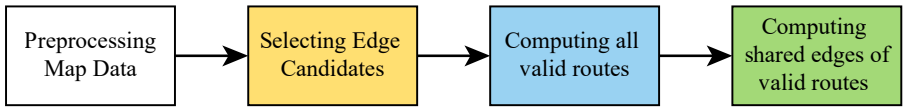


Fig. 1: Map Matching algorithm process

table I. Our map matching algorithm relies on four pipelined steps (Figure 1):

1) **Preprocessing map data:** we first organize the map into a regular grid to accelerate the identification of edges close to a given location;
2) **Selecting edge candidates for each measurement:** each measurement Z_i in the trace allows us to approximate the actual position P_i to a disk centered on Z_i with a radius $\mathcal{E}_{gps}$ (which is fixed to consider the measurement error). Then, we identify all the *edge candidates* ($Cand(Z_i)$) that correspond to Z_i. More precisely, any edge that crosses the boundary of the disk is an edge candidate. We make the distinction between the incoming (their tail is outside the disk) and the outgoing (their head is outside the disk) edges.
3) **Identifying valid routes from the set of edge candidates:** we then have to compute routes, which include one edge candidate for each Z_i, and which respect a time delay constraint. For each pair of consecutive measurements Z_i and Z_{i+1}, a Breadth First Search (BFS) calculates all the possible subroutes i.e a sequence of edges that start with an incoming edge candidate for Z_i and finish with an outgoing edge candidate of Z_{i+1}. The cost of a subroute must respect the elapsed time between measurements Z_i and Z_{i+1}. Finally, the different subroutes are assembled and we verify the end-to-end time delay constraint.
4) **Extracting common segments for all the possible routes:** we identify the edges which are **for sure** on the journey, constituting a set of *common segments*. The longer the segments, the more accurate our algorithm: ideally one single segment would cover the whole route meaning that a single option matches our constraints.

A. Pre-processing

Identifying the edges close to a given location would require to fetch the whole graph, and to compute the distance of each of them. Thus, to accelerate the computation, we organized the map into geographical cells, i.e. rectangles of δ_{cell} degrees of width and height. We compute for each edge its set of cells by using a modified version of the Bresenham's algorithm [17]. More precisely, an edge belongs to a cell if both of its end-vertices lie inside the cell, or if the edge crosses its boundaries. When searching for all the edges at a maximum distance of a specific location, we just have to scan all the edges belonging to the cells around the location.

B. Selecting edge candidates for each measurement

GPS typically follows a Gaussian distribution with standard deviation σ_{gps} [18]. Thus, the true position of the vehicle may be located anywhere in a disk centered on the measurement (Z_i) with the radius:

$$\mathcal{E}_{gps} = 3 \times \sigma_{gps} \tag{1}$$

which accounts for a 99.9 % certainty that the disk area contains the true position P_i.

An edge candidate is any edge that allows accessing and/or exiting the area that contains the true position of the vehicle. In other words, any edge that crosses the disk centered at Z_i. Thus, Z_i and the closest point of an edge candidate are separated by a distance smaller than $\mathcal{E}_{gps}$. Let us denote by $d(u, v)$ the Euclidean distance between the points u and v. An edge e is a candidate if one of the following conditions holds:

1) the head is outside the disk: e in an outgoing edge candidate;

$$(d(tail(e), Z_i) \leq \mathcal{E}_{gps}) \quad \wedge \quad (d(head(e), Z_i) > \mathcal{E}_{gps}) \tag{2}$$

2) the tail is outside the disk: e is an incoming edge candidate;

$$(d(head(e), Z_i) \leq \mathcal{E}_{gps}) \quad \wedge \quad (d(tail(e), Z_i) > \mathcal{E}_{gps}) \tag{3}$$

3) both the head and tails are outside the disk but the edge intersects the disk: e is both an incoming and outgoing edge candidate.

$$(d(proj(e, Z_i), Z_i) < \mathcal{E}_{gps}) \ \wedge \ (d(tail(e), Z_i) > \mathcal{E}_{gps}) \ \wedge \ (d(head(e), Z_i) > \mathcal{E}_{gps}) \tag{4}$$

where $tail(e)$ and $head(e)$ denote the tail and head of the edge e respectively, and $proj(e, Z_i)$ denotes the projection of Z_i on the edge e.

C. Constructing valid routes from a list of edge candidates

We have now to infer the global route which corresponds to a sequence of edge candidates, and which also respect the time delay constraint. Formally, a route $\mathcal{R}$ is an ordered sequence of edges $E = \{e_i\}_{i \in [0,k]}$ such that $\forall i \in [0, k-1], head(e_i) = tail(e_{i+1})$.

We make a distinction between:

- A True route: this corresponds to the set of edges which were followed during the journey;
- Valid routes: all the possible routes which respect the constraints. The true route is included in the set of valid routes, thus preventing us from a false negative.

We propose an iterative approach, where the route is grown step by step, appending subroutes from one measurement to the next. More formally, a subroute is a sequence of consecutive edges such that:

$$subRoute(e_i, e_{i+1}) = \{e_k \quad | \, head(e_k) = tail(e_{k+1})\}$$
$$s.t.\ e_i \in Incoming(Z_i) \wedge e_{i+1} \in Outgoing(Z_{i+1}) \quad (5)$$

Remark 1: For the first pair of measurements (Z_0, Z_1), a subroute begins with an outgoing edge candidate of Z_0. This exception is due to the fact that the first measurement Z_0 might not possess any incoming edge candidate, since the journey has started inside the disk, possibly in a dead-end street.

To compute all the *valid routes*, we propose an iterative two step approach:

1) calculating subroutes, for a pair of consecutive measurements. We also verify that they are valid concerning the timestamps (time delay constraint). Typically, in Figure 2, the subroute $(a_3, a_4, a_5, a_6, a_7)$ is a valid subroute from Z_1 to Z_2.
2) appending the corresponding subroute to all the already computed valid routes, and then verifying the end-to-end delay constraint. In Figure 2, the subroute $(a_3, a_4, a_5, a_6, a_7)$ can be combined to the valid route (a_2, a_3), creating a new route $(a_2, a_3, a_4, a_5, a_6, a_7)$ from Z_0 to Z_2.
   Remark 2: We later retract the new route until we reach an incoming edge of measurement Z_2. In our example we would obtain $(a_2, a_3, a_4, a_5, a_6)$ after retracting the route. We explain why in the next subsection.

When the algorithm reaches the last measurement Z_N, all the valid routes are identified.

1) Constructing the subroutes: The objective now is to construct all the possible subroutes between two edge candidates. We have to verify that a subroute is *valid*, i.e. the vehicle is able to join the two measurement points without exceeding the speed limit.

To identify all the valid subroutes, we apply a Breadth First Search (BFS) approach starting from each incoming edge candidate for the measurement Z_i, and stopping after either of the two conditions:

1) an outgoing edge candidate for the measurement Z_{i+1} is visited (a valid subroute is discovered). We will explain below why we have to stop at an *outgoing* edge candidate;
2) the cost of the subroute exceeds the delay constraint (the exploration has to stop, backtracking to the other edges to discover). We will explain in the next subsection how the cost of a subroute is actually computed.

The BFS algorithm evaluates all the subroutes in the graph starting from every incoming edge e of Z_i, such that e corresponds to the end of a route that was calculated earlier. All the neighbors are recursively explored, their id being pushed on top of a stack. In parallel, our algorithm maintains the minimum time cost to walk the explored subroute to verify the delay constraint. As soon as an outgoing edge for the next measurement point is scanned, the stack is saved: a new subroute was discovered.

When either a subroute has been discovered, or the delay constraint is violated, the BFS backtracks to the previous neighbor to explore. We implement a simple stack of unexplored neighbors since we have to make a complete exploration to detect *all* the possible subroutes.

Let us consider the example in Figure 2. A valid route has been constructed up to the head of the edge c_1. We apply the BFS strategy, and we obtain recursively the subroutes $(c_1, c_2, a_5, a_6, a_7)$ and $(c_1, d_1, e_1, e_2, a_7)$. Assuming that the subroutes $(c_1, c_2, a_5, d_3, d_4)$, $(c_1, d_1, d_2, d_3, d_4)$ and $(c_1, d_1, d_2, a_6, a_7)$ have been discarded because they violated the time constraints.

Remark 3: Please note that we stop the BFS only when scanning an **outgoing** edge for Z_{i+1}. If the two measurement points are very close because of a small sampling period or large measurement inaccuracies, the two corresponding disks may overlap, preventing the algorithm from finding a subroute that ends with an incoming edge at Z_{i+1}.In Figure 2, the disks of Z_2 and Z_3 overlap. In particular, the incoming edge a_6 of Z_2 for the valid route is *after* the incoming edge a_5 of Z_3. Thus, we cannot compute a subroute that begins and ends with incoming edges at Z_2 and Z_3 respectively. On the contrary, we can safely stop at the first outgoing edge of Z_3, which is always located *after* the incoming edge of Z_2, as demonstrated below.

Let Z_{min} be the minimum index such that the disks centered on Z_0 and Z_{min} respectively do not overlap. Similarly, let Z_{max} be the maximum index such that the disks centered on $Z_{|\{Z_i\}|-1}$ and Z_{max} do not overlap. For example, in figure 2, Z_{min} corresponds to Z_1 and Z_{max} corresponds to Z_1 aswell.

Lemma 1: For any pair of consecutive measurements $(Z_i, Z_{i+1})_{i \in [Z_{min}, Z_{max}]}$, there exists a sequence of connected edges such that the first is an incoming edge candidate of Z_i and the last one is an outgoing edge candidate of Z_{i+1}

Proof 1: The true route corresponds to a sequence of edges $\{e_k\}_{k \in [0, |\mathcal{R}|-1]}$ in the graph. By construction, at least one edge of the true route is at a maximum distance of $\mathcal{E}_{gps}$ from Z_i $(min < i < max)$. Let $E_i = \{e_k\}$ denote this set, and $kmin$ be the minimum index. e_{kmin} is an incoming edge for Z_i.

Let us prove it by contradiction. If e_{kmin} is not an incoming edge, e_{kmin-1} is also in the set E_i, which is impossible since $kmin$ is minimum. Besides, $kmin > 0$, since at least one edge is in the disk centered on Z_0 and not in the set E_i, by construction of Z_{min}. Thus, e_{kmin} has to be an incoming edge for Z_i.

We demonstrate similarly that an edge e_{kmax} of the true route is an outgoing edge for Z_{i+1}. Moreover, the incoming edge for Z_i has been traversed before the timestamp T_i corresponding to the measurement Z_i (first arrival in the disk). Inversely, the outgoing edge has been visited after T_{i+1}, and by definition, $T_i < T_{i+1}$, thus $kmin \leq kmax$.

The sequence of the edges in the true route, from e_{kmin} to e_{kmax}, forms a valid subroute, which will be scanned by the BFS.

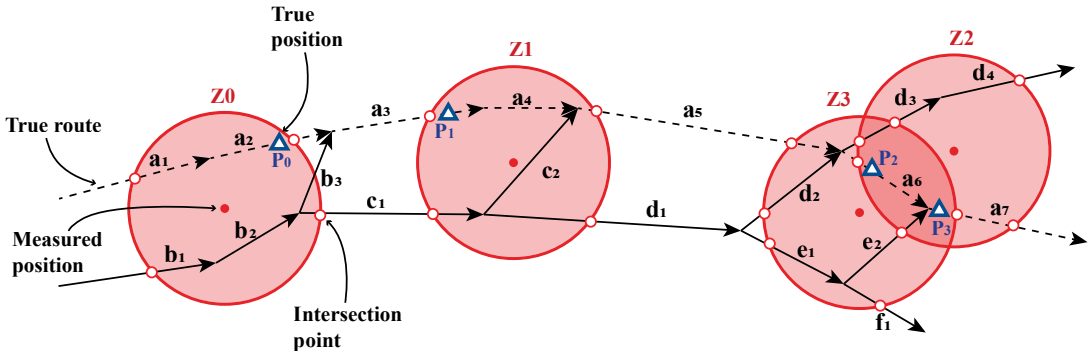


Fig. 2: Example illustrating the process of appending subroutes.

Remark 4: We cannot assume that the true route *enters* in the first disk, and *exits* from the last one. Thus, we scan all the edges inside or crossing the first and last disks. This exhaustive search increases slightly the computation time, but only for the first and last measurement points.

2) Verifying the cost of a subroute: We have to compute the minimum time to travel a corresponding subroute. Thus, we define the cost for an edge e as the time required to join the tail and the head vertices with the speed limit of the corresponding edge.

$$\forall e \in E, C(e) = C\left(head(e), tail(e)\right) = \frac{d(head(e), tail(e))}{speed(e) * \Delta_{speed}} \tag{6}$$

with $speed(e)$ the speed limit associated to the edge e in the map, and Δ_{speed} a safety margin to take into account speed excesses. By extension, the cost of a subroute corresponds to the sum of the costs of its edges.

Because the exact location corresponding to a measurement is by definition imprecise, we do not know the exact distance travelled between two GPS measurements. We know that the minimum distance is the distance between the intersection points of the outgoing edge candidate of Z_i and the incoming edge candidate of Z_{i+1} with the disks $D(Z_i)$ and $D(Z_{i+1})$ respectively (positions P_0 and P_1 in Figure 2). Thus, we need to calculate the cost for the section of the subroute that lies outside the area enclosed by the disks $D(Z_i)$ and $D(Z_{i+1})$. This cost has to be below the time between two GPS samples ($T_{sampling}$) for the subroute to be feasible.

In figure 2, for the subroute (a_2, a_3, a_4, a_5), this cost corresponds to the sum of costs from the intersection of a_2 with $D(Z_0)$ to the intersection of a_3 with $D(Z_1)$.

Lemma 2: The true route always corresponds to at least one valid subroute.

Proof 2: Let us consider the true route defined by a sequence of edges $\mathcal{R} = \{e_i\}_{i \in [0, |\mathcal{R}|]}$. Let P_{out} (resp. P_{in}) be the intersection of the last outgoing (resp. incoming) edge in $\mathcal{R}$ with the disk centered on Z_i (resp. Z_{i+1}). P_{out} is by definition the latest possible position of the vehicle along the true route at the timestamp T_i, which leads to the measurement P_i. Similarly, P_{in} is the earliest possible position for the timestamp T_{i+1}. Thus, $\sum_{e_k \in (P_{in}, P_{out})} C(e) \leq T_{i+1} - T_i = T_{sampling}$. In conclusion, the true route will not be discarded by this condition, and is discovered with the BFS (lemma 1).

3) Appending a subroute to a valid route: We now have to assemble the subroute with the set of valid routes to extend them until the last measurement point. Because we start the next subroute from an incoming edge (sequentiality constraint), we cannot assemble the subroute directly. We thus propose to:

1) prune all the edges in the subroute until (i) an incoming edge for Z_{i+1} is scanned, **or** (ii) the end of a valid route is detected. The second condition holds to still consider the case for which the two disks overlap, and stopping earlier the backtracking is more efficient;
2) append this pruned subroute at the end of all the existing valid routes stopping at the corresponding edge candidate;
3) verify the end-to-end delay constraint for each valid route created in this way.

Then, the next measurement point will be considered.

We have to verify that the subroute respects the time constraint:

$$\sum_{e \in s} C(e) \leq i * T_{sampling} \tag{7}$$

with s being the sequence of edges in the valid route, from the last outgoing edge of Z_0 until the first incoming edge of Z_i.

Remark 5: The first and last measurements in the trace Z_0 and $Z_{|Z|}$ respectively, are treated separately because neither can we guarantee that Z_0 possesses incoming edge candidates nor $Z_{|Z|}$ possesses outgoing edge candidates. Indeed, the journey could start and terminate at a dead-end street.

D. Computing shared road segments in valid routes

Once all valid routes for the journey are established, we aim to compute the subset which is common to all the routes: we are sure that the vehicle has followed these specific edges.

More formally, let VR denote the set of valid routes. We compute the set of edges $\mathcal{S}$ such that:

$$\forall e \in \mathcal{S}, \quad \forall r \in VR, \quad e \in r \tag{8}$$

We use the term *correct matching ratio* (cmr) to define the ratio of all edges in $\mathcal{S}$ (weighed by their length) to the total length of the *true route*:

$$cmr = \frac{\sum_{e \in \mathcal{S}} d(e)}{\sum_{e \in \mathcal{T}} d(e)} \tag{9}$$

with $\mathcal{T}$ being the *true route*.

V. Performance Evaluation

We carried two different experiments to evaluate our algorithm. In the first experiment, we emulated GPS traces and used OpenstreetMap[2] as a common map. In the second experiment, we used a real GPS trace from a publicly available dataset [19].

A. Emulated GPS traces

We first evaluate the accuracy of our unambiguous map matching method by emulating GPS traces. Prior to using the OSM data for our experiment, we cleaned it from duplicate entries and missing information. Specifically, we remove all the isolated vertices, i.e. not part of the largest connected component. These private or disconnected roads constitute less than 5% of the vertices of the graph. Finally, we precompute the grids for the OSM map (a tile corresponds to 0.001°).

TABLE II: Dataset

Road Network	**London**	**Paris**	**Luxembourg**
Number of vertices	836,271	169,879	27,306
Number of edges	1,637,300	284,246	50,607
Degree	8	6	5

We used 3 different cities (Tab. II) with different characteristics to evaluate the robustness of our method. For each city, we generated 500 GPS traces in the following way:

1) we pick randomly two waypoints (A and B), located approximatively 8km apart;
2) we compute the shortest path from A to B. We emulate the movement of a vehicle, by following the path with the speed limit of each segment, as given by openstreetmap. A sample is saved every 1 second;
3) for each waypoint, we emulate a real GPS measurement, i.e. the actual location with an additionnal error caused by the GPS system. The location inaccuracy follows a Gaussian distribution with a standard deviation of 4.07m [18].
4) we finally subsample the GPS trace with a period comprised between 1 and 50 seconds. We can thus compare the different sampling rates: they correspond to the same journey.

We consider here only regular sampling periods to more easily interpret the results. Indeed, an irregular sampling would correspond to a mix of different cases, making the results more difficult to interpret. Tab. III summarizes our evaluation setup.

We measured the following metrics:

[2]https://www.openstreetmap.org/

TABLE III: Evaluation setup

σ_{gps}	4.07 m (standard deviation of the GPS chipset)
Δ_{speed}	1.2 (safety margin for the speed limit, cf. section IV-C2)
δ_{cell}	0.001° (size of the grid for the precomputation)
T_{samp}	$\in [1, 50]$ (sampling rate of the GPS trace)
CPU	Core i7-4600U CPU @ 2.10 GHz
memory	16 GB RAM
software	Java 8, Eclipse IDE Oxygen.2 Release (4.7.2)

correct matching ratio: we compute the weighted subset of edges which are common to all the valid routes (cf. section IV-D). $cmr = 1$ corresponds to a perfect match (i.e. there is only one valid route which happens to be the true route). Similarly, $cmr = 0$ means that even though we did not discard the *true route*, there exists at least one valid route that does not share any edge with the rest of the valid routes.

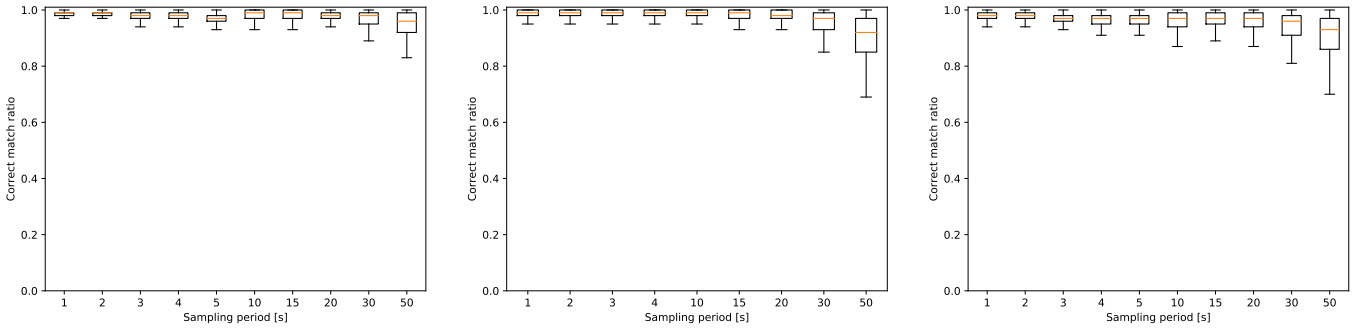
computation time: time required to compute the set of valid routes.

We then analysed the effect of road junctions density (i.e. number of edges) around a specific waypoint on the correct matching ratio. Figure 3 illustrates the correct matching ratio of London, Paris and Luxembourg, for sampling periods ranging from 1 to 50 seconds. The true route is always in the set of valid routes. In addition, for very small sampling periods, it is often the only valid route. Consequently, we are able to identify the whole true route. The accuracy decreases for higher sampling periods. Above a sampling of 15 to 20 seconds (depending on the city), some measurement points are far from each other, and several valid subroutes exist to join them.

Our algorithm is unambiguous and does not pick randomly one of these possible routes. However, even for very large sampling rates (one GPS measurement every 50 s), more than 95% of the true route is accurately identified. In the worst case, for some cities, such as London, we are able to identify 80% of the true route. In conclusion, our unambiguous algorithm is very efficient to exploit sparse GPS measurements.

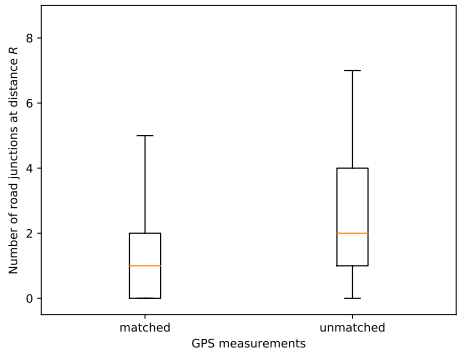
We also identify the metrics that affect the success or failure of the matching process. We suspect that the number of road junctions near a given measurement Z_i, increases the odds of computing more subroutes which decreases the probability of matching Z_i to a single route i.e the *true route*. For London and Luxembourg experiments, we measured the number of road junctions at a distance $d < 4 \times \mathcal{E}_{gps}$ of matched and unmatched measurements. For each experiment we matched approximately 12,000 measurements distributed over a dataset of 500 journeys. We observe in figure 4 that there is indeed more road junctions near unmatched measurements. Though, we still need to thouroughly test this hypothesis to be sure it holds for various road networks.

Figure 5b illustrates the computation time (for London experiment) when varying the sampling period. Whatever the conditions, we identify accurately the true route in less than 80ms. Our algorithm is thus efficient for online strategies, where the reactiveness is of primary importance. We can observe in figure 5a the average execution time for the three main

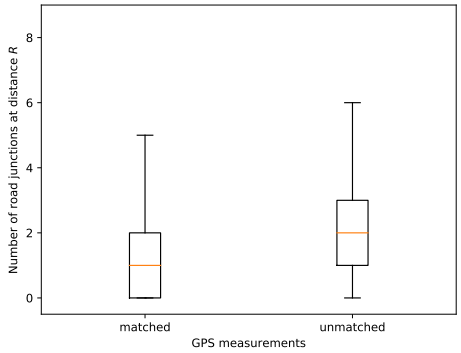


(a) London (b) Paris (c) Luxembourg

Fig. 3: Correct matching ratio for London, Paris, Luxembourg road networks.

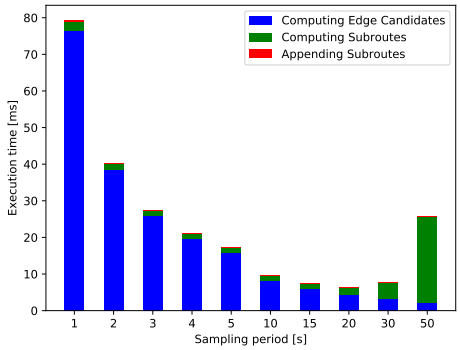


(a) London

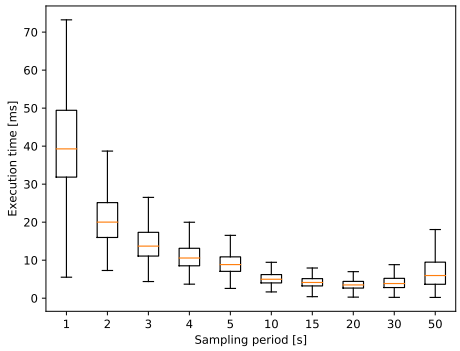


(b) Luxembourg

Fig. 4: Road junction vertices density near matched and unmatched measurements ($R = 4 \times \mathcal{E}_{gps}$).



(a) Average execution time of main methods of map matching algorithm



(b) Execution time distribution

Fig. 5: Performance of map matching algorithm based on execution time.

methods of the algorithm. For low sampling periods, we have many measurement points to consider, which increases the computation time for computing all edge candidates. Inversely, large sampling periods imply a larger computation time for the BFS, to identify all the possible subroutes, and verifying their cost. The computation time remains below 20ms for a sampling period of 5 s, which corresponds also to a close to perfect correct matching ratio.

These results were obtained with emulated GPS traces, whose parameters (e.g., measurement uncertainty, car speed) remained fixed throughout the emulation. We thus aimed at validating our proposition on real-world traces, whose dynamics may impact our results.

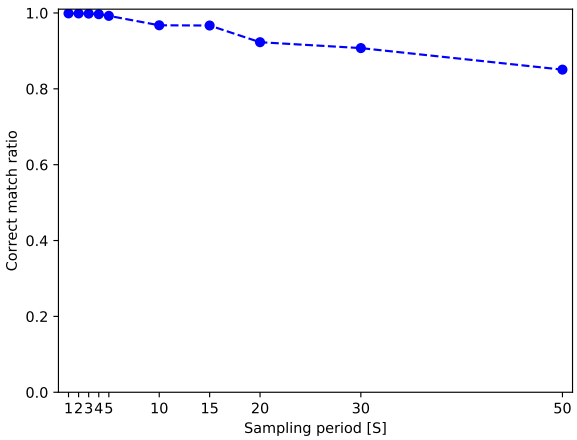


Fig. 6: Correct matching ratio for Seattle road network using real GPS trace data.

B. Real GPS trace (Seattle)

We here evaluate our algorithm in a real case scenario, by using data sampled at 1 Hz using a RoyalTek RBT-2300 GPS logger during a road trip in Seattle, WA [19]:

- **Road Network:** The 2009 road network representation of Seattle, Washington, USA area.
- **GPS trace:** 7531 GPS points collected over 2 hours of driving (80 km)[3].
- **Ground Truth:** The true sequence of road segments that correspond to the GPS trace.

The error in the GPS measurements is not a metric we control (as for the emulated trace) but is rather due to the GPS measuring device and environment.

Such real-world measures allowed us to take into account the dynamics of the road network which would cause the vehicle to slow down or even halt instead of traveling at the allowed speed limit for the whole trip.

Figure 6 depicts the performance results of our algorithm for matching a real GPS trace. The results corroborate the first part of our evaluation using emulated GPS data. At small sampling periods (less than 5 seconds), we are almost able to reconstruct the true route in its entirety. The correct match ratio decreases to 85% with a sampling period of 50 seconds.

VI. Conclusion & Perspectives

We presented a time-efficient map matching algorithm, to reconstruct the route from a sequence of data points. We proposed an unambiguous method: we do not aim to find the most probable route, but rather to identify the road segments that were used for *sure*. This way, we can merge heterogeneous traces, extracted for instance from a crowd-sensing application. Thus, our algorithm relies on identifying candidate edges for each data point, and computing routes among them which respect a time delay constraint. Our algorithm is time-efficient, and can be used for real-time situations. Our performance evaluation shows that our map matching technique unambiguously identifies more than 85% of the true route for a sampling period up to 50 seconds. When the data points are too interspaced, our algorithm terminates with several routes that cannot be differentiated.

In the future, we plan to analyse the impact of the local topography on the accuracy of the map matching technique. Obviously, an urban and dense environment can provide several similar routes, which complicate the matching procedure. Ideally, we would provide an adaptive method, able to tune dynamically the sampling rate depending on a set of local metrics, to reduce the energy consumption of many crowd-sensing applications.

[3]The details of the trace are available on: https://www.microsoft.com/en-us/research/publication/hidden-markov-map-matching-noise-sparseness/

References

[1] A. Gharaibeh, M. A. Salahuddin, S. J. Hussini, A. Khreishah, I. Khalil, M. Guizani, and A. Al-Fuqaha, "Smart Cities: A Survey on Data Management, Security, and Enabling Technologies," *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2456–2501, Fourthquarter 2017.

[2] T. Liebig, N. Piatkowski, C. Bockermann, and K. Morik, "Dynamic route planning with real-time traffic predictions," *Information Systems*, vol. 64, pp. 258 – 265, 2017.

[3] B. Guo, C. Chen, D. Zhang, Z. Yu, and A. Chin, "Mobile crowd sensing and computing: when participatory sensing meets participatory social media," *IEEE Communications Magazine*, vol. 54, no. 2, pp. 131–137, February 2016.

[4] M. Ahmed, S. Karagiorgou, D. Pfoser, and C. Wenk, "A comparison and evaluation of map construction algorithms using vehicle tracking data," *GeoInformatica*, vol. 19, no. 3, pp. 601–632, July 2015.

[5] J. Jokar Arsanjani, P. Mooney, A. Zipf, and A. Schauss, *Quality Assessment of the Contributed Land Use Information from OpenStreetMap Versus Authoritative Datasets*. Cham: Springer International Publishing, 2015, pp. 37–58.

[6] F. van Diggelen and P. Enge, "The World's first GPS MOOC and Worldwide Laboratory using Smartphones ," in *International Technical Meeting of The Satellite Division of the Institute of Navigation (GNSS)*. ION, 2015, pp. 361 – 369.

[7] M. Quddus and S. Washington, "Shortest path and vehicle trajectory aided map-matching for low frequency GPS data," *Transportation Research Part C: Emerging Technologies*, vol. 55, pp. 328 – 339, 2015.

[8] W. Li, D. Nie, D. Wilkie, and M. C. Lin, "Citywide estimation of traffic dynamics via sparse GPS traces," *IEEE Transactions on Intelligent Transportation Systems*, July 2017.

[9] X. Liu, J. Biagioni, J. Eriksson, Y. Wang, G. Forman, and Y. Zhu, "Mining Large-scale, Sparse GPS Traces for Map Inference: Comparison of Approaches," in *SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2012, pp. 669–677.

[10] Y. Li, Y. Li, D. Gunopulos, and L. Guibas, "Knowledge-based Trajectory Completion from Sparse GPS Samples," in *SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 2016, pp. 33:1–33:10.

[11] Y. Zheng, "Trajectory Data Mining: An Overview," *ACM Trans. Intell. Syst. Technol.*, vol. 6, no. 3, pp. 29:1–29:41, May 2015.

[12] J. Yuan, Y. Zheng, C. Zhang, X. Xie, and G. Z. Sun, "An Interactive-Voting Based Map Matching Algorithm," in *International Conference on Mobile Data Management*, May 2010, pp. 43–52.

[13] P. Szwed and K. Pekala, "An Incremental Map-Matching Algorithm Based on Hidden Markov Model," in *Artificial Intelligence and Soft Computing*, 2014, pp. 579–590.

[14] G. Hu, J. Shao, F. Liu, Y. Wang, and H. T. Shen, "IF-Matching: Towards Accurate Map-Matching with Information Fusion," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 1, pp. 114–127, 2017.

[15] J. Chen and M. Bierlaire, "Probabilistic Multimodal Map Matching With Rich Smartphone Data," *Journal of Intelligent Transportation Systems*, vol. 19, no. 2, pp. 134–148, 2015.

[16] R. Mohamed, H. Aly, and M. Youssef, "Accurate Real-time Map Matching for Challenging Environments," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 4, pp. 847–857, April 2017.

[17] J. Bresenham, "Algorithm for computer control of a digital plotter." *IBM systems journal*, vol. 4, no. 1, pp. 25–30, 1965.
[18] L. Heng, G. X. Gao, T. Walter, and P. Enge, "Statistical characterization of GPS signal-in-space errors," in *International Technical Meeting of the Institute of Navigation (ION ITM)*, San Diego, CA, 2011, pp. 312–319.
[19] P. Newson and J. Krumm, "Hidden markov map matching through noise and sparseness," in *17th ACM International Conference on Advances in Geographic Information Systems (SIGSPATIAL GIS)*, Seattle, WA, November 2009, pp. 336–343.