RoadRunner: Improving the Precision of Road Network Inference from GPS Trajectories

Songtao He[1], Favyen Bastani[1], Sofiane Abbar[2], Mohammad Alizadeh[1],
Hari Balakrishnan[1], Sanjay Chawla[2], Sam Madden[1]
[1]MIT CSAIL, [2]Qatar Computing Research Institute, HBKU
[1]{songtao,fbastani,alizadeh,hari,madden}@csail.mit.edu, [2]{sabbar,schawla}@hbku.edu.qa

ABSTRACT

Current approaches to construct road network maps from GPS trajectories suffer from low precision, especially in dense urban areas and in regions with complex topologies such as overpasses and underpasses, parallel roads, and stacked roads. This paper proposes a two-stage method to improve precision without sacrificing recall (coverage). The first stage, RoadRunner, is a method that can generate high-precision maps even in challenging scenarios by incrementally following the flow of trajectories, using the connectivity between observations in each trajectory to decide whether overlapping trajectories are traversing the same road or distinct parallel roads, and to correctly infer road segment connectivity. By itself, RoadRunner is not designed to achieve high recall, but we show how to combine it with a wide range of prior schemes, some that use GPS trajectories and some that use aerial imagery, to achieve recall similar to prior schemes but at substantially higher precision. We evaluated RoadRunner in four U.S. cities using 60,000 GPS trajectories, and found that precision improves by 5.2 points (a 33.6% error rate reduction) and 24.3 points (a 60.7% error rate reduction) over two existing schemes, with a slight increase in recall.

CCS CONCEPTS

• **Information systems** → **Spatial-temporal systems**; **Geographic information systems**; **Sensor networks**; • **Applied computing** → **Cartography**;

KEYWORDS

Road Network, GPS, Map Inference, Spatial Data, Trajectory

1 INTRODUCTION

The availability of accurate and up-to-date road maps is critical for many applications, including GPS-based navigation services, disaster relief efforts, and autonomous transportation. However, creating and maintaining road network maps is currently human-intensive, expensive, and slow. Thus, automating parts or all of the map creation and maintenance process holds the potential to benefit many applications by providing maps that quickly reflect road network changes. Crowdsourced GPS trajectories are a useful data source for this task. These trajectories are sequences of GPS observations collected as vehicles travel along the road network, and are available at scale nowadays from smartphones.

Inferring road network maps from GPS data has received considerable attention in recent years [2, 6, 12, 15]. Broadly, existing inference schemes infer a high-coverage but low-accuracy initial graph, and then apply refinement heuristics (e.g., graph spanners pruning [19] and k-means refinement [7]) to improve precision. However, we find that these methods fail to produce high quality maps, especially in dense urban areas and in areas with complex intersections; these areas often have large volumes of traffic and play an important role in a city's road network. In the two left panels of Figure 1, we show the results from two such approaches on selected regions of three cities.

The resulting maps suffer from three significant problems:

- They connect overpasses with underpasses in a planar graph, despite the underlying techniques using heuristics to avoid these spurious connections.
- They connect adjacent roads that do not intersect.
- They fail to capture detailed topology such as highway interchanges.

Because prior schemes developed over a period of several years suffer from these problems, we believe that strategies that start with a low-accuracy initial graph and then refine it will not be able to regain high precision.

Thus, in this paper, we turn the strategy on its head. We propose a two-stage architecture that focuses on *precision first*, and *recall second*. In the first stage, we infer a high-accuracy road network to accurately capture road topology in complex regions like overpasses/underpasses, stacked roads, parallel roads, multi-road intersections, and dense urban areas. Then, in the second stage, rather than developing new refinement heuristics, our approach regains recall by simply running any of several prior schemes, but taking care not to disrupt the segments and interconnections computed in the first stage. The result is much higher precision with recall similar to the second-stage scheme.

However, accurately inferring road topology in complex regions is difficult even when high recall is not a concern. For example, in Figure 2, if we look only at a point cloud of GPS observations across the trajectories (where we discard the connectivity between observations defined by the trajectories), we cannot discern whether the two roads meet. Thus, the common technique of taking a histogram over observations (*kernel density estimation* [11]), in which each cell is weighted by the number of GPS samples contained in the cell, does not reveal the underlying network topology. On the other

SIGSPATIAL '18, November 6–9, 2018, Seattle, WA, USA

ACM ISBN 978-1-4503-5889-7/18/11…$15.00
https://doi.org/10.1145/3274895.3274974

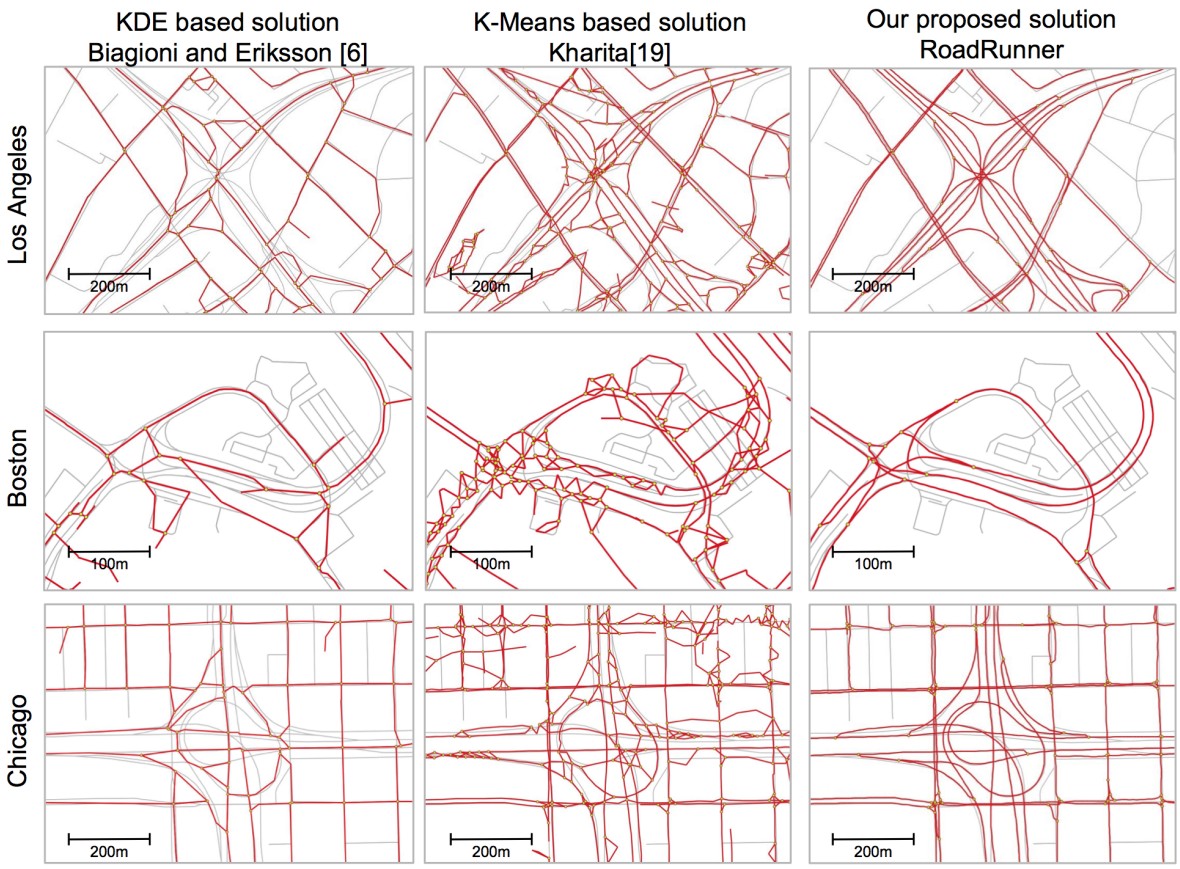

Figure 1: Motivation. GPS-trajectory based mapmaking algorithms running on complex interchanges in major cities. Red lines show algorithm output overlaid on underlying ground truth map in gray. Two existing algorithms are on left, our proposed RoadRunner algorithm is on right. Both existing approaches connect parallel segments on highways, often with many small, spurious crossing segments, and also add incorrect intersections where roads of different heights cross.

hand, the topology becomes clear if we bring back the trajectory connectivity: because no trajectory that starts from the left road exits along the right road and vice versa, we can conclude that the two roads do not intersect.

We present *RoadRunner*, a method that exploits trajectory connectivity to generate accurate maps in challenging regions where existing methods fail. RoadRunner constructs a road network graph *incrementally* by following the flow of GPS trajectories. This iterative process enables RoadRunner to consider GPS observations on each iteration not in isolation, but in the context of several predecessor and successor observations in the same trajectory. This context, which we use in a trajectory filtering algorithm, is crucial to precisely separating out nearby but distinct roads, and doing so in a way that is robust to GPS noise and complex road topologies. The right panel of Figure 1 exemplifies how RoadRunner produces maps with significantly higher precision than prior work.

Although our incremental procedure and filtering strategy allow RoadRunner to infer roads with high precision, they also cause RoadRunner to miss roads that are covered by few GPS trajectories, such as roads in residential areas. Fortunately, existing methods already perform well in such regions. Thus, after running RoadRunner to produce a high-precision map covering the most challenging areas of the road network, we run an existing high-recall method in the second stage. Then, we apply a merging procedure to identify segments found by the high-recall method but not by RoadRunner, and integrate them into the inferred map.

In summary, we make the following contributions:

- We propose a two-stage map inference architecture that enables us to generate high precision road network graphs without sacrificing the coverage. At the core of this architecture is RoadRunner, our high-precision first-stage method that uses the connectivity of GPS trajectories to produce accurate maps in dense urban areas and at complex intersections where current approaches perform poorly.
- We show how to integrate several prior schemes as the second stage in our two-stage architecture with a merging procedure to combine the road segments inferred by RoadRunner with those inferred by existing approaches. We apply the merging procedure to two current state-of-the-art GPS-based approaches, Biagioni-Eriksson (BE) [6] and Kharita [19], and to RoadTracer [5], which processes aerial imagery.
- We evaluate our two-stage architecture over 4 km × 4 km regions of four U.S. cities containing 1,864 kilometers of roads on a GPS trajectory dataset of over 60,000 trajectories. We find that our proposed two-stage architecture significantly improves the quality of the inferred maps compared with the

state-of-art solutions. We summarize the evaluation result in table 1.

Schemes	Precision	Recall	Routing Error
BE	84.5%	37.2%	76.1 meters
RoadRunner+BE	89.7%	42.8%	24.8 meters
Kharita	60.0%	43.0%	73.7 meters
RoadRunner+Kharita	84.3%	44.7%	41.4 meters

Table 1: Summary of the evaluation results

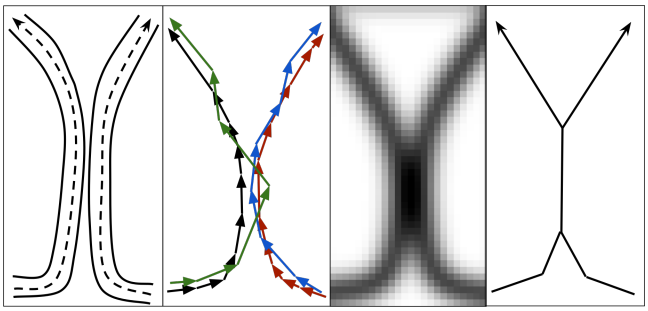

Figure 2: Illustration of the importance of accounting for the association between observations in a trajectory. Two roads that pass near each other but do not meet are shown on the left. In the center, we show a set of trajectories that pass these roads, along with an example histogram over GPS observations in those trajectories. We cannot tell whether the roads intersect from the histogram, although it is clear from the original trajectories. Existing approaches often produce a road network graph similar to the one on the right.

2 RELATED WORK

The rapid adoption of GPS-enabled smartphones has led to considerable interest in the problem of automatic road network inference from GPS trajectories [2, 7, 15]. However, as we demonstrated in Figure 1, prior schemes yield noisy maps with low precision on complex topologies such as highway interchanges. We show that precision can be improved by exploiting the connectivity between GPS observations at different times along the same trajectory.

Broadly, road network inference schemes can be divided into three categories [7]. *k-means* approaches begin by clustering the GPS observations. Cluster centers become vertices in the inferred road network graph, and edges are added between the clusters of successive observations in each trajectory [1, 13, 17]. During clustering, the longitude, latitude, heading, and speed of each observation may be considered, but the connectivity between observations in the same trajectory is ignored. Although this connectivity is used to add edges, this stage still only considers pairs of consecutive observations rather than longer trajectory subsequences.

Kernel density estimation (KDE) approaches first generate a spatial histogram where each cell is weighted by the number of GPS trajectories that pass through the cell. Kernel smoothing is applied on the histogram (typically using a Gaussian distribution), followed by morphological thinning or similar skeletonization techniques to extract centerlines and produce a graph [9, 11, 18]. These techniques do not consider the connectivity between GPS observations.

Trajectory merging approaches merge GPS trajectories one at a time into an initially empty road network graph [3, 8]. Again, while these approaches iterate over a trajectory to merge it into the graph, the merging process operates only on pairs of successive observations at a time.

Recently, a number of methods have been proposed that combine or extend these techniques. Biagioni et al. [6] propose a hybrid pipeline using KDE with an adaptive thresholding scheme to obtain an initial road network graph, followed by geometry and topology refinement and map-matching-based pruning to further improve accuracy. Chen et al. [10] propose a supervised learning framework that leverages prior knowledge on real-world road networks to learn the shape of different junctions, and integrate this with a cluster based algorithm. Stanojevic et al. [19] develop a novel model in which map construction is framed as a network alignment problem. The derived optimization problem is then mapped into a hybrid algorithm combining k-means clustering and graph spanners. Zheng and Zhu [21] propose a revisited version of the trace merging method, applying a novel clustering algorithm that uses a partial curve matching method based on Fréchet distance to measure the partial similarity between any trajectory and a previously created link. While these methods improve on earlier schemes, none utilize the long-term connectivity between observations in the same trajectory when constructing the road network graph.

Other data sources, aerial imagery in particular, have also been leveraged for automatic road network inference. DeepRoadMapper [16] applies CNN-based segmentation followed by an extensive post-processing pipeline to extract a road network graph from aerial imagery. RoadTracer [5] improves on DeepRoadMapper [16]. It uses an iterative graph construction strategy to obtain a graph directly from a CNN, thereby attaining higher precision that segmentation-based approaches. However, because of occlusion by tall buildings, shadows, and overpasses, accurately inferring roads from aerial imagery in dense urban areas and complex intersections is challenging—indeed, these methods assume a fully planar road network graph, and thus yield low precision in these regions.

3 ROADRUNNER

RoadRunner iteratively constructs the road network from an initial graph by following the flow of GPS trajectories. We show the structure of RoadRunner in Algorithm 1. The algorithm begins with an initial graph; this may be obtained from an existing graph or a graph inferred by another approach. We first push all the vertices in this initial graph into a queue Q. We call the vertices in the queue *active vertices*. On each iteration, RoadRunner picks an active vertex from the queue and extends the current graph from this active vertex by performing two key operations — adding new road segments (tracing), or connecting the active vertex with an existing vertex when two physical roads join together (merging).

We show an example of how RoadRunner works in Figure 3. The algorithm starts with an initial graph G which has only one single vertex (the green vertex near the upper-right corner). We visualize the partially constructed graph and the active vertices at different iterations. The construction procedure uses the GPS data to gradually extend the graph by following the road network, including forking at intersections and merging when two roads come together. The algorithm stops tracing a road when it reaches a dead end, leaves the region of interest, or joins with other roads.

Figure 3: We visualize the partially constructed graph G (red) and the active vertices queue Q (blue) from different iterations in the construction procedure. In this example, the algorithm begins with a single vertex (green) at the upper-right corner of the region.

Algorithm 1 RoadRunner. The asymptotic complexity of this algorithm is $O(L \cdot D)$, where L is the total length of the road network and D is the average number of GPS points explored at each step (e.g., within 100 meters)

procedure RoadRunner(InitialGraph)
  $G \leftarrow$ InitialGraph
  $Q \leftarrow$ Push all vertices of G into a queue
  while Q is not empty **do**
    $v \leftarrow Q.pop()$
    $\mathcal{D} \leftarrow$ **Trace**(G, v) ▷ Return the direction(s) of the GPS flow(s) around vertex v.
    for each direction θ in $\mathcal{D}$ **do**
      $loc \leftarrow v.loc + d \cdot (cos\theta, sin\theta)$
      $u \leftarrow G.initVertexAtLocation(loc)$
      $u.predecessor \leftarrow v$
      $G.addEdge(v, u)$
      $succ \leftarrow$ **Merge**(G, u) ▷ Try to merge the vertex u with the current graph G, return the state of merging.
      if $succ$ is False **then**
        $Q.push(u)$
  return G as the inferred road network graph

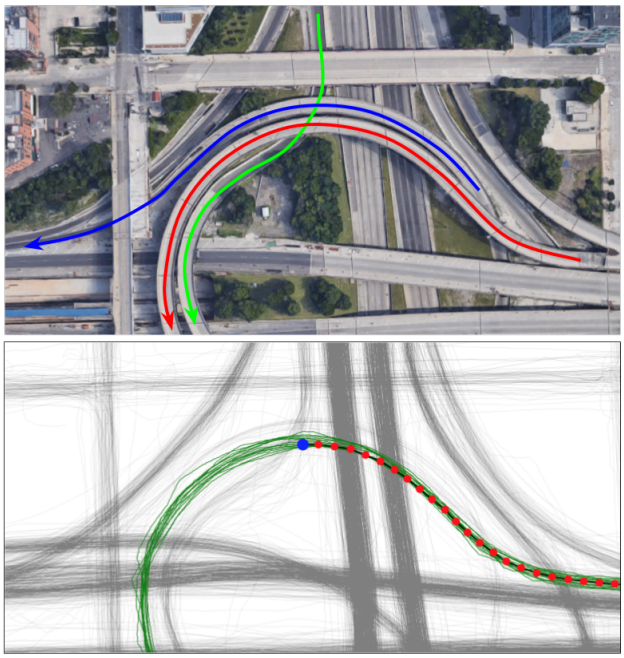

Figure 4: We show the GPS trajectories (grey) near a complex highway junction as well as the corresponding aerial image of the same region. The red vertices represent the partially constructed graph and the blue vertex is the active vertex from where we are going to extend the graph. We highlight the trajectories that pass near the partially constructed graph in green. In the aerial image, we highlight three roads that are challenging for map inference algorithm.

Finally, after the search queue is empty, we obtain the road network graph for the whole region.

Constructing the road network graph in an iterative manner enables RoadRunner to exploit the *long-term connectivity* between GPS observations to accurately capture road geometry and topology as it adds each segment to the graph. By contrast, most prior approaches rely on a histogram over observations [11], the position of individual or pairs of GPS observations [8], or the local heading of observations [10, 12, 19] to construct the road graph. Some prior approaches use long-term trajectory information *after* an initial road network graph has been produced. For example, BE [6] leverages the trajectory sequences in a post-processing phase to prune edges from an initial low-accuracy graph. However, refining a low-accuracy graph is very difficult, particularly, in regions with complex road topology or high GPS noise.

To robustly trace and merge roads, at each iteration, RoadRunner compares trajectory sequences with the partially constructed graph to filter trajectories that are not related to the road of interest. Consider Figure 4, where we are extending the graph from the blue vertex corresponding to a highway ramp. Since the three highlighted roads are close to each other and have almost the same heading, if we take GPS observations from all of the trajectories into account, we may connect the underpass and overpass of the green road and the red road, or merge the red road with the blue road. However, by excluding trajectories that do not pass near a sequence of edges in the partially constructed graph corresponding to the current road, we obtain a much cleaner subset of GPS trajectories

that covers only the red road. Our iterative architecture enables us to construct such a subset of trajectories on each step, and to use it to accurately capture the road network.

To realize this idea, we introduce a primitive called a *way path filter*. Given a sequence of circles centered at locations along a road, the way path filter prunes the GPS trajectory data to retain only trajectories that pass through the circles in order. Thus, the circles act as waypoints that we require the trajectories to traverse.

Formally, given the trajectories T and a sequence of circles $P = \langle (p_1, r_1), (p_2, r_2), \ldots, (p_n, r_n) \rangle$, where the ith circle is centered at p_i and has radius r_i, the way path filter is a function $filter(T, P)$ that computes a subset of T containing trajectories that pass through the circles in P sequentially. A trajectory $t \in T$ is in this subset if there exists a subsequence of GPS observations in the trajectory, $\langle t_{j_1}, \ldots, t_{j_n} \rangle$ such that the distance from t_{j_i} to p_i is at most r_i for all i.

For a particular active vertex, we apply the way path filter to retain only trajectories that are likely to have traversed the road corresponding to the active vertex. To do so, we compute a path in G of length k that terminates at the active vertex, and then apply the filter with circles drawn around vertices along this path. The radius of these circles should correspond to the width of the road at the vertex, so that the circle covers all trajectories that traverse the road. We estimate this width from the trajectory data (Section 3.3.1).

The length of the path, k, impacts the precision of the inferred map. A larger k applies a filter through a longer path, enabling more accurate tracing and merging operations. In practice, we set k to be 75 meters, as we find this is long enough for us to distinguish the most challenging road structures. In fact, in our experiments, we find that increasing k yields higher precision but slightly lower recall. After k exceeds 75 meters, precision no longer improves.

Next, we discuss RoadRunner's tracing operation (Section 3.1), merging operation (Section 3.2), and some implementation details (Section 3.3).

3.1 Tracing

The tracing operation extends the road network graph by adding new vertices and edges to an active vertex in the direction of the GPS trajectories that pass through that vertex. RoadRunner simultaneously follows all peak directions indicated by the trajectories. It predicts these directions using Algorithm 2.

Consider the example in Figure 5, where we are predicting the peak directions of GPS trajectories near the active vertex v (blue). Let $P = \langle (p_0, r_0), (p_1, r_1), \ldots, (p_{k-1}, r_{k-1}) \rangle$ be a sequence of circles centered at vertex v and its $k-1$ predecessors in the partially constructed graph (line 2-3). In the figure, we highlight the trajectories in $filter(T, P)$. We can clearly see that the trajectories split into three groups at the intersection.

To predict the directions of these outgoing groups of trajectories, we check 72 evenly spaced angles from 0 to 2π. For each direction θ, we create a circle $w = (v + D(cos\theta, sin\theta), r)$ located at a distance D from v in the direction θ. We count the number of GPS trajectories that pass near the path $\mathbf{P} + \langle w \rangle$ using the way path filter (line 7). We subtract the counted number by M. The parameter M helps control

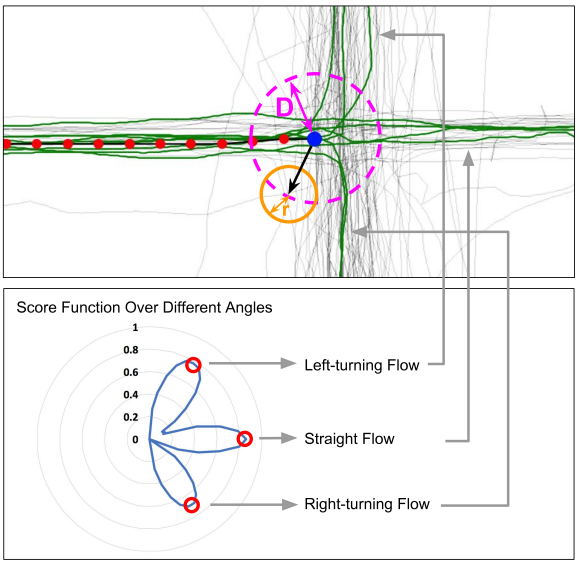

Figure 5: We show an example of the tracing operation near a four-way intersection. In the above figure, the red vertices are the partially constructed graph and the blue vertex is the active vertex. We highlight the GPS trajectories that pass near this partially constructed graph in green. The green GPS flow splits into three groups at the intersection. We predict the directions of these outgoing GPS flows through a score function. Here, we can clearly see three local peaks in the score function over different angles.

Algorithm 2 Trace Operation

```
1:  procedure TRACE(Graph G, Vertex v)
2:      u = ⟨u_i⟩ ← the first k − 1 predecessors of v.
3:      P ← the circle sequence of path u + ⟨v⟩
4:      N, Score ← arrays indexed by different angles.
5:      for each θ in evenly spaced angles from 0 to 2π do
6:          w ← circle (v.loc + D · (cosθ, sinθ), r)
7:          T′ ← filter(T, P + ⟨w⟩)
8:          N[θ] ← min(0, |T′| − M)
9:      Score ← smooth(N)
10:     return angles of local peaks of Score[θ]
```

the precision of constructed maps by excluding low confidence (low density) GPS flows.

We define the score function as the smoothed version of this counter over different angles. In the *smooth* function, we convolve the input array with a Gaussian kernel. Smoothing removes small changes in the score function and lets us focus only on major peaks. We extract the directions of all outgoing GPS flows by identifying local peaks in the smoothed score function. We show an example in Figure 5. There are three local peaks in the score function, corresponding to the left-turning flow, the straight flow, and the right-turning flow. After we get the directions of the outgoing flows, we extend the graph G by adding new vertices at a small fixed distance from v towards each direction.

3.2 Merging

When the construction procedure encounters a road segment that has already been explored in the graph G, we need to merge the current road with the previous path. However, merging is challenging: we need to merge roads that connect while ensuring that overpasses/underpasses, parallel roads, and multilayer roads remain separated. Correctly capturing connectivity is crucial because even a small number of incorrect connections lead to a large number of navigation errors.

Existing approaches [6, 19] aggressively merge road segments, considering only local information such as distance and heading. This yields numerous incorrect connections in challenging regions like dense urban areas and highway intersections. We find that deciding whether we should merge two road segments with only local information such as distance and heading is not enough. As the example shown in Figure 6, a universal merging threshold may not exist if we only consider the local information.

To overcome this challenge, we introduce a novel merging criteria that can accurately decide whether road segments should merge or not. When two road segments are close to each other, instead of only considering the local information of these two road segments such as distance and heading, we look at the GPS trajectories that pass through these two road segments. We merge the road segments only if the distributions of these two groups of GPS trajectories match in the future (after traveling past the two road segments).

In Figure 6, we show the distributions of filtered GPS trajectories that pass the blue and green road segments. We can clearly see that the two distributions of example (a) disagree with each other, while the two distributions are very similar in example (b). This design enables us to make correct decisions on whether the two roads should be merged or not in a very general way, supporting a wide range of road types. Again, we use the partially constructed graph as a trajectory filter to generate the two distributions. The details of the merging algorithm are shown in Algorithm 3.

Algorithm 3 Merge Operation

```
1:  procedure MERGE(Graph G, Vertex v)
2:      D_v ← GenDistribution(G, v)
3:      for each vertex u near v do
4:          D_u ← GenDistribution(G, u)
5:          if D_u is similar with D_v then
6:              G.addEdge(u, v)
7:              return True
8:      return False
9:  procedure GENDISTRIBUTION(Graph G, Vertex v)
10:     P_v ← the path formed by v's predecessors
11:     T_v ← filter(T, P_v)
12:     T'_v ← for each t in T_v, we only keep a portion of it; this
        portion starts near v and continues for a fixed distance, e.g.,
        300 meters.
13:     return the spatial distribution of T'_v
```

3.3 Implementation Details

In this section, we briefly discuss two implementation details: vertex radius estimation and tracing acute branches.

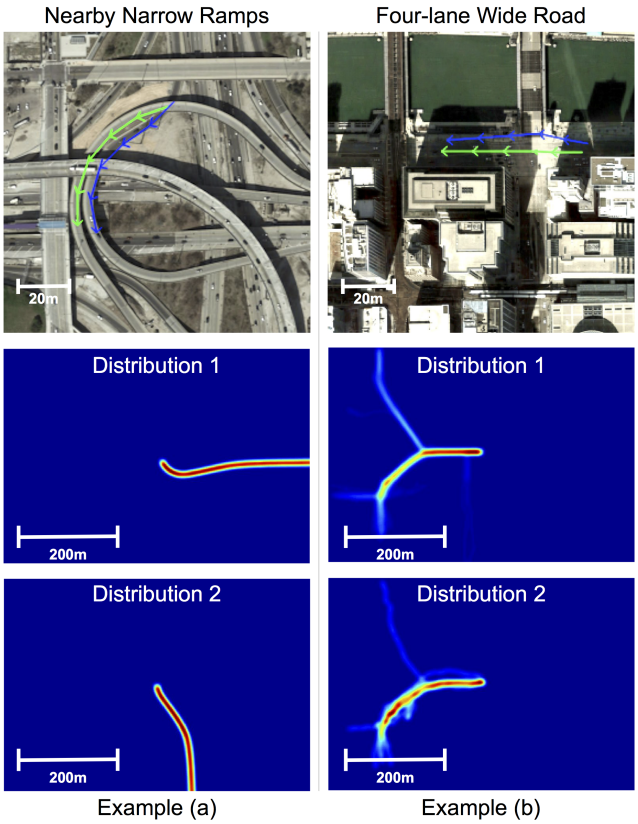

Figure 6: Suppose we are considering to merge the green road and the blue road in the above two examples. Simply considering the distance between the two roads cannot yield valid results. In fact, the distance between the two roads in example (b) is larger than the distance in example (a). However, we need to merge the two roads in example (b) rather than the two roads in example (a). Below the satellite images, we show the distribution of GPS trajectories after they pass through the blue road segments (distribution 1) and the green road segments (distribution 2).

3.3.1 Vertex Radius Estimation. Each vertex in the road network graph has a *radius* attribute. Recall that in the *way path filter*, we filter GPS trajectories based on this radius attribute. Instead of using a fixed radius for all vertices, we estimate this radius for each vertex during the graph construction process. The basic idea is to estimate the width of the road at vertex v from the GPS trajectories and use that as the radius.

Specifically, we consider the vertex sequence from v's $2k$th predecessor to its kth predecessor. Then, we apply the way path filter on this vertex sequence. We look at the distribution of the filtered trajectories along the perpendicular direction of the edge $v.predecessor \rightarrow v$. Here, we assume this distribution follows Gaussian mixture model, where the component closest to the center line corresponds to the road through $v.predecessor \rightarrow v$, while other components capture nearby roads that may have just forked from the main road. We estimate the number of the mixture components as well as the mean and standard deviation of each component. Then we use the standard deviation of the component located

closest to the center line as the radius of vertex v. Note that this estimation algorithm depends on the radius of v's predecessors. Thus, we set the radius of all vertices in the initial graph to a fixed initial value, e.g., 5 meters.

This dynamic radius estimation solution allows the way path filter to remain effective on roads of diverse widths and GPS noise levels.

3.3.2 Tracing Acute Branches . The tracing algorithm in Section (3.1) may fail to trace all branches when the angle between two branches is very small. This often happens on highway offramps, where the large traffic volume disparity between the highway and the offramp, along with the small angle of the branch, make it hard to identify the GPS flow through the ramp. We solve this problem by re-using the Gaussian mixture components estimated in the vertex radius estimation (Section 3.3.1). In the road width estimation, each mixture component, located along the perpendicular direction of the current road, represents a nearby road that just forked from the current road. We check all these nearby roads. If they are not covered by the road network graph, we add them to the graph and push their last vertices into the active vertices queue.

4 TWO-STAGE MAP INFERENCE

As shown in Figure 1, RoadRunner can capture complex portions of the road network with high accuracy. However, we find that RoadRunner may miss roads in regions that are covered by very few GPS trajectories, such as residential areas. The reason is that RoadRunner aggressively filters trajectories that don't conform to the structure of the currently explored road at each step of the search; this strategy enables very high precision but misses lightly covered roads. Fortunately, these are the very regions where current state-of-the-art inference algorithms excel.

Thus, we develop a merging procedure to get the best of both worlds. We obtain a highly accurate road network graph covering noisy and topologically-complex areas from RoadRunner, and then merge segments inferred by an existing approach that correspond to new roads.

Let G_1 be a road network graph inferred by RoadRunner, and G_2 be one inferred by an existing scheme that captures sparsely covered roads. We prune edges and portions of edges in G_2 that lie within R_{merge} meters of G_1 to obtain G_2'; this eliminates segments corresponding to roads that RoadRunner has already captured. We then compute G as the union of G_1 and G_2'. However, roads added from G_2' will be disconnected from the rest of the road network. To add back connections, we iterate through dead-end vertices in G, i.e., vertices with exactly one incident edge. We merge a dead-end vertex v with an edge (u, w) if both of the following hold:

- The distance from v to (u, w) is less than R_{merge}.
- $|filter(T, \langle v, p, u\rangle)|$ or $|filter(T, \langle v, p, w\rangle)|$ exceed a threshold, where p is obtained by projecting v onto the line segment (u, w).

Together, these conditions prevent the introduction of spurious connections.

5 EVALUATION

In this section, we compare our two-stage map inference scheme with RoadRunner against two GPS trajectory map inference algorithms, BE [6] and Kharita [19], and one aerial imagery inference approach, RoadTracer [5], on 16 sq km regions of four cities. BE applies kernel density estimation followed by several refinement steps. Kharita uses graph spanners to prune redundant edges from an initial graph constructed with k-means clustering. RoadTracer is the state-of-the-art computer vision approach that infers roads from aerial imagery.

5.1 Dataset

We evaluate the approaches on a large dataset of over 60 thousand GPS trajectories spanning 4km by 4km regions at the centers of four cities: Los Angeles, Boston, Chicago, and New York City. We use these regions as our evaluation dataset because:

- These regions contain diverse road structures, from complex highway interchanges to small residential roads, and a wide range of GPS noise, from near-perfect GPS accuracy in open areas to heavy noise in the downtown core. We believe this dataset reflects the challenges of automatic map inference algorithms in real world.
- These regions (urban areas) resembles rapidly developing cities such as Doha (Qatar), where digital maps lag reality. In these cities, the construction of new complex road structures such as highway junctions is also very common.
- These regions have good-quality ground truth map from OpenStreetMap [14], which makes quantitative evaluation possible.

5.2 Metrics

We evaluate inferred maps on two metrics: TOPO [7], which is commonly used in related work, and a shortest-path metric, which we develop by combining ideas from several existing path-comparison-based metrics [2, 20].

5.2.1 TOPO. TOPO captures both geometry and topology differences between two maps. We first drop "seeds" at 50-meter intervals on every road in the ground truth map. For each seed, we try to find a corresponding point in the inferred map with similar distance and orientation (i.e., the angles of the edges that the seed and point fall on). If there exists such a point, we say the seed is valid.

For each valid seed, we drop "holes" every 5 meters on the edges of the ground truth map that can be reached within 300 meters from the seed. We then drop "marbles" in the same way from the nearest corresponding point in the inferred map. Then, we compute the maximum one-to-one matching between the marbles and holes. A marble and a hole can be matched only if the distance between them is smaller than 15 meters and the difference between the orientations of the edges they belong to is smaller than 45 degrees. From this matching, we compute a precision and recall for the seed:

$$precision = \frac{\#\ of\ matched\ marbles}{\#\ of\ all\ marbles} \quad recall = \frac{\#\ of\ matched\ holes}{\#\ of\ all\ holes}$$

We compute the overall precision and recall in a region as the average of the precision and recall from all seeds in this region. In the computation of the overall precision, we ignore the invalid

seeds in the region. In contrast, we consider the recall of an invalid seed to be 0 in the computation of the overall recall of a region. This yields a fair comparison when two inferred maps have different numbers of valid seeds.

5.2.2 Shortest-path metric. We propose a shortest-path metric to evaluate the correctness of navigation routes in the inferred map. In each city, we randomly sample 10,000 origin-destination pairs using the GPS trajectory data; each origin and destination is sampled uniformly from the origins and destinations of the trajectories. Then, for each sampled pair, we find the closest points to the origin and the destination in the inferred map. We compute the shortest path $P_{inferred}$ in the inferred map between these two points.

To evaluate the correctness of this path, we find the path P_{GT} in the ground truth map that minimizes the Fréchet distance [4] between $P_{inferred}$ and P_{GT}. If $P_{inferred}$ uses an invalid road or connection (e.g., the path jumps to a highway road from another non-connected road), the corresponding portion of P_{GT} may involve a long detour, yielding a large Fréchet distance between the two paths.

To aggregate results across all origin-destination pairs, we define the routing error as the median Fréchet distance.

5.3 Parameters

For all the schemes we evaluated, we fix most of their parameters except one tuning parameter, which we vary to explore the trade-off between precision and recall (Table 2). We explain the choice of this parameter for each different scheme below.

RoadRunner (RR). We vary the threshold of the minimum trajectory number M in the scoring function (recall that we only add a new edge toward angle θ if there are at least M trajectories on this direction). Increasing this threshold makes RoadRunner more cautious when adding new roads.

Biagioni and Eriksson (BE). During map-matching-based pruning, roads with fewer than L_{mm} matched GPS trajectories are removed from the road network graph. We vary L_{mm}. When L_{mm} is higher, more roads are pruned. The authors set $L_{mm} = 2$.

Kharita. We vary the initial seed radius for k-means clustering. This radius corresponds to the maximum degree of GPS noise that Kharita can handle. A larger radius increases precision by merging noisy GPS observations with other observations from the road, but may also merge observations from two nearby roads. The authors propose two seed radiuses, 20m and 75m.

In the following sections, we denote particular parameter settings by suffixing the approach name with this setting. For example, in BE-2, we set the minimum number of matched trajectories $L_{mm} = 2$.

5.4 Geometry and Topology Correctness

In this section, we focus on the geometry and topology correctness of the inferred map. We evaluate BE, Kharita and RoadRunner schemes as well as the RoadRunner-based two-stage schemes with TOPO metric. The evaluation covers all the GPS data in our dataset (approximately 4km by 4km for each city.)

In the two-stage map inference scheme, we first use RoadRunner (Section 3) to generate the high precision map, covering the most challenging areas. Then, we use the algorithm introduced in Section 4 to merge this high precision map with maps generated by BE or Kharita.

Scheme	Parameter Notation	Parameter Set
BE	Minimum GPS traversals	2, 5, 10, 30, 50
Kharita	Initial seed radius	20, 50, 75, 100
RoadRunner	Minimum GPS traversals	2,3,4,6,10

Table 2: The parameter set we used for different schemes

Error-Rate Frontier of Different Schemes. To obtain a comprehensive understanding of the potential performance of all the schemes, we evaluated the schemes with a wide range of parameters. The parameters we used for all the schemes are shown in Table 2. For two-stage schemes, we evaluate them with all the pairs of parameter combinations.

In Figure 7, we show the error-rate frontier of different schemes. The error-rate frontier of a scheme is generated from the output maps with all its different parameter configurations. We would like to use this error-rate frontier to demonstrate the best achievable performance of different algorithms. More specifically, we can compare the error rates of different schemes conditioned on recall by looking at the cross points of the schemes' error-rate frontiers and a horizontal line corresponding to a certain recall.

As shown in Figure 7, tuning the parameters of the existing map inference algorithms can yield improvements in error rate. However, due to the inherent limitations of these algorithms, the improvement in the error rate often comes at the expense of a sharp decrease in recall. The best achievable error rate is also limited.

By contrast, as a standalone scheme, RoadRunner can achieve a significantly lower error-rate compared with other schemes in all four cities. However, since RoadRunner sometimes fails to infer roads when there are not enough GPS trajectories, its highest achievable recall is lower compared to other schemes.

As we discussed in §4, our merging procedure is intended to yield a two-stage solution that achieves both high precision and high recall. Indeed, Figure 7, shows that the RoadRunner-based two-stage solutions significantly reduce the error rate with no impact on recall. For example, the error-rate frontier of the hybrid of RoadRunner and BE completely surpasses the frontier of BE scheme alone in each of the four cities.

These results show that when merging the RoadRunner scheme with an existing map inference algorithm, the merging procedure effectively replaces the high error-rate regions of the output of the inference algorithm with RoadRunner's accurate map. Meanwhile, the roads missed by RoadRunner but found by the map inference algorithm are mostly retained in the final inferred map.

Comparison with the best parameter settings. We take the parameter setting that can produce the best average F_1 score for each scheme as their best parameter setting. We compare our two-stage schemes against BE and Kharita with the best parameter settings. As shown in Figure 7, the two-stage scheme significantly reduces the error rates by 5.2 points(33.6%) versus BE, and 24.3 points (60.7%) versus Kharita on average over the four cities, with a slight improvement in recall.

Note that recall here is limited because the ground truth OpenStreetMap road network includes alleys, service roads, and other minor roads that are not covered by our GPS trajectory dataset.

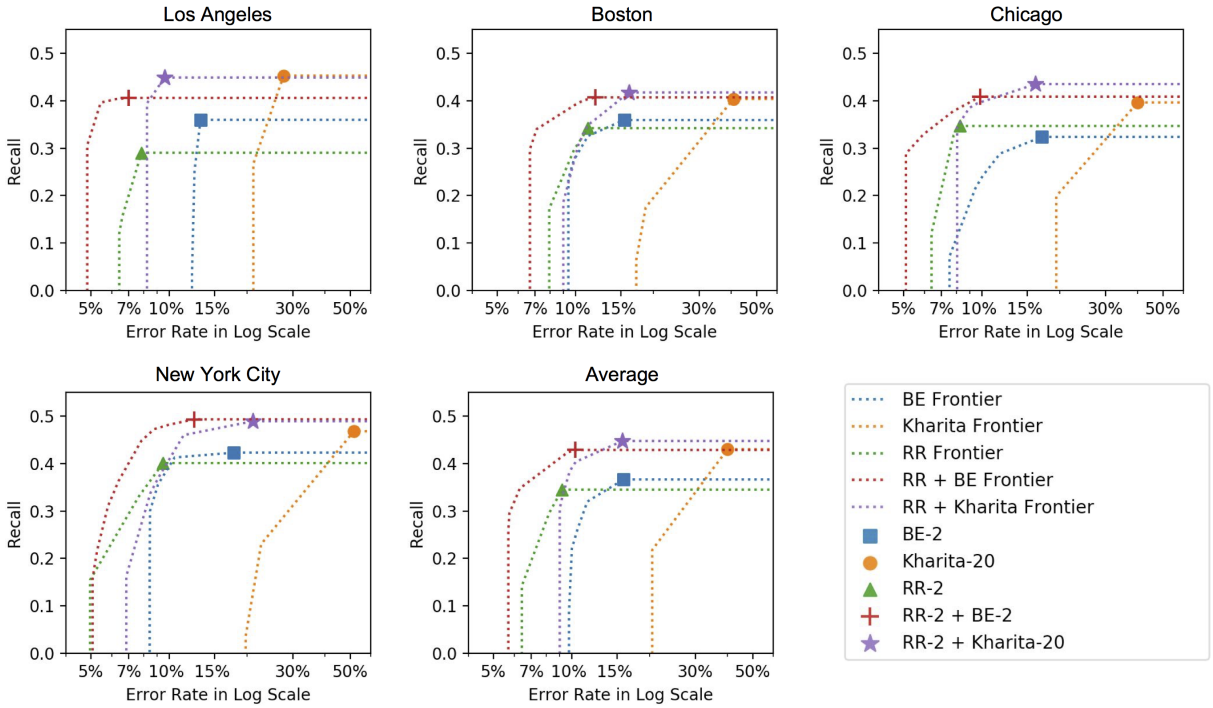

Figure 7: The error-rate frontier of RR, BE, Kharita schemes and the two-stage schemes, RR plus BE and RR plus Kharita. The markers show the point on the frontier corresponding to the specific parameters that produce the best average F_1 scores.

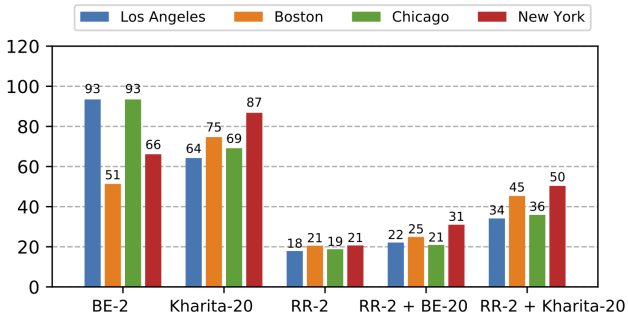

Figure 8: The median of the minimal Fréchet Distances in the shortest-path metric. The unit of the y-axis is in meter.

5.5 Usability in Navigation Scenario

We evaluate the navigation usability of the inferred maps from RoadRunner, BE, Kharita and the RoadRunner based two-stage scheme. As shown in Figure 8, our RoadRunner scheme significantly reduces the routing error by 3.9x versus BE, and 3.8x versus Kharita. The two-stage scheme also yields a significant reduction in the routing error, by 3.1x versus BE and 1.8x versus Kharita, even with a slight improvement on the map coverage (from TOPO metric).

These results show that RoadRunner captures the major road network very accurately. The two-stage scheme inherits this major road network from RoadRunner, yielding higher confidence routes in navigation scenario. The results also imply the importance of getting road network correctly in challenging areas; this areas often have a large volume of traffic and play an important role in a city's road network.

5.6 Integration with computer vision-based solution

We also study the impact of RoadRunner when combined with a computer-vision-based map inference solution that uses satellite imagery to infer the location of roads. Specifically, we compare the overall quality of the output map from RoadTracer[5] with a RoadRunner-enhanced two-stage solution. Because RoadTracer is also an incremental algorithm, in the two-stage solution, instead of merging two maps into one, we first use RoadRunner to generate a high precision map; then we use this map as the initial input to RoadTracer, which fills in missing roads in the RoadRunner output based on satellite imagery.

We use F_1 score of TOPO metric to quantify the overall quality of the output maps. We show the comparison results in Table 3. For the overall F_1 score, our two-stage solution achieves an improvement of 16.43% on average over the four cities against the satellite imagery alone solution.

We show the generated map from our two-stage solution in Figure 9. We find this combination of RoadRunner and imagery based solutions enables us to produce maps with much higher quality than prior solutions: the GPS based solution RoadRunner accurately captures the road structures in challenge areas where existing computer-vision based solutions fail, on the other hand, the imagery based solutions fill up the missing roads in areas where GPS data is very sparse or not available. We envision that this combination is the right way to take toward fully automated map generation systems.

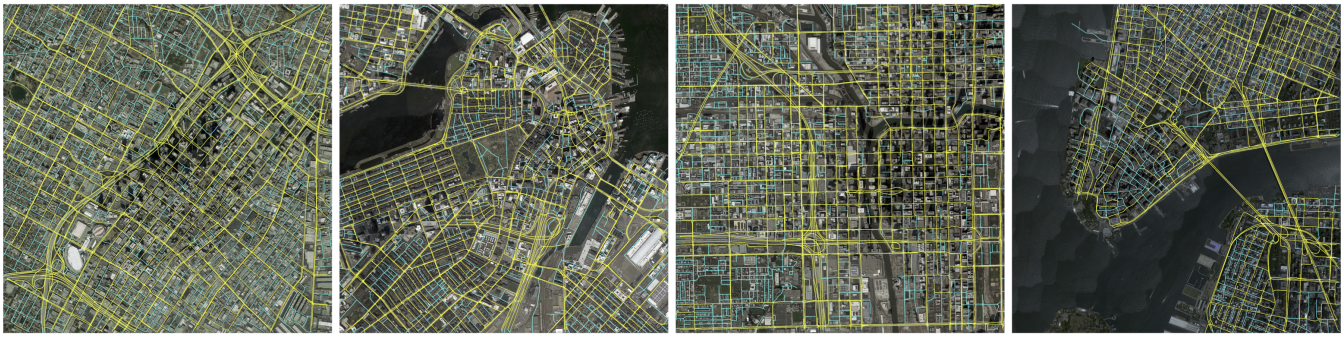

Figure 9: The generated map of the two-stage scheme with RoadRunner RR-2 (yellow) and RoadTracer (cyan).

City	RoadTracer	RR-2 + RoadTracer	Gain
Los Angeles	0.634	0.673	6.17%
Boston	0.540	0.608	12.68%
Chicago	0.585	0.653	11.56%
New York City	0.497	0.673	35.33%

Table 3: F_1 score of RoadTracer and RR-2 + RoadTracer

5.7 Runtime of RoadRunner

RoadRunner involves some computationally expensive operations, but the running time of RoadRunner is practical. In our evaluation, the CPU time (equivalent single-core running time) of our implementation is close to the KDE-based solution [6]. For example, in Los Angeles, RoadRunner takes 110 minutes to process 2.5 million GPS points in a 16 km^2 area on an AWS c5.9xlarge instance with 15% CPU utilization.

6 CONCLUSION

In this paper, we proposed a two-stage map inference architecture that enables us to generate high precision road network graphs without sacrificing coverage. As the core of this architecture, we presented RoadRunner, an automatic road network inference method that leverages the long-term structure of GPS trajectories to generate accurate maps in the dense urban areas and complex intersections where existing methods fail. We evaluated RoadRunner together with two state-of-the-art GPS-based methods and a recent computer vision-based solution on 64 km^2 from four U.S. cities. Compared with the existing map-making methods, our RoadRunner based two-stage scheme yields an improvement in the overall quality of the inferred map by 15.29%[1] on average in TOPO $F_1 score$. This quality improvement represents up to a 60.7% error rate reduction of road segments and up to a 3.1x reduction of routing errors in navigation scenario, with a small increase in recall.

Insufficient precision of current automated map-making solutions is perhaps the biggest obstacle that prevents them from real-world deployment. We believe that our approach, which significantly improves the precision of automatically generated maps while not reducing recall, marks an important step towards automating the process of road map generation.

[1] 12.17% for RR+BE, 17.28% for RR+Kharita, 16.43% for RR+RoadRunner

REFERENCES

[1] G. Agamennoni, J. I. Nieto, and E. M. Nebot. Robust inference of principal road paths for intelligent transportation systems. *IEEE T-ITS*, 12(1):298–308, 2011.
[2] M. Ahmed, S. Karagiorgou, D. Pfoser, and C. Wenk. A comparison and evaluation of map construction algorithms using vehicle tracking data. *GeoInformatica*, 19(3):601–632, 2015.
[3] M. Ahmed and C. Wenk. Constructing street networks from GPS trajectories. In *ESA*, 2012.
[4] H. Alt and M. Godau. Computing the fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, 5(01n02):75–91, 1995.
[5] F. Bastani, S. He, S. Abbar, M. Alizadeh, H. Balakrishnan, S. Chawla, S. Madden, and D. DeWitt. Roadtracer: Automatic extraction of road networks from aerial images.
[6] J. Biagioni and J. Eriksson. Map inference in the face of noise and disparity. In *ACM SIGSPATIAL 2012*.
[7] J. Biagioni and J. Eriksson. Inferring road maps from global positioning system traces: Survey and comparative evaluation. *TRR-JTRB*, (2291), 2012.
[8] L. Cao and J. Krumm. From GPS traces to a routable road map. In *ACM SIGSPATIAL*, pages 3–12, 2009.
[9] C. Chen and Y. Cheng. Roads digital map generation with multi-track gps data. In *International Workshop on Geoscience and Remote Sensing*, 2008.
[10] C. Chen, C. Lu, Q. Huang, Q. Yang, D. Gunopulos, and L. J. Guibas. City-Scale Map Creation and Updating using GPS Collections. In *KDD*, 2016.
[11] J. J. Davies, A. R. Beresford, and A. Hopper. Scalable, distributed, real-time map generation. *IEEE Pervasive Computing*, 5(4), 2006.
[12] O. H. Dørum. Deriving double-digitized road network geometry from probe data. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, page 15. ACM, 2017.
[13] S. Edelkamp and S. Schrödl. Route planning and map inference with global positioning traces. In *Computer Science in Perspective*. 2003.
[14] M. Haklay and P. Weber. OpenStreetMap: User-generated street maps. *IEEE Pervasive Computing*, 7(4):12–18, 2008.
[15] 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 *KDD*, 2012.
[16] G. Máttyus, W. Luo, and R. Urtasun. Deeproadmapper: Extracting road topology from aerial images. In *International Conference on Computer Vision*, volume 2, 2017.
[17] S. Schroedl, K. Wagstaff, S. Rogers, P. Langley, and C. Wilson. Mining gps traces for map refinement. *Data Mining and Knowledge Discovery*, 9(1):59–87, Jul 2004.
[18] W. Shi, S. Shen, and Y. Liu. Automatic generation of road network map from massive gps, vehicle trajectories. In *IEEE ITSC 2009*, 2009.
[19] R. Stanojevic, S. Abbar, S. Thirumuruganathan, S. Chawla, F. Filali, and A. Aleimat. Robust road map inference through network alignment of trajectories. In *Proceedings of the 2018 SIAM International Conference on Data Mining*. SIAM, 2018.
[20] J. D. Wegner, J. A. Montoya-Zegarra, and K. Schindler. A higher-order CRF model for road network extraction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1698–1705, 2013.
[21] K. Zheng and D. Zhu. A novel clustering algorithm of extracting road network from low-frequency floating car data. *Cluster Computing*, pages 1–10, 2018.