COBWEB: A Robust Map Update System Using GPS Trajectories

Zhangqing Shan[*†], **Hao Wu**[*†], **Weiwei Sun**[*†], **Baihua Zheng**[‡]
[*]School of Computer Science, Fudan University, Shanghai, China
[†]Shanghai Key Laboratory of Data Science, Fudan University, Shanghai, China
[‡]Singapore Management University, Singapore
[shanzhangqing, wuhao5688, wwsun]@fudan.edu.cn, bhzheng@smu.edu.sg

ABSTRACT

The accuracy and completeness of a digital map plays a critical role in determining the quality of most location-based services. Unfortunately, road networks change frequently. Consequently, we study the issue of automatic map update in this paper. We propose a system called COBWEB which takes all the unmatched trajectories as input and generates the missing road segments with both the geometry properties and topology features well preserved. We conduct a comprehensive experimental study via real trajectory data generated by roughly $15,000$ taxis in Singapore within a 5-month period. Compared with existing work, COBWEB demonstrates a better and more stable performance and a stronger resilience to various sampling rates and data sizes.

Author Keywords

Map update; map inference; GPS trajectories; GPS noise; low sampling rate; map matching.

ACM Classification Keywords

H.2.8 [**Database Management**]: Database Applications - *Data mining, Spatial databases and GIS*;

INTRODUCTION

Thanks for the fast deployment of GPS devices in mobile platforms, location-based services (LBSs) have been blooming in recent years. For most LBSs, the quality of service is directly related to the accuracy and completeness of a digital map. Unfortunately, road networks change quickly. For example, all the governments invest in transport infrastructure every year and part of the budget is for upgrading the road networks. The Ministry of Transport of China reported that 8,268 kilometers of new freeways were built in China in 2013, which is about 8% of the total length. Thus, digital maps must keep up-to-date because of the fast change of the road network. However, a large number of map providers update their maps by investigating the real roads manually. This approach consumes a significant amount of human and financial resources and it is hard to update the maps timely. Consequently, here comes the appeal for developing an automatic map update system.

Fortunately, with the fast development of GPS devices, large collections of GPS trajectories are becoming widely available. A research problem, namely *Map Inference* that is based on GPS trajectory data has attracted lots of attentions from academy recently. *Map Inference* aims to generate the skeleton of the whole road network and it focuses on the *main* skeleton of the network. Although updating issue is not the real focus of *Map Inference*, *Map Inference* does provide a possible solution as all the missing roads form a sub-map and *Map Inference* can find the main skeleton of this sub-map by mining GPS trajectories. However, if a detailed map that accurately captures majority of, if not all, the roads (including both major roads and minor roads) is required, map inference approaches might not be suitable to address map update issue as *none* of the map inference approaches can detect minor missing roads.

On the other hand, to the best of our knowledge, there are only one work which directly aims at *Map Update* problem [24]. However, the proposed approach is very simple and it fails to work well in the following situations which are, unfortunately, very common in practice. These situations can be regarded as the main challenges of the *Map Update* and *Map Inference* problem.

- **Low Sampling Rate:** Most trajectory data are collected in low sampling rate due to many limitations such as energy consumption and data storage. Approaches based on trajectory clustering, including [4, 13, 16], will have poor performance under this situation since trajectories become hard to be clustered and cannot represent the exact shape of roads.
- **GPS Noise:** It is a common observation that most GPS data points are noisy. However, very few works can handle the noise problem before recovering the map.
- **Timely Responsiveness:** For most related work, the bigger the trajectory dataset is, the better the performance will be. They cannot supply satisfactory results using sparse dataset, e.g. trajectories of a single day. However, most related work has to collect sufficient raw trajectories to support their computation in massive data consumption. In

UbiComp '15, September 7-11, 2015, Osaka, Japan.

http://dx.doi.org/10.1145/2750858.2804286

another word, they show weaknesses to undertake the updating tasks under the data restriction in short time.

- **Segments Connection in Complex Road Networks:** Most existing work adopts a simple way to connect the roads together, which will not work well when the road structure is complex and will be affected by the shape of the generated roads since the core idea of them is simply stretching one side of the generated road till it crosses over another road. This reckless method weakens the accuracy of cross location and is likely to ruin the topological structure of road network.

To the best of our knowledge, no existing work can handle all the challenges listed above concurrently. By considering all the four challenges mentioned above, we intend to design a map update approach that is robust, flexible and meanwhile stable in different situations. Our approach consists of three major steps, i.e., *Preprocessing*, *Cobweb Processing*, and *Refining*. The *Preprocessing* step extracts the data that can help to generate the missing roads; the *Cobweb Processing* step organizes the trajectory data using a graph structure called *Cobweb* and reduces the vertices and edges from the *Cobweb* into a *Road-Tree*. Note that this step plays the key role of our approach, thus we also name the system as COBWEB. Finally, the *Refining* step constructs the *Road-Graph* which consists of all the missing roads and merges the *Road-Graph* into the original road network to finish map update. Notice that our system is cyclic which means after one iteration of map update, we can use the new-coming trajectory data and match them onto the latest-updated road network to trigger the next iteration to generate more new roads. The contributions of our work are as follows:

- We propose a novel approach called COBWEB to implement both *Map Update* and *Map Inference* effectively.
- We take GPS measurement error into consideration and propose a filtering algorithm that can filter out noise GPS points effectively. The algorithm is self-adaptive and can deal with regions with different point densities. In this way, our approach overcomes the challenge of **GPS Noise** automatically.
- We introduce a new graph structure called *Cobweb* with several attractive properties. With the help of *Cobweb* structure, challenges of **Low Sampling Rate**, **Timely Responsiveness** and **Segments Connection in Complex Road Networks** can be addressed perfectly.
- We conduct experiments using massive real data from Singapore. Trajectories of more than 15,000 taxis are used in our experiments. Result shows that our approach outperforms other *Map Update* and *Map Inference* approaches significantly and our approach is robust under different circumstances.

RELATED WORK

COBWEB has a close relationship with *Map Update*, *Map Inference* and *Map Matching* issues. Before COBWEB starts to process GPS trajectories, some preprocessing works are necessary. According to the instruction of [27], our system integrates with components of map-matching and noise filtering.

Map Update

Map Update aims at updating the new roads based on the current map. *Map Inference* technology can also be used in *Map Update* problem by inferring the sub-graph of the map or regenerating the whole map. To some degree, they share a high similarity. However, they are still different. Although both of them can solve the problem of each other, their emphases are different. *Map Inference* lays emphasis on obtaining the main roads of the city and does not care about the small/minor roads. However, *Map Update* works in a more delicate manner. It focuses on those small roads and tries its best to recover all the missing roads as much as possible. It does not pay much attention to the topology of the road since the missing roads only cover a very small portion of the map.

The *CrowdAtlas* approach proposed in [24] is the only piece of systematical work related to *Map Update*. It takes trajectory data as input, and matches them to a given map using map matching algorithm. The unmatched trajectories are then clustered according to the Hausdorff distance. During the clustering process, the size of each cluster increases as more and more unmatched trajectories are clustered. Once the size of one cluster reaches a predefined threshold, a new road will be generated based on the trajectories within that cluster.

CrowdAtlas performs well in high sampling rate data but has a poor performance in low sampling rate data because the shape of generated new roads could be wrong. In addition, *CrowdAtlas* is only suitable for updating fragmentary parts of road network because of its clustering operation. It shows low effectiveness to produce an integrated and well-structured road network for a missing region in existing map.

Map Inference

Map Inference is a technology to generate the whole map automatically. For *k-means* style algorithms [1, 9, 19, 26], they perform a traditional *k-means* style clustering of GPS samples, where the distance measure may involve both the geometry distance to the cluster center as well as its GPS bearing. Then, they link appropriate cluster centers to generate roads according to the traces that pass through the clusters. For *Kernel Density Estimation (KDE)* methods [6, 7, 20, 22], they transform GPS points or trajectories into discretized image representing the density of samples or segments at each pixel. After that, a binary image of the roads in the region is generated according to a binary threshold. Finally, the centerlines could be extracted by a variety of methods, e.g., Voronoi partitioning. For *Trace Merging algorithms* [4, 16], they accumulate traces into a growing road network, where each addition meets the location and meanwhile bears constraints of existing roads. [3] makes some accurate conclusion for most above algorithms.

[12] gives a thorough evaluation for most typical algorithms in *Map Inference*. According to its overall comparison, *Trace Clustering Algorithm (TC1)* [13] and *Kernel Density Estimation (KDE)* [6, 7, 20, 22] are outstanding algorithms with superior performance. *TC1* decomposes each trajectory into several small segments, then clusters remaining segments using a *single-linkage clustering*-based method, and finally

(a) G (b) G_{RG} (c) G+G_{RG}

Figure 1. An example of the map update processing

extracts the centerline of each road from the clustering using *B-spline fitting*. Nevertheless, *TC1* also shows its weakness in low sampling rate condition. In addition, it may experience low performance for generating the integrated and well-structed road network because of the data loss in the decomposition. *KDE* owns robustness in low sampling rate condition, however, it suffers a lot in data sparsity condition.

Map Matching

The target of Map matching technology is to reduce the influence of GPS error, and to match GPS points to roads correctly. After *Map Matching*, we obtain a route sequence for each trajectory, which indicates the object's actual moving path. There are three main categories of map matching technology, including one based on the geometry of each road [18, 25], one based on the topology of the road network [5, 17, 15], and the other based on probability [14, 21]. The third one is the most competitive because of its outstanding accuracy. In *ACM SIGSPATIAL Cup 2012* on *Map matching* contest, 4 of top 5 winners used the improved *Hidden Markov Model (HMM)* [2]. Because [21] demonstrated the highest precision in the contest ($> 98.5\%$), it is employed as the map matching algorithm to extract the unmatched trajectories from source data.

PRELIMINARY

In this section, we first present some important concepts and then describe the problem we are going to study in this paper.

Definition 1. ***Road Network.*** *A road network is a directional graph $G(V, E)$, where V refers to the set of vertices and E refers to the edge set. An edge $e(v_i, v_j) \in E$ represents a real road segment from vertex v_i to vertex v_j in reality.*

Definition 2. ***GPS Trajectory.*** *A GPS trajectory T is defined as a sequence of n GPS points, i.e. $T = \{(x_1, y_1, t_1), (x_2, y_2, t_2), \ldots, (x_n, y_n, t_n)\}$. Here, (x_i, y_i) refers to longitude and latitude of the i-th point at time stamp t_i in the trajectory.*

Definition 3. ***Generated Road Graph.*** *Given an original road network G, let G_{RG} represent the sub-network that is not captured by G but generated by certain map update algorithm. Then, $G + G_{RG}$ represents the updated version of G, corresponding to a real road network G_{real}.*

Take Figure 1 as an example. Map update algorithm recovers the missing roads G_{RG} in Figure 1(b) using the GPS trajectories on the original road network G shown in Figure 1(a). After that, it adds G_{RG} to G to generate the new map $G + G_{RG}$ shown in Figure 1(c).

COBWEB

In this section, we introduce a novel system as our solution to handle the *Map Update* problem, namely COBWEB. COBWEB consists of three components, as shown in Figure 2, *Preprocessing*, *Cobweb Processing*, and *Refining*.

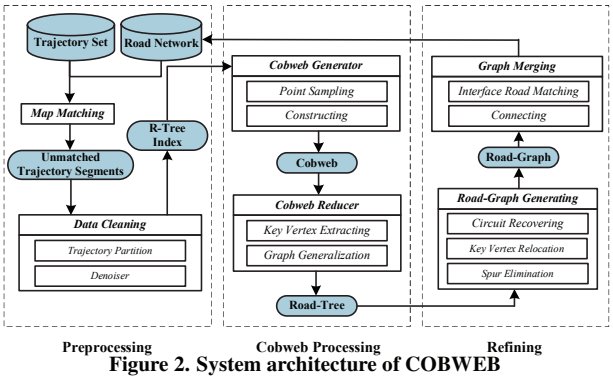


Figure 2. System architecture of COBWEB

To be more specific, *Preprocessing* is to preprocess all the GPS trajectories via *Map Matching*. GPS trajectories are matched onto a given road network. When the map is incomplete, some trajectories cannot be mapped as the corresponding road segments are missing in the map. Our map update algorithm only considers those GPS trajectories that fail in the step of *Map Matching*. We design a cleaning process to filter noisy GPS points that are generated because of GPS errors. Finally, an R-tree index is constructed to index the GPS trajectories which fail in the map-matching but pass the cleaning process.

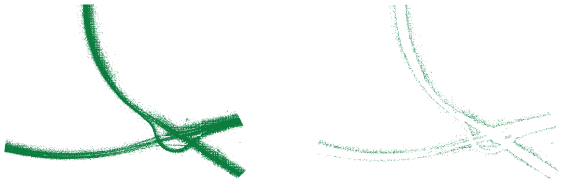
Cobweb Processing organizes the GPS points using a graph structure called *Cobweb* and reduces the vertices and edges from the *Cobweb* into a *Road-Tree*. Note that this component plays a key role in the system, which explains the reason why we name our system as COBWEB. Refining recovers the *Road-Graph* from *Road-Tree* with topological compensation and merges the *Road-Graph* into the original road network to finish map update. We detail the three components in the following. They also represent the three main steps of our map update algorithm adopted by COBWEB.

Preprocessing

The *Preprocessing* in COBWEB mainly takes care of the preprocessing of raw GPS trajectories, including map matching and data cleaning. The former is to pick out the GPS trajectories corresponding to unknown road segments that are missing from the given road network; and the latter is to filter out the noisy GPS points that do not accurately record the real positions of moving objects. We employ [21] as our map matching algorithm to map GPS trajectories to road segments because of its high precision. After that, all the unmatched trajectory segments would be collected as the input data for following step.

GPS technology has intrinsic errors mainly due to the complexity of user environment. As the map update algorithm tries to recover the missing roads based on the GPS trajectories, the accuracy of the recovered road segments highly depends on the quality of the input GPS trajectories. In other words, an input set containing lots of noisy GPS data actually implies that the generated road segments are not accurate. However, many existing works overlook the quality issue of GPS trajectories. This reckless attitude explains why some algorithms have very poor performance on certain datasets.

In COBWEB, we include a data cleaning sub-step which can filter out GPS points with large errors. Intuitively, GPS points with large GPS error are more likely to be far away from other points. As visualized in Figure 3, data cleaning step filters out those GPS points far away from others. This phenomenon suggests that a *nearest neighbor-based* approach may work well. In other words, we can rely on the distance between a point p and its k^{th} nearest neighbor $p.o^k$ in the entire input GPS trajectory set, denoted as $d(p, p.o^k)$ to judge whether p should be filtered out or not. Using the distance to the k-nearest neighbor as the criterion of detecting the noise is one type of proximity-based outlier detection which is discussed in [23]. An intuition is that the larger the distance $d(p, p.o^k)$ is, the more likely that p is a large-error GPS point. We introduce γ as the filtering threshold and filter out all the GPS points p with $d(p, p.o^k) > \gamma$. Notice that γ should not be a fixed value since the density of the GPS point distribution within one region is different from that in another region, which means γ of a road with dense GPS points should be smaller than the one with sparse points. In the following, we introduce a heuristic based approach to adjust the value of γ.



(a) Original GPS points (b) De-noised GPS points

Figure 3. An example of data cleaning

Given a road segment seg, we represent it using a center line L_{seg} with width $2L_d$. Let $S_{seg}(T)$ represent a set of trajectories T that can be successfully mapped to seg. Then, for any GPS point p on a trajectory $T \in S_{seg}(T)$, its vertical distance to L_{seg}, denoted as $p.\eta$, follows the Gaussian distribution, according to [8, 15]. In other words, $p(\eta) = e^{-\frac{(\eta-\mu)^2}{2\sigma^2}}/\sqrt{2\pi}\sigma$. Without loss of generality, we assume $\mu = 0$ and GPS points mapped to different road segments share a common standard deviation σ value. Consequently, we can use all the GPS trajectories that are successfully mapped to existing road network (i.e., those trajectories not considered by map update algorithms) to derive σ value (e.g., 15.4307m in our experiments).

$$N(p_c, \gamma) = \theta \iint_{(x-p_c.x)^2+(y-p_c.y)^2 \leq \gamma^2} p(x)dxdy \qquad (1)$$

Once the Gaussian Distribution $p(\eta)$ is known, we form a function $N(p_c, \gamma)$ to approximate the number of GPS points that have their distances to a given point p_c bounded by γ, as shown in Eq. (1). Note, θ here is a density function which represents the number of GPS points in a unit region, and it can be approximated by the ratio of the total number of points mapped to a road segment L_{seg} to the length of L_{seg} (i.e., $|L_{seg}|$). As mentioned before, all the GPS points p_c with $N(p_c, \gamma) < k$ shall be filtered as the distance from p_c to their k^{th} nearest neighbor $d(p_c, p_c.o^k)$ is definitely larger than γ.

In order to maximize γ value, we set $N(p_c, \gamma)$ to be k. Note that as the road segment L_{seg} of missing road is unknown, we rely on all the GPS points that can be successfully mapped to existing road network to derive the value of γ. $N(p_c, \gamma)$ value takes a density function θ as input and we notice the density of different areas could be very different. Consequently, we introduce a grid-based partitioning step to data cleaning, denoted as *trajectory partition*. We partition the whole road network into small equal-sized grids, denoted as P_i, and then map GPS points of GPS trajectories to grids. Those grids not containing any GPS points are discarded as very likely they do not cover any road segment. As the grids are small, we can assume the density of GPS points within a grid is uniform, and we can also use $\frac{P_i.area}{2L_d}$ to approximate the length of road segment falling within P_i, where $P_i.area$ refers to the area size of a grid P_i and L_d is a system parameter. Meanwhile, we can count the number of points within each partition P_i, denoted as $P_i.count$. Thereafter, the density function θ_i corresponding to partition P_i can be approximated by $\frac{P_i.count}{P_i.area} \times 2L_d$. Although there are multiple points p_c mapped to each road, we take the boundary point p_c with their distance to L being L_d as representative and rely on p_c to derive γ value.

As a summary, the *Preprocessing* step of COBWEB is responsible for preprocessing the GPS trajectories. It employs existing map matching algorithm to separate the trajectories into two sets, *matched* and *unmatched*. The set of unmatched trajectories becomes the input of the map update algorithm, and we propose a simple but effective filtering algorithm to remove noise GPS data to address the challenge of **GPS Noise**. The set of matched trajectories is also utilized to understand certain properties of the road network (e.g., density of GPS points).

Cobweb Processing

After the *Preprocessing* step, GPS points are ready for map update algorithm and then we move to the next step, namely *Cobweb Processing*, that is handled by *Cobweb Generator* and *Cobweb Reducer*. Before we present the details of this step, we would like to first formally introduce a novel data structure *Cobweb* in Definition 4. It tries to organize the set of input GPS points using a graph structure $C(V, E)$, under three restrictions. Restriction i) is to guarantee that $C(V, E)$ only consists of the points of input set P; restriction ii) is to remove redundant GPS points that are very close to each other; and restriction iii) is to guarantee the topological structure of missing roads is well captured.

Definition 4. ***Cobweb.*** *Given a point set P and two bounds R_v and R_c, a* ***Connecting-On-Bounds*** *(Cobweb) is an undirected graph $C(V, E)$ such that i) $V \subseteq P$; ii) $\forall v_i, v_j \in V$, $d(v_i, v_j) \geq R_v$; and iii) $\forall e(v_p, v_q) \in E$, $d(v_p, v_q) \leq R_c$. In other words, R_v is the lower bound of each edge's length in Cobweb and R_c is the upper bound of that.*

We would like to highlight that the *Cobweb* structure is very novel, and it has a few unique characteristics that will benefit map update algorithms. First, our *Cobweb* structure is based on GPS points instead of trajectories and hence the sampling rate does not affect the performance of our *Cobweb*-based map update algorithm, as long as we collect a large

number of trajectories. This can effectively address the challenge of **Low Sampling Rate**. Second, in this point-based strategy, the distribution of points could be captured even with sparse trajectory dataset. Thus, COBWEB is able to recover roads effectively regardless of various data volume. In this way, the challenge of **Timely Responsiveness** is addressed. Third, *Cobweb* is a graph and it connects nearby GPS points via edges which can help to preserve the topological structure of the underlying road network that is missing. As to be demonstrated later, *Cobweb* can recover even complex road structure to successfully address **Segments Connection in Complex Road Networks** issue.

The generation of *Cobweb* is simple, with its pseudo code listed in Algorithm 1. It randomly picks a point p in the input GPS point set P and inserts it to V, and meanwhile removes all the points with distances to p smaller than R_v (the set returned via function RangeSearch(p, P, R_v)) from P. The above process repeats until P is empty (i.e., points in P are either removed or maintained by V). Thereafter, for all the pairs of vertices (v_i, v_j) in V with their Euclidean distances bounded by R_c, we insert an edge $e(v_i, v_j)$ to E accordingly.

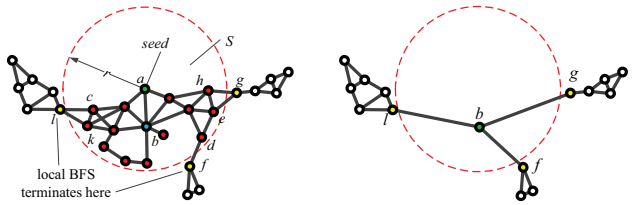
After constructing *Cobweb*, we then try to locate the points that are located on (or near to) the center of missing road segments or cross roads in order to recover the missing roads. We achieve this goal via introducing a new structure namely *Road-Tree*. *Road-Tree* is formed by generalizing Cobweb $C(V, E)$, with its pseudo code listed in Algorithm 2. The main idea is to locate a set of points, denoted as S, that are located near the same missing road/cross road, and then replace the set S with a vertex v_c, namely *key vertex*, that is closest to the center point of S. All the edges from a point $p \in V - S$ to a point $s \in S$ are replaced by edges $e(p, v_c)$.

Algorithm 1 Gen-Cobweb(P,R_c,R_v)

```
1: V ← ∅, E ← ∅;
2: while P is not empty do
3:    p ← a random point in P;
4:    V ← V ∪ {p}, P ← P − {p};
5:    V_p ← RangeSearch(p, P, R_v);
6:    P ← P − V_p;
7: for ∀v_i, v_j ∈ V do
8:    if d(v_i, v_j) ≤ R_c then
9:       E ← (v_i, v_j)
10: return C(V, E);
```

Now we explain how to find the set S, i.e., lines 5-25 in Algorithm 2. The main idea is to reduce Cobweb from any seed vertex $seed \in V$ guided by a radius r via local Best-First Search (BFS). Started from $seed$, we will reach all the vertices that have not yet been visited and meanwhile have their network distances to $seed$ on *Cobweb* bounded by r. Take Figure 4(a) as an example. The local BFS search initiated from seed vertex a will reach all the vertices located within the dot-line circle, denoted as $E_{a,r}$. Note our search is based on network distances on *Cobweb* but not Euclidean distance and all the vertices within the circle can be reached from a via network distance bounded by r. The local BFS then resumes at points l, g, and f, those points that are connected directly to some points of $E_{a,r}$ but have their network distances to a beyond r. In other words, we will have another three sets, i.e., $E_{l,r}$, $E_{g,r}$, and $E_{f,r}$.

After all the points in S are visited, we find a vertex in S that is closed to the geo-center of S, namely *key vertex* (e.g., vertex b in our example); we then replace S with the key vertex to generalize the road segment/cross road where points in S are located. Meanwhile, the edges from a point outside of S to any point inside S are now connected to the key vertex, e.g., edges $e(l, c)$ and $e(l, k)$ are replaced by $e(l, b)$, edges $e(g, e)$ and $e(g, h)$ are replaced by $e(g, b)$, and edge $e(f, d)$ is replaced by $e(f, b)$, as shown in Figure 4(b).



(a) Local BFS initiated at a (b) Generalization

Figure 4. An example of Cobweb generalization

Algorithm 2 Form-RoadTree($C(V, E)$,r, δ)

```
1: s ← a random point p in V, mark s;
2: Q_g ← {s}, Q_l ← ∅, count ← 0, S ← ∅;
3: while Q_g is not empty do
4:    seed ← de-queue(Q_g);
5:    for ∀e(seed, p) ∈ E do
6:       if p is not marked then
7:          mark p, en-queue(Q_l, p, ||seed, p||_C);
8:    while Q_l is not empty do
9:       p ← de-queue(Q_l);
10:      if ||p, seed||_C < r then
11:         count++, S ← S ∪ {p};
12:         for ∀e(p, p') ∈ E do
13:            if p' is not marked then
14:               mark p', en-queue(Q_l, p', ||seed, p'||_C);
15:      else
16:         if count > δ then
17:            v_c ← GeoCenter(S);
18:            V' ← AdjacentVertex(C, S);
19:            V ← V − S + {v_c};
20:            E ← E − ∪_{s_i,s_j∈S} e(s_i, s_j);
21:            for each vertex v ∈ V' do
22:               en-queue(Q_g, v), E ← E ∪ {e(v_c, v)};
23:            count ← 0, S ← ∅;
24:         while Q_l is not empty do
25:            p ← de-queue(Q_l), unmark p;
26: return C(V, E);
```

Algorithm 2 takes search radius r and minimal queue size δ as input, so we need to decide their values before we start Algorithm 2. Here, we set minimal queue size δ as 5 directly. Because if $\delta < 5$, the generalization quality would be diminished by the shortage of data. At the same time, we use several preliminary probe searches to compute the most appropriate radius r. We initiate from point a to increase r step by step (e.g., in our implementation we increase r by 5 meters in each iteration until r exceeds 50 meters) until there are only

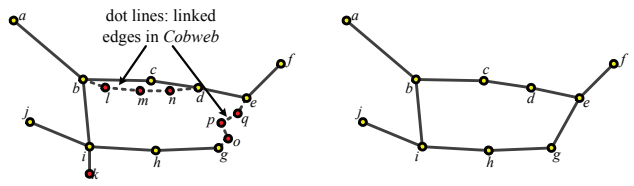
a small amount of unvisited and linked points outside the expansion circle or r exceeds a large threshold. In Figure 4(a), only three points (i.e., points l, f, and g) are unvisited and linked points, so we can make the judgement that g, l, f are not located within the same road segment/road cross at the position of S. The intuition behind is that for a given road segment/cross road that has not yet been explored, the initial few expansions will reach a large number of points located nearby. As the expansion continues, more and more sub-areas within the road segment/cross road are explored and eventually the expansion will reach mainly the space outside the road segment/cross road and hence the next expansion circle will cover a very small number of, if not zero, unvisited points.

Refining

When the *Road-Tree* from a *Cobweb* is ready, the *Refining* step starts which consists of two sub-steps, namely *Road-Graph Generating* and *Graph Merging*. The former is to recover the missing roads from *Road-Tree* and the latter is to connect the recovered missing roads to the original map to complete the *Map Update*.

Road-Tree constructed in previous step actually captures the main structure of the missing roads. However, because of the construction algorithm adopted in this paper, Road-Tree is free of any circuit while circuits do exist in real road network. Consequently, *Road-Graph Generating* based on *Road-Tree* is to recover those circuits by checking each pair of non-adjacent nodes (v_i, v_j) in the *Road-Tree*. Our judgement on whether v_i and v_j shall be connected directly in *Road-Graph* is based on two distance metrics, i.e., their network distance on *Cobweb* $C(V, E)$ denoted as $|v_i, v_j|_C$ and their distance on *Road-Tree* denoted as $|v_i, v_j|_T$. If $|v_i, v_j|_C \leq \theta_C$ and $|v_i, v_j|_T \geq \theta_T$, v_i and v_j shall be linked directly in *Road-Graph*. Here, the configuration of parameters θ_C and θ_T implements the avoidance of the duplicated edges and circuit recovering in road network.

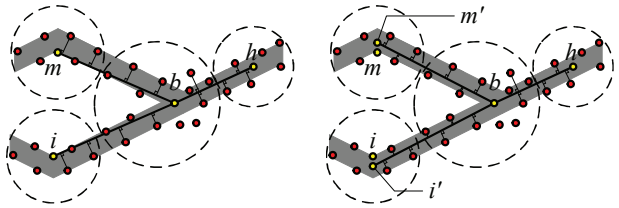
An example partial *Road-Tree* is plotted in Figure 5(a), where the solid lines are from *Road-Tree* and dashed lines are from *Cobweb*. Take two vertex pairs (b, d) and (e, g) as examples, under the assumption that $\theta_C = 50m$ and $\theta_T = 100m$. Given the fact that $|b, d|_C = 40m$ and $|b, d|_T = 35m$, b and d will not be connected; on the other hand, as $|g, e|_C = 30m$ and $|g, e|_T = 150m$, g and e are connected on *Road-Graph*, as shown in Figure 5(b). When *Road-Graph* is generated from *Road-Tree*, we try to eliminate spur road segments from *Road-Graph*. Pair i and k is the example of spur road segments. The real distance between i and k is very short ($|i, k|_T = 15m$) and hence below the minimum length of any real road segment (e.g., 25 meters in our experiments), so vertex k and edge (i, k) are deleted from the *Road-Graph*.



(a) Road-Tree (b) Road-Graph

Figure 5. Circuit recovering and spur elimination

Once *Road-Graph* is ready, we can start the last sub-step to map the recovered missing roads back to the original map. To achieve this goal, we first re-position the vertices of *Road-Graph*, i.e., all the key vertices identified in above *Road-Graph Generating* step, to accurate map locations. As shown in Figure 6, we plot four key vertices b, m, i and h, and their corresponding sets S (i.e., those points located within the dash line circles centered at key vertices). Take road segments formed by i and b as an example. The points having their projection on line segment (i, b) are original GPS points corresponding to this road segment. If (i, b) has been accurately mapped to the exact position in reality, the summation of the distances from those GPS points to the segment (i, b) shall be minimized. In other words, we can re-locate i and b to their accurate positions by minimizing the distance summation. We utilize gradient descending mechanism to approach the best location gradually.



(a) before relocation (b) after relocation

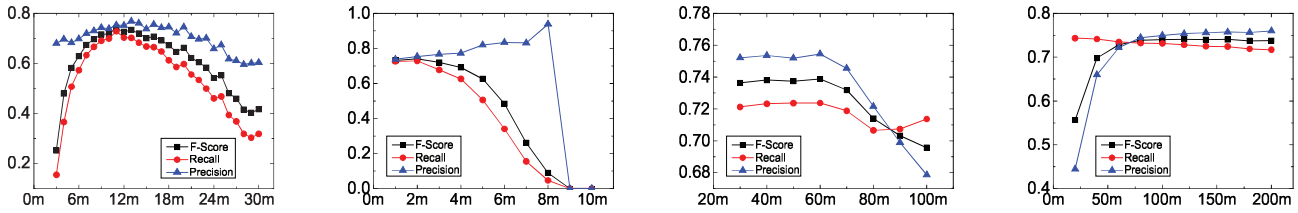
Figure 6. Key vertex relocation by gradient descending mechanism

For the merging sub-step, we propose a new approach. Recall that in the *Preprocessing* step, we only extract those trajectories which cannot be matched to the original map. Here, we make a small change. For each unmatched trajectory collected in the *Preprocessing* step, we extend both sides and extract those successfully matched GPS points, namely *interface points* in this work. Based on unmatched GPS points and the set of *interface points*, following steps find not only the missing roads but also the road segments adjacent to those missing roads, which are named as *interface roads* in this work. Since we can match those *interface roads* to the original map, at the same time *Cobweb* well preserves the topological structure of all the generated roads, it is feasible to connect each missing road to the original map by figuring out corresponding part of original road network for each interface road.

EXPERIMENTS

Recall and *precision* are often employed as the metrics to quantify the effectiveness of a map inference or map update approach. With the help of Definition 3, *precision* ρ is defined as $\frac{|(G_{RG} \cap G_{real}).E|}{|G_{RG}.E|}$ and *recall* β is defined as $\frac{|(G_{RG} \cap G_{real}).E|}{|(G_{real} - G).E|}$, with G_{real} representing the road network in reality. However, it is hard to make a judgment whether an algorithm with high *precision* and low *recall* is better than another one with low *precision* but high *recall*. Thus, we adopt the comprehensive metric *F-Score* (the harmonic-mean of *recall* and *precision*) to make an overall comparison among various algorithms [3, 12]. *F-Score* α is defined as $\frac{2 \times \rho \times \beta}{\rho + \beta}$. In the following, we conduct our evaluation via *F-Score*, because a higher *F-Score* implies a better performance in both *recall* and *precision*.

In this section, we conduct two sets of experiments, *parameter experiments* and *comparison experiments*. The former is



(a) R_C (b) R_V (c) θ_C (d) θ_T

Figure 7. Performance under various parameters

to evaluate the impact of different parameters, including R_C, R_V, θ_C, and θ_T, on the performance of COBWEB and also to figure out a proper setting for those parameters. The latter is to compare COBWEB against other existing algorithms, including *TC1*, *KDE* and *CrowdAtlas*, on map update and map inference. As we know most map inference applications consider *concentrated trajectory data* while map update applications focus on *discrete trajectory data*, we consider both types of trajectory datasets in our experiments in order to provide a complete view on the performance of different algorithms.

A collection of massive raw taxi trajectories in Singapore supports all the experiments above. It contains 1,550,000 trajectories (20 billions GPS points), collected within a 5-month period. For ground-truth, we select Singapore road network data from OpenStreetMap (OSM) to check the results of *TC1*, *CrowdAtlas*, *KDE* and COBWEB.

Parameter Experiments

Our COBWEB relies on many parameters, and our first set of experiments is to evaluate the impact of different parameters, including R_C, R_V, θ_C, and θ_T, on the performance of COBWEB. We conduct our evaluations with *F-score*, *recall* and *precision*. Performance of these parameters is derived based on geometrical shape and location of road segments. For each road segment in generated road graph G_{RG}, we decompose it into 20 pieces uniformly. Then, each piece is compared with ground-truth $G_{real} - G$. If this piece matches certain road segment nearby and is parallel in the ground-truth (maximal distance between generated pieces and ground-truth roads is 20m and maximal deviation angle of direction is 15°), it is an element of $G_{RG} \cap G_{real}$. In this way, the *F-Score*, *recall* and *precision* could be derived.

We select a portion of road network located in the downtown of Singapore with complicated geometrical shape and topological relationship as the focused region. The dataset includes $31,708$ trajectories (i.e., $200,009$ GPS points) in this $1.5km \times 3km$ rectangle region. Figure 7 plots the performance of COBWEB under different parameters. It can be observed that the best configuration for geometry is $R_C = 11m$, $R_V = 2m$, $\theta_T = 120m$, and $\theta_C = 30m$.

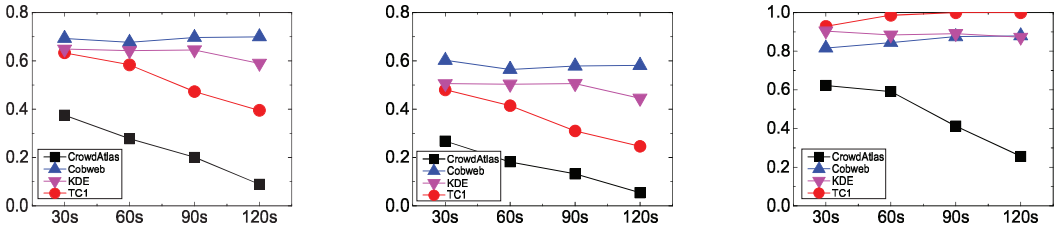
First, we study the impact of R_C, the maximum distance threshold to decide whether two vertices in *Cobweb* shall be connected. As shown in Figure 7(a), the performance is first improved as R_C increases its value and achieves the best performance when $R_C = 11m$. After that, the performance drops slowly as R_C further increases its value. A low value of R_C makes the distribution of edges in *Cobweb* sparse, thus performance is improved with increase of R_C at first. However, if the value exceeds the average distance between two parallel roads, GPS points on two roads may be linked directly. Consequently, some parts of *Cobweb* become vague and chaos. This phenomenon explains the descending trend after the highest point.

Second, we study the impact of R_V, the distance threshold to decide whether nearby points of a vertex v in all unmatched trajectories shall be included into *Cobweb*. As observed in Figure 7(b), with the increase of R_V, the recall performance decreases gradually and precision performance maintains at a high level initially and then drops dramatically. The high value of R_V raises up the sampling threshold of *Cobweb Generator* and decreases the volume of *Cobweb*, and hence undermines the *recall* metric. Before R_V exceeds a certain threshold, the high value could simplify *Cobweb* so as to generate some major roads more accurately. When it exceeds that threshold (around $8m$ in our experiments), *Cobweb* becomes so sparse that *Cobweb Reducer* could not generate proper *Road-Tree* for most parts of the road network, which destroys the *precision* performance suddenly. In summary, *F-Score* performance only maintains a high level at the initial range.

Third, we study the impact of θ_C and θ_T, two distance thresholds used to decide whether two vertices shall be connected directly in *Road-Graph*. As shown in Figure 7(c), the comprehensive performance maintains a high level at first and then declines gradually as θ_C changes. A large θ_C would introduce too many duplicated edges to *Road-Graph* and thus deteriorates the quality of generated roads. On the other hand, θ_T demonstrates a different impact. As shown in Figure 7(d), a small θ_T shows a weak status at first. After θ_T falls into an appropriate range, it can effectively avoid the generation of duplicated edges and meanwhile recover circuits. Consequently, the metrics score higher and higher. In summary, we can conclude that the best configuration complies with following rules in most times, $R_C \geq 2R_V$, $2m \leq R_V \leq 10m$, $30m \leq \theta_C \leq 50m$, and $60m \leq \theta_T \leq 120m$.

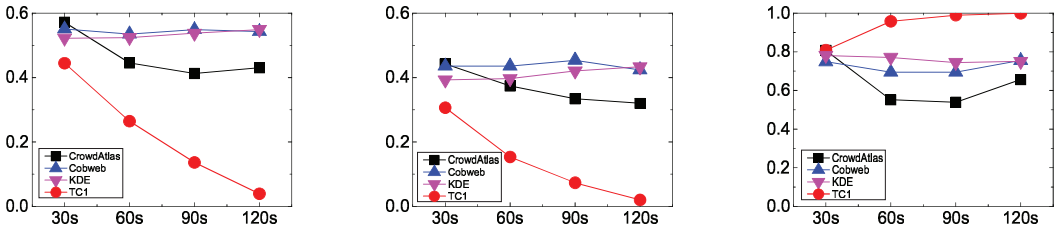
Comparison Experiments

In our second set of experiments, we compare the performance of COBWEB with three competitors. As explained before, we conduct this set of experiments on two types of dataset, namely *concentrated dataset* and *discrete dataset*. In order to check the robustness of different algorithms in various sampling intervals, we further partition both datasets into four groups, according to their average sampling interval, i.e., 30s, 60s, 90s, and 120s. Note that 30s (i.e., update per 30 seconds) refers to an example of high sampling rate while 120s refers to an example of low sampling rate. We visualize generated roads of two types of datasets in Figure 12 and



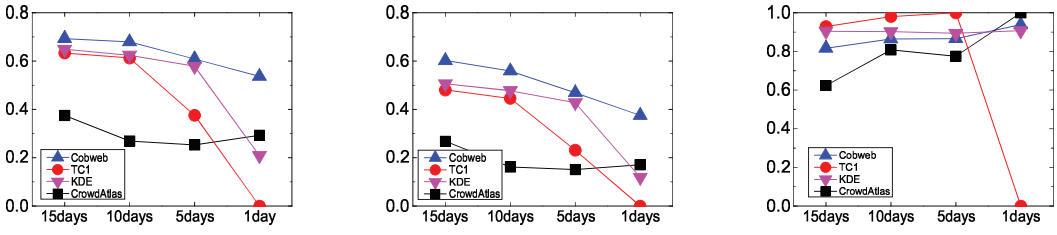
(a) F-Score (b) Recall (c) Precision

Figure 8. Performance under various sampling intervals using concentrated dataset



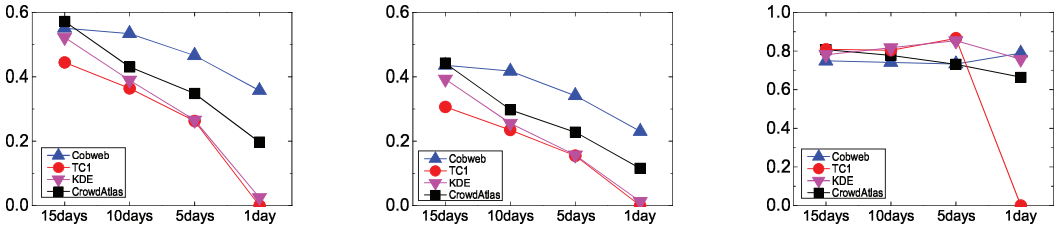
(a) F-Score (b) Recall (c) Precision

Figure 9. Performance under various sampling intervals using discrete dataset



(a) F-Score (b) Recall (c) Precision

Figure 10. Performance under various data volume using concentrated dataset



(a) F-Score (b) Recall (c) Precision

Figure 11. Performance under various data volume using discrete dataset

Figure 13 respectively, under the sampling rate of 30s and 120s. From the figures, we can observe that trajectories are distributed intensively in concentrated dataset and discretely in the other dataset. Besides, in order to check the robustness of several algorithms in various data volume, we divide both datasets into four different sizes respectively, i.e., 1 day size, 5 days size, 10 days size and 15 days size. The variety of performance in different data size reflects the scalability and robustness of different algorithms.

Concentrated dataset consists of $169,858$ trajectories of 15 days ($600,001$ GPS points) in a $3km \times 4km$ rectangle region. In this dataset, all the trajectories are located in several highways intensively, so this data represents the common statement of map inference application. After several training detection, the parameter configuration of *KDE*, *TC1* and *CrowdAtlas* is set in best range. For COBWEB, the parameter setting is that $R_C = 15m$, $R_v = 7m$, $\theta_C = 30m$, and $\theta_T = 60m$.

We plot the shapes of road segments generated by different algorithms under two sampling intervals (30s and 120s) in Figure 12, and report the performance of different algorithms under various sampling intervals in Figure 8. As observed, the shape of trajectories is distorted more and more as the sampling interval becomes longer and longer. For *CorwdAtals*, the distortion of trajectories undermines the performance of both *recall* and *precision*. *TC1* decomposes the original trajectories into short well-shaped segments, and uses these segments to generate roads. Hence, the *precision* of *TC1* is robust to the change of sampling interval. However, when the interval is very high, original trajectories are split into too many fragmental pieces which causes the loss of lots of geometry and topology information. Consequently, the *recall* performance deteriorates with the rise of interval. On the contrary,

COBWEB only uses GPS points to generate *Cobweb* which well preserves the geometry and topology information. Thus, COBWEB demonstrates a superior performance under different sampling intervals. Like COBWEB, *KDE* transforms GPS points into a discretized image and then uses the pixels of this image to generate road network. Thus, the variety of sampling interval has a very limited impact on its performance.

Discrete dataset consists of $365,243$ trajectories ($1,045,342$ GPS points) in $11km \times 14km$ rectangle region. In this dataset, all the trajectories are distributed in entire Singapore City downtown region discretely. This kind of dataset commonly exists in map update application. By training detection, the parameter configuration of *TC1*, *KDE* and *CrowdAtlas* stays at best position. For COBWEB, the parameter setting is that $R_C = 10m$, $R_v = 5m$, $\theta_C = 25m$, and $\theta_T = 100m$. Similar as the previous dataset, we plot the shapes of generated road segments corresponding to two different sampling intervals in Figure 13. We also report the performance of different algorithms under various sampling intervals for discrete dataset in Figure 9. In this map update dataset environment, *TC1* shows a poor recall performance in each interval. Because in this discrete type, the average length of each trajectory is shorter (30m$\sim$50m), and the decomposition process of *TC1* causes the loss of geometry information. This explains why the recall curve remains in low position. On the other hand, *CrowdAtlas* performs better in discrete dataset. This is because it collects each trajectory in each road position until the amount of collection is sufficient, and then clusters these trajectories into a new road. This strategy elim-

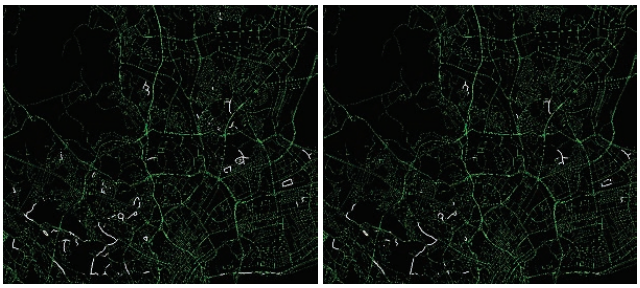
(a) COBWEB - 30s (b) COBWEB - 120s

(c) CrowdAtlas - 30s (d) CrowdAtlas - 120s

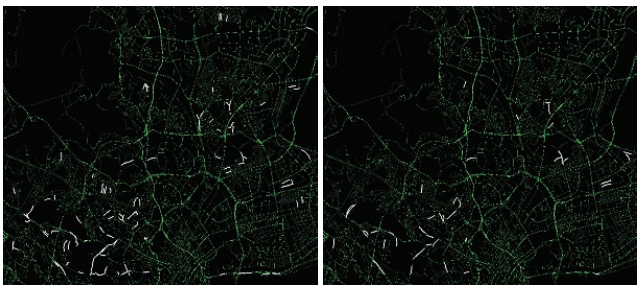
(e) TC1 - 30s (f) TC1 - 120s

(g) KDE - 30s (h) KDE - 120s

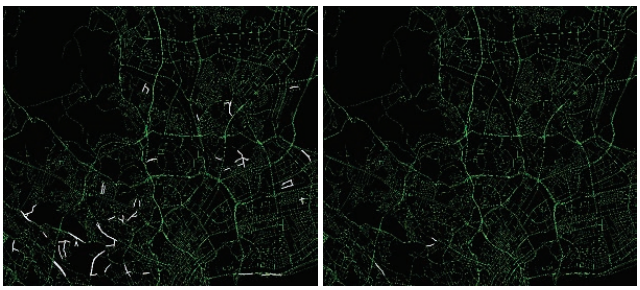
Figure 12. Visualization of generated roads using concentrated dataset



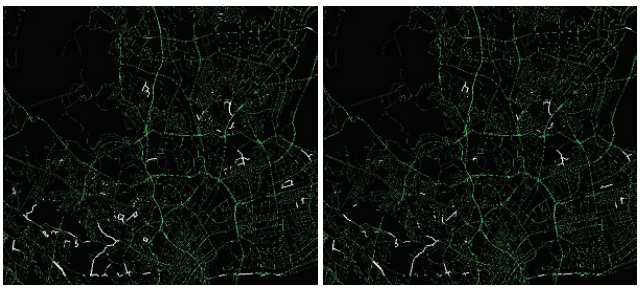
(a) COBWEB - 30s (b) COBWEB - 120s



(c) CrowdAtlas - 30s (d) CrowdAtlas - 120s

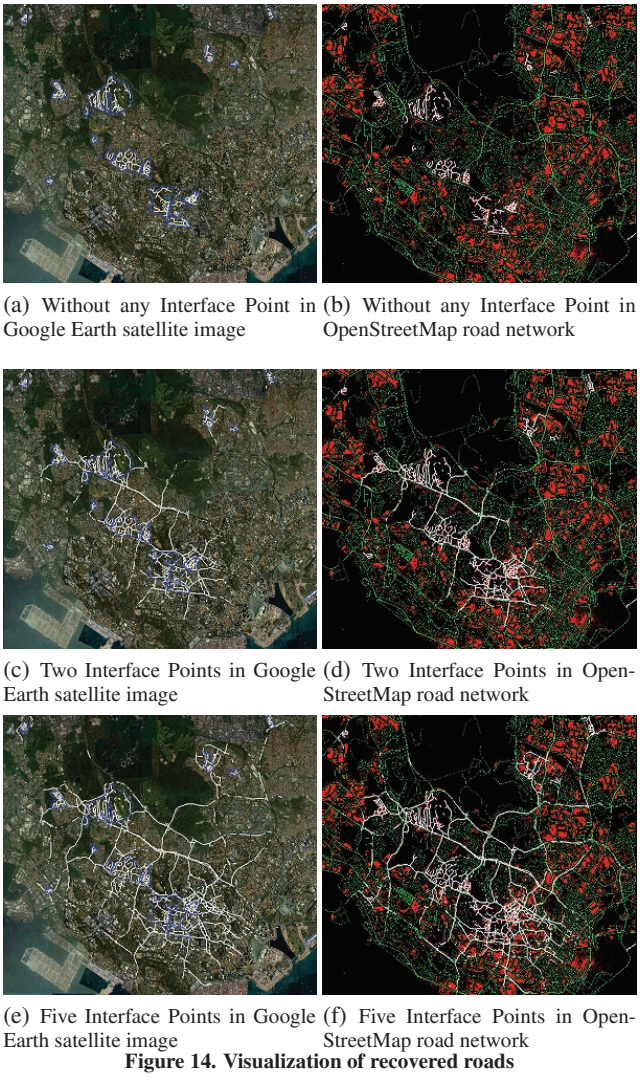


(e) TC1 - 30s (f) TC1 - 120s



(g) KDE - 30s (h) KDE - 120s

Figure 13. Visualization of generated roads using discrete dataset



(a) Without any Interface Point in Google Earth satellite image

(b) Without any Interface Point in OpenStreetMap road network

(c) Two Interface Points in Google Earth satellite image

(d) Two Interface Points in OpenStreetMap road network

(e) Five Interface Points in Google Earth satellite image

(f) Five Interface Points in OpenStreetMap road network

Figure 14. Visualization of recovered roads

inates the large loss of geometry information, thus *CrowdAtlas* maintains a stable recall performance. As for the *precision*, *CrowdAtlas* shows a fluctuation in 30s and 60s interval because some noise data exists in these two parts. COBWEB uses denoiser to eliminate most noise data. Consequently, it shows robustness on *recall* and *precision* performance again. *KDE* demonstrates a similar performance as COBWEB.

Figure 10 and Figure 11 report the robustness comparison of four algorithms under different data sizes based on *concentrated dataset* and *discrete dataset*, respectively. As data size becomes smaller, the clustering effectiveness in *CrowdAtlas* becomes less effective, which explains why *recall* falls gradually. For *KDE*, the data sparsity exposes its weakness. The drain of data eliminates more and more valid pixels in discretized image, and hence the structure of road network suffers significantly. For *TC1*, we witness an obvious drop trend in both *recall* and *F-Score* metrics. It means that the low data volume ruins the feasibility of *TC1* as it cannot get sufficient trajectories to support its clustering. Although the failure (empty results) of *TC1* in 1 day data size causes the sudden drop of *precision*, all four algorithms achieve relatively high *precision* in most cases. When considering the comprehensive metric *F-Score*, COBWEB is the most stable one in both datasets.

Demonstration

In our last set of experiments, we demonstrate the performance of COBWEB in terms of map update for the entire Singapore City. We use the trajectory data reported by $15,000$ taxis in Singapore within a 5-month period to update the road network from OpenStreetMap(OSM). OSM supplies a free access to use and edit map data to public. However, the update of new submission map data is implemented by organizing informal and small meetings to manually complete missing features (according to [11, 10]). Original dataset contains $1,550,000$ trajectories (20 billions GPS points). After map matching, $2,571,750$ trajectory segments ($21,713,000$ GPS points) remain unmatched and are collected as the input for map update. After the filtering of denoiser, $15,392,700$ GPS points remain. Finally, the total length of generated roads is around 108km. We visualize the roads recovered by COBWEB using white lines in Figure 14.

In Figure 14(a) and 14(b), we only plot the generated roads. In Figure 14(c) and 14(d), we plot generated roads together with the shorter interface roads via expanding each unmatched trajectory with 2 GPS points in both directions in the pre-processing step. In Figure 14(e) and 14(f) we plot the generated roads with longer interface roads via expanding each unmatched trajectory with 5 GPS points in both directions. More than 90% of interface roads overlap with the corresponding parts of the road network accurately, which means COBWEB scores a perfect precision rate in graph merging. All the red regions in Figures 14(b), 14(d), and 14(f) stand for those denoised GPS points. Note in Figure 14, we plot our recovered road segments using Google Earth satellite image and road network from OpenStreetMap as backgrounds.

CONCLUSION

COBWEB is proved to be a robust map update system. The robustness reflects in three aspects. First, it is robust to poor data quality. COBWEB expertises at recovering complex road networks from poor trajectory dataset with low sampling rate and remarkable GPS noise. Second, it is robust to sparse dataset. COBWEB is able to recover roads effectively regardless of the data volume. Third, it is robust to variety of roads absence. COBWEB outperforms the current best algorithms of both *Map Update* and *Map Inference* significantly. We desire to implement the road-deletion update ability as the next work, although the abandoned roads are extremely rare in the real world. In the future, we plan to publish COBWEB as an open source in order to maximize its usage.

ACKNOWLEDGMENTS

This research is supported in part by Microsoft Research Asia Collaborative Research Project under grant FY15-RES-THEME-011, National Natural Science Foundation of China (NSFC) under grant 61073001, Shanghai Natural Science Foundation under grant 14ZR1403100, Shanghai Science and Technology Development Funds (13dz2260200, 13511504300) and Project B1-30-03 in Student Scientific Innovation Act of Fudan University.

REFERENCES

1. Agamennoni, G., Nieto, J. I., and Nebot, E. M. Robust inference of principal road paths for intelligent transportation systems. *IEEE Transactions on Intelligent Transportation Systems 12*, 1 (2011), 298–308.
2. Ali, M., Krumm, J., Rautman, T., and Teredesai, A. ACM SIGSPATIAL GIS Cup 2012. In *Proc. ACM SIGSPATIAL GIS 2012* (2012), 597–600.
3. Biagioni, J., and Eriksson, J. Inferring road maps from gps traces: Survey and comparative evaluation. In *Proc. TRB 2012*, Citeseer (2012).
4. Cao, L., and Krumm, J. From GPS traces to a routable road map. In *Proc. ACM SIGSPATIAL GIS 2009* (2009), 3–12.
5. Chang, B.-J. Vehicle location and navigation systems. *Telematics Communication Technologies and Vehicular Networks: Wireless Architectures and Applications: Wireless Architectures and Applications* (2009), 119.
6. Chen, C., and Cheng, Y. Roads digital map generation with multi-track GPS data. In *Proc. ETT and GRS 2008*, vol. 1 (2008), 508–511.
7. Davies, J. J., Beresford, A. R., and Hopper, A. Scalable, distributed, real-time map generation. *IEEE Pervasive Computing 5*, 4 (2006), 47–54.
8. Diggelen, F. v. Gnss accuracy-lies, damn lies, and statistics. *GPS World 18*, 1 (2007), 26–33.
9. Edelkamp, S., and Schrödl, S. Route planning and map inference with global positioning traces. In *Computer Science in Perspective*. Springer, 2003, 128–151.
10. Haklay, M., et al. How good is volunteered geographical information? a comparative study of openstreetmap and ordnance survey datasets. *Environment and planning. B, Planning & design 37*, 4 (2010), 682.
11. Haklay, M., and Weber, P. Openstreetmap: User-generated street maps. *Pervasive Computing, IEEE 7*, 4 (2008), 12–18.
12. Liu, X., Biagioni, J., Eriksson, J., Wang, Y., Forman, G., and Zhu, Y. Mining large-scale, sparse GPS traces for map inference: comparison of approaches. In *Proc. ACM SIGKDD 2012* (2012), 669–677.
13. Liu, X., Zhu, Y., Wang, Y., Forman, G., Lionel, M. N., Fang, Y., and Li, M. Road recognition using coarse-grained vehicular traces. Tech. rep., Technical Report HPL-2012-26, HP Labs, 2012.
14. Lou, Y., Zhang, C., Zheng, Y., Xie, X., Wang, W., and Huang, Y. Map-matching for low-sampling-rate GPS trajectories. In *Proc. ACM SIGSPATIAL GIS 2009* (2009), 352–361.
15. Newson, P., and Krumm, J. Hidden markov map matching through noise and sparseness. In *Proc. ACM SIGSPATIAL GIS 2009* (2009), 336–343.
16. Niehoefer, B., Burda, R., Wietfeld, C., Bauer, F., and Lueert, O. GPS community map generation for enhanced routing methods based on trace-collection by mobile phones. In *Proc. SPACOMM 2009* (2009), 156–161.
17. Ochieng, W. Y., Quddus, M., and Noland, R. B. Map-matching in complex urban road networks. *Revista Brasileira de Cartografia 2*, 55 (2003).
18. Phuyal, and Bishnu, P. Method and use of aggregated dead reckoning sensor and gps data for map matching. In *Proc. ION GPS 2002* (2001), 430–437.
19. Schroedl, S., Wagstaff, K., Rogers, S., Langley, P., and Wilson, C. Mining GPS traces for map refinement. *Data mining and knowledge Discovery 9*, 1 (2004), 59–87.
20. Shi, W., Shen, S., and Liu, Y. Automatic generation of road network map from massive GPS, vehicle trajectories. In *Proc. ITSC 2009* (2009), 1–6.
21. Song, R., Lu, W., Sun, W., Huang, Y., and Chen, C. Quick map matching using multi-core cpus. In *Proc. ACM SIGSPATIAL GIS 2012* (2012), 605–608.
22. Steiner, A., and Leonhardt, A. A map generation algorithm using low frequency vehicle position data. *Transportation Research Board, 90th Annual* (2011).
23. Tan, P.-N., Michael, S., and Vipin, K. Introduction to data mining. 666–667.
24. Wang, Y., Liu, X., Wei, H., Forman, G., Chen, C., and Zhu, Y. Crowdatlas: self-updating maps for cloud and personal use. In *Proc. MobiSys 2013* (2013), 27–40.
25. White, C. E., Bernstein, D., and Kornhauser, A. L. Some map matching algorithms for personal navigation assistants. *Transportation Research Part C: Emerging Technologies 8*, 1 (2000), 91–108.
26. Worrall, S., and Nebot, E. Automated process for generating digitised maps through gps data compression. In *Australasian Conference on Robotics and Automation* (2007).
27. Zheng, Y. Trajectory data mining: an overview. *ACM Transactions on Intelligent Systems and Technology (TIST) 6*, 3 (2015), 29.