Learning to Generate Maps from Trajectories

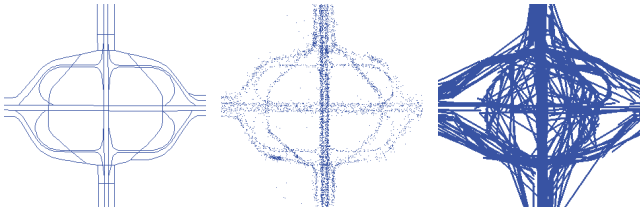
Sijie Ruan,[1,2,3] **Cheng Long,**[4] **Jie Bao,**[2,3] **Chunyang Li,**[1] **Zisheng Yu,**[1]
Ruiyuan Li,[1,2,3] **Yuxuan Liang,**[5] **Tianfu He,**[6,2,3] **Yu Zheng**[1,2,3]
[1]School of Computer Science and Technology, Xidian University, Xi'an, China
[2]JD Intelligent Cities Research, Beijing, China, [3]JD Intelligent Cities Business Unit, JD Digits, Beijing, China
[4]School of Computer Science and Engineering, Nanyang Technological University, Singapore
[5]School of Computing, National University of Singapore, Singapore
[6]School of Computer Science and Technology, Harbin Institute of Technology, Harbin, China
{sjruan, yuzisheng}@stu.xidian.edu.cn, c.long@ntu.edu.sg, {baojie, ruiyuan.li}@jd.com
{lichunyang_1, yuxliang, Tianfu.D.He, msyuzheng}@outlook.com

Abstract

Accurate and updated road network data is vital in many urban applications, such as car-sharing, and logistics. The traditional approach to identifying the road network, i.e., field survey, requires a significant amount of time and effort. With the wide usage of GPS embedded devices, a huge amount of trajectory data has been generated by different types of mobile objects, which provides a new opportunity to extract the underlying road network. However, the existing trajectory-based map recovery approaches require many empirical parameters and do not utilize the prior knowledge in existing maps, which over-simplifies or over-complicates the reconstructed road network. To this end, we propose a deep learning-based map generation framework, i.e., DeepMG, which learns the structure of the existing road network to overcome the noisy GPS positions. More specifically, DeepMG extracts features from trajectories in both spatial view and transition view and uses a convolutional deep neural network T2RNet to infer road centerlines. After that, a trajectory-based post-processing algorithm is proposed to refine the topological connectivity of the recovered map. Extensive experiments on two real-world trajectory datasets confirm that DeepMG significantly outperforms the state-of-the-art methods.

Introduction

With the increased population mobility and the rising of car-sharing service, accurate and updated road network data becomes vital. Conventional approaches for collecting road network data such as field surveys are costly and labor-intensive. With the wide usage of GPS embedded devices, a huge amount of trajectory data has been generated by different types of mobile objects (Zheng et al. 2014), which provides a new opportunity to extract the underlying road network. Yet, generating a map from trajectories is a non-trivial task, which is mainly because trajectories usually involve GPS noises and are collected with different sampling rates. To illustrate this, we present in Fig 1 a) the map near a roundabout area in Beijing, b) the GPS point cloud based on a taxi trajectory dataset and c) the line segments between two consecutive GPS points of the dataset. As could be noticed, using the point cloud for map generation would be incapable of distinguishing parallel roads that are spatially close and using the line segments for map generation would result in many redundant edges. And such a case would be even worse if the sampling rate decreases.



(a) GT. (b) Point. (c) Line.

Figure 1: Motivation Examples.

Despite the challenges of map generation, many methods have been developed for this problem, which can be categorized into three classes: 1) Clustering-based approach (Edelkamp and Schrödl 2003; Chen et al. 2016; Stanojevic et al. 2018), which first generates a set of road ends using some clustering algorithm based on geospatial distances and direction similarities and then links the road ends as road segments by using trajectories; 2) Trace-merging based approach (Cao and Krumm 2009), which scans trajectories sequentially and for each one, either merges it to some existing road segments or generates some new road segments if no suitable merging operations are possible; 3) Kernel density estimation (KDE) based approach (Biagioni and Eriksson 2012b; Wang, Wang, and Li 2015), which performs KDE on the point cloud and extracts a map afterward. While these existing methods provide some insights for map generation, they still suffer from various types of issues. Consider the clustering-based and trace-merging based approaches. Since they generate road segments somehow based on the line segments between consecutive points, in cases that trajectories are sampled with low rates, they would generate many shortcuts that do not exist in a real map. Besides, these approaches are ineffective in distinguishing parallel road segments towards the

same direction, which is mainly because they provide not many mechanisms for handling GPS noises in trajectory data. Consider the KDE based approach. While it has the capability of handling low-sampling trajectories (since it is based on point clouds), it would often merge several parallel roads that are close to one another into a single one.

In this paper, we propose a Deep learning-based Map Generation framework (DeepMG) to generate routable maps from trajectories, which can handle trajectories with different sampling rates and distinguish parallel road segments leveraging the knowledge of existing maps. The key contributions of this paper can be summarized as follows:

- We propose the first deep learning-based approach for generating routable maps from trajectories, which enjoys various superiorities over existing approaches.
- We design the map generation framework DeepMG, consisting of two modules: 1) *geometry translation*, which extracts features in trajectories from two views and then feeds the features into a Trajectory-Road translation model called T2RNet for predicting road centerlines (which correspond to line segments on a map). 2) *topology construction*, which extracts a graph structure from the predicted centerlines, links among those dead ends of edges for better connectivity and then refines the graph structure (map) by using the trajectory data.
- We conduct extensive experiments as well as case studies on two real-world taxi trajectory datasets. Experiments show DeepMG significantly outperforms the state-of-the-art map reconstruction algorithms.

Overview

Preliminary

Definition 1 (*Trajectory*)**.** A trajectory is a sequence of spatio-temporal points, denoted as $tr =< p_1, p_2, ..., p_n >$, where each point $p = (x, y, t)$ consists of a location (x, y) (e.g., longitude and latitude) at time t. Points in a trajectory are organized chronologically, namely, $\forall i < j : p_i.t < p_j.t$.

Definition 2 (*Map*)**.** A map is a directed graph, denoted as $G =< V, E >$, where each vertex $v \in V$ is associated with a location (x, y), and each edge $e \in E$ is a vertex tuple (u, v), denoting the connectivity from u to v.

Definition 3 (*Map Matching*)**.** Map matching is a process of inferring the underlying sequence of edges $e_1 \rightarrow e_2 \rightarrow \cdots \rightarrow e_n$ for a given trajectory tr based on a certain map.

Problem Statement. Given a set of trajectories $\mathcal{T} = \{tr_1, tr_2, \cdots, tr_n\}$, infer its underlying map G.

Framework

The overview of our map generation framework DeepMG is presented in Fig 2, which is comprised of two steps: 1) *Geometry Translation*, which extracts features from trajectory data and then feeds the features to T2RNet for predicting the road centerlines; 2) *Topology Construction*, which extracts a graph structure from the predicted road centerlines, generates some extra links for better connectivity and then refines the graph structure using the trajectory data again.

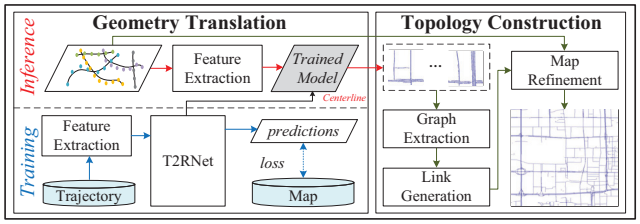


Figure 2: The Framework of DeepMG.

Geometry Translation

Feature Extraction

Similar to some tasks for aerial image segmentation (Demir et al. 2018), we split the region of interest into tiles and regard each tile as a data sample. For each tile, we partition it into a grid with $I \times J$ cells and for each grid cell, we extract some features from the trajectories in two views, namely the spatial view and the transition view.

Spatial View. In the spatial view, the features of a grid cell are based on the trajectory data within the cell. Specifically, we consider four types of features: 1) *Point*, which is the GPS point density of a grid cell (we use one of the three equal quantiles of point densities). The Point feature is the most straightforward indicator of the underlying roads. 2) *Line*: the number of line segments in each grid cell, which are generated from two consecutive points in trajectories (we use normalized values in the range [0,1]). The Line feature could help with recovering the roads when points are sparse. 3) *Speed*, which is the average moving speed in a grid cell. The speeds are inferred based on consecutive points, which could help with recovering a road segment completely since, on the same road, the speed usually does not change that much. 4) *Direction*, which captures the moving directions within a grid cell. We consider 8 regular directions (e.g., north, northeast, etc.), count the occurrences of each direction for a movement between two consecutive points, and then normalize the counts into a histogram. The direction feature could be helpful for distinguishing two parallel roads in opposite directions. In each grid cell, we can obtain an 11-dimensional vector with the above features, and the spatial view of the grid could be denoted as $\mathbf{X}_s \in \mathbb{R}^{11 \times I \times J}$.

Transition View. In the transition view, the features of a grid cell are based on the trajectory data that spans over this cell and some other cells. The features in this view are essential especially for the trajectories with low sampling rates since in these cases, the two consecutive points would be often located in different cells, which further implies that many features in the spatial view such as Line, Speed and Direction would have limited usage. Specifically, for a grid cell c, we consider those neighboring cells c' such that there exist two consecutive points which are from c' to c. We capture these cells c' with a binary matrix such that the entry corresponding to c' is set to 1 if there exist two consecutive points which are from c' to c, and 0 otherwise. Besides, we consider those cells c'' such that there exist two consecutive points which are from c to c'' and capture the cells with another binary matrix similarly as we do for cells c'. These two matrices

then correspond to the features in the transition view. Note that these two matrices would usually be very sparse, which could degrade the effectiveness of using these features. To mitigate this issue, we control the neighborhood area by a distance parameter, and use a lower resolution for forming cells c' and c''. We denote the neighborhood size of c as $T \times T$, and the features in the transition view could be denoted as $\mathbf{X}_t \in \mathbb{Z}_2^{2\times T\times T\times I\times J}$, where $\mathbb{Z}_2$ means the set of binary values $\{0, 1\}$.

T2RNet

Recall that the task of the geometry translation module is to predict the road centerlines of a map from trajectory data. An intuitive idea is to model this task as a pixel-wise image classification one and then utilize existing models for the task. However, due to the fact that a typical road centerline would be single-pixel wide, which is too shallow for conventional image classification models to be effective. Motivated by this, we introduce an auxiliary task to support the task of road centerline prediction. Specifically, the auxiliary task is to predict the spatial regions that involve roads, which we call *road regions*. We call this auxiliary task *road region prediction*. Note that the task of road region prediction should be easier than that of road centerline prediction since for the former task, the targets are wider and existing image classification models could be effectively utilized. Specifically, we propose a multi-task learning architecture called *T2RNet* whose overview is presented in Fig 3. T2RNet involves four components, namely (1) *transition embedding*, (2) *shared encoder*, (3) *road region decoder*, and (4) *road centerline decoder*, with details explained next.

(1) Transition Embedding. This component is motivated by that the features in the transition view, captured by $\mathbf{X}_t$, could be very sparse since objects usually move from and to a limited number of cells nearby. Inspired by the idea of word embedding (Mikolov et al. 2013), we first use dense layers to transform $\mathbf{X}_t$ into a dense representation $\mathbf{H}_t \in \mathbb{R}^{k\times I\times J}$ and the structure is depicted in the top left of Fig 3, where k is the size of the embedded dimension.

(2) Shared Encoder. The shared encoder, which is shared by two decoders, is used to encode the features at different levels. Specifically, the concatenation of $\mathbf{X}_s$ and $\mathbf{H}_t$ is passed to M downsampling blocks. Each downsampling block contains a ConvBlock and a Max Pooling layer. The structure of the ConvBlock, as shown on the top right of Fig 3, contains two 3×3 convolutional layers, and there are F filters in the first ConvBlock, which are doubled after each downsampling. In the ConvBlock, each convolutional layer is followed by a Batch Normalization (Ioffe and Szegedy 2015), with a ReLU to introduce non-linearity. We denote the feature map after a ConvBlock as $\mathbf{E}_m \in \mathbb{R}^{2^{m-1}F\times \frac{I}{2^{m-1}}\times \frac{J}{2^{m-1}}}$, where $m = 1, \cdots, M$, and $\mathbf{E}_m$ is downsampled by half after the Max Pooling layer. Finally, another ConvBlock, which serves as the bottleneck layer, is used to produce the final feature map $\mathbf{E}_{M+1}$.

(3) Road Region Decoder. This is to predict the road regions given the encoded information from the shared encoder (i.e., the road region prediction task). Specifically,

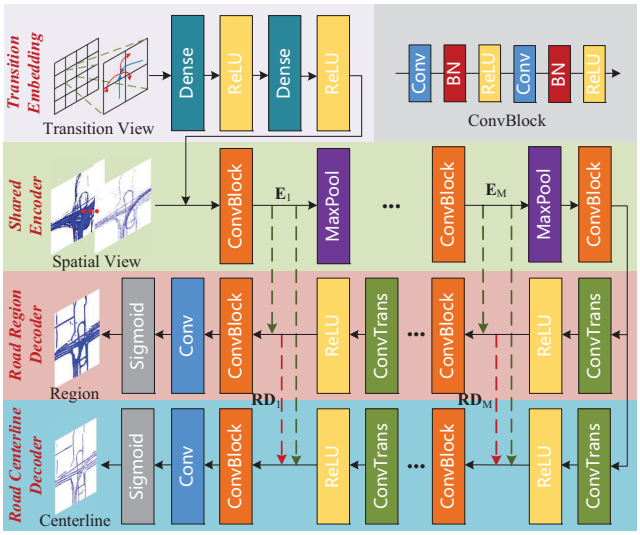


Figure 3: T2RNet Architecture.

$\mathbf{E}_{M+1}$ passes through M upsampling blocks. Each upsampling block first uses a transposed convolution layer (ConvTrans) followed by a ReLU for the upsampling purpose. Its output $\mathbf{RD}_m \in \mathbb{R}^{2^{m-1}F\times \frac{I}{2^{m-1}}\times \frac{J}{2^{m-1}}}$ is concatenated with $\mathbf{E}_m$, and then fed into a ConvBlock for information fusion. Finally, a convolution layer (3×3, 1) and a Sigmoid activation are applied to generate the final road region prediction.

(4) Road Centerline Decoder. This is to predict the road centerline leveraging both the encoded information and the information from road region decoder (i.e., the road centerline prediction task). The idea is similar to that of the road region decoder except that during the information fusion in each layer, we concatenate not only the information from the encoder, i.e., $\mathbf{E}_m$, but also that from the road region decoder, i.e., $\mathbf{RD}_m$, so that the learned road region information would help with the road centerline prediction task.

Optimization. For each edge in the map, we use a one-pixel wide line segment linking its two ends as a road centerline. The tile of road centerlines is denoted as $\mathbf{Y}_c \in \mathbb{Z}_2^{I\times J}$. We also mask nearby grids around the road centerlines to generate the labels of road region $\mathbf{Y}_r \in \mathbb{Z}_2^{I\times J}$. Since the positive and the negative labels are extremely imbalanced for $\mathbf{Y}_c$ and $\mathbf{Y}_r$, we employ the Dice loss (Milletari, Navab, and Ahmadi 2016), which is designed for tackling small foreground issues, as our loss function. Specifically, the Dice loss of the prediction $\hat{\mathbf{Y}}$ given the ground truth $\mathbf{Y}$ is:

$$\mathcal{L}_{Dice}(\hat{\mathbf{Y}}, \mathbf{Y}) = 1 - \frac{2\sum_i^I \sum_j^J \hat{Y}_{ij} Y_{ij} + \epsilon}{\sum_i^I \sum_j^J \hat{Y}_{ij} + \sum_i^I \sum_j^J Y_{ij} + \epsilon} \quad (1)$$

where the ϵ term is used to ensure the loss function stability by avoiding the numerical issue of being divided by 0.

We minimize the hybrid Dice loss of road centerline prediction and road region prediction by gradient descent.

$$\mathcal{L}(\boldsymbol{\theta}) = (1-\lambda)\mathcal{L}_{Dice}(\hat{\mathbf{Y}}_c, \mathbf{Y}_c) + \lambda\mathcal{L}_{Dice}(\hat{\mathbf{Y}}_r, \mathbf{Y}_r) \quad (2)$$

where $\boldsymbol{\theta}$ are all trainable parameters and λ is the hyperparameter that balances two types of loss.

Topology Construction

While the geometry translation module can predict the road centerlines, the topological connectivity can not be guaranteed. Many techniques for connecting broken edges have been proposed in the literature of aerial imagery road detection (Mnih 2013; Máttyus, Luo, and Urtasun 2017; Sun et al. 2018). Nevertheless, these techniques are incapable of inferring the directions of roads or guaranteeing the connection between two edges that are truly linked. Differing from these studies, we use trajectories as evidence for constructing the topology of a map, which links broken road segments and infers the directions simultaneously. Specifically, our solution involves three steps: namely 1) *graph extraction*, which extracts an initial map from the predicted road centerlines; 2) *link generation*, which generates all possible transition links among dead ends of edges; 3) *map refinement*, which refines the map further based on trajectories.

Graph Extraction

In this step, we concatenate all predicted road centerlines based on their underlying tiles. Then we adopt the combustion technique (Shi, Shen, and Liu 2009) to construct an initial undirected map from the predicted centerlines, where each road centerline would have a corresponding edge.

Link Generation

In this step, we use a heuristic algorithm to generate all possible links among dead ends of edges nearby. The pseudo code is given in Algorithm 1. Since a road edge is straight, the generated links should also be smooth when linking broken edges. Therefore, we generate a link from each dead end v of an edge within a radius R_{link} in the following two cases.

- The extension of the edge starting from v intersects another edge. In this case, we create a new vertex at the intersection t, and generate a link between v and t (Line 9).
- The intersection point does not exist, but the nearest vertex n of the edge to v is a dead end, and there could be a smooth transition from v to n (e.g., the turning direction is not greater than 90°). In this case, we generate a link between v and n (Line 13).

In detail, we first create two sets V_{new} and E_{new} to record vertices and links that will be added to the map (Line 1). After all dead ends are examined, each vertex in V_{new} is added and used to split an existing edge on which the vertex is located on and each link in E_{new} is added (Line 14-17).

As shown in Fig 4a, we generate a link between o and a on e_1 and another link between o and b of e_2 (the turning from o to e_2 is smooth). Note that we do not generate a link from o to e_3 since its transition to e_3 is not smooth.

Map Refinement

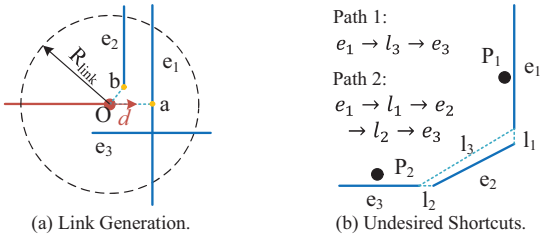
In this step, we refine the map further based on trajectories. The main idea is to remove edges and generated links that are not traversed through often. Specifically, we perform trajectory map matching (Yuan et al. 2010b) to map the trajectories to the current map. We note that directly applying existing methods is not sufficient for the map refinement purpose here since they usually assume that moving objects follow the shortest paths between consecutive points, which would cause problems as illustrated in Fig 4b. In the figures, the solid lines are predicted edges, and the dash lines are generated links. During map matching, if e_1 is a candidate road to match for P_1 and e_3 is a candidate road for P_2, then the algorithm uses *Path 1* to capture the transition probability from e_1 to e_2 since this corresponds to the shortest path. However, *Path 2* is much more likely an actual path, since the edges are more reliable than links. Therefore, we use the edge length multiplied by a penalty factor α ($\alpha > 1$) as the edge weight for created links so that the shortest path calculation will prefer edges over generated links.

After we have matched all trajectories on the map, the inferred paths are used as evidence of the transition. We remove those edges or generated links who witness less than S times of transitions, and obtain the refined directed map.

Algorithm 1 Link Generation.

Input: Initial undirected map G; link radius R_{link}.
Output: Linked undirected map G.

1: $V_{new} \leftarrow \emptyset$, $E_{new} \leftarrow \emptyset$;
2: **for** $v \in G.V$ **do**
3: **if** $deg(v) = 1$ **then**
4: $u \leftarrow get\_adjacent\_vertex(v)$;
5: $C \leftarrow get\_neighboring\_edges(v, R_{link})$;
6: **for** $e \in C$ **do**
7: $t \leftarrow cal\_intersection\_point(u, v, e)$;
8: **if** $t \neq null$ **then**
9: $V_{new} \leftarrow V_{new} \cup \{t\}$, $E_{new} \leftarrow E_{new} \cup \{(v, t)\}$;
10: **else**
11: $n \leftarrow$ the nearest vertex of e to v;
12: **if** $deg(n) = 1$ **and** $angle(\overrightarrow{uv}, \overrightarrow{vn}) \leq 90°$ **then**
13: $E_{new} \leftarrow E_{new} \cup \{(v, n)\}$;
14: **for** $v \in V_{new}$ **do**
15: split the edge, on which v is located; add v to G;
16: **for** $e \in E_{new}$ **do**
17: add e to G;
18: **return** G;



(a) Link Generation. (b) Undesired Shortcuts.

Figure 4: Topology Construction Illustration.

Experiments

In this section, we report the overall performance of DeepMG, following by the performance of its two modules.

Experimental Settings

Datasets. We use two real-world taxi trajectory datasets, i.e., TaxiBJ and TaxiJN, with different sampling rates for evaluations. TaxiBJ is obtained from T-Drive (Yuan et al. 2010a) and TaxiJN is a private one donated by the government of Jinan. Several preprocessing algorithms, e.g., noise filtering, stay point removal (Zheng 2015), are applied. The detailed statistics of the datasets are given in Tab 1, where the statistics of the test region are given in the parentheses. Each sample is a 256×256 grid, and each cell is 2m×2m, which is the coarsest resolution so that two parallel centerlines are distinguishable. The number of samples for training, validation, and test are 744, 180, 100 for TaxiBJ, and 1251, 313, 100 for TaxiJN. The baselines are only compared in the test region. We use OpenStreetMap as the ground truth, and road region labels are generated by masking 2 cells surrounding the centerlines. We remove edges on which there are less than 10 trajectories in TaxiBJ, and 5 in TaxiJN, respectively.

Table 1: Data Descriptions.

Dataset	TaxiBJ	TaxiJN
#Days	30	30
#Vehicles	500	70
Sampling Rate	∼30s	∼3s
Size (km^2)	16×16 (5×5)	16×26 (5×5)
#Points	3.1M (304K)	5.7M (322K)
#Trajectories	66,124 (13,462)	29,556 (3,954)
Roads (km)	2,772 (284)	2,048 (123)

Metrics. The accuracy of a generated map depends on geometry and topology. We use the de-facto standard for measuring the quality of the map proposed in (Biagioni and Eriksson 2012a) to simultaneously measure the geometric and topological similarity of maps. The main idea is that starting from a random location on the road, we find its road network reachable grid cells within a maximum radius. The reachable cells are marked as 1, and 0 otherwise. We report the average F1 score in different spatial resolutions over n sampled starting locations. We vary the cell size from $5m$ to $20m$, sample $n = 200$ random starting cells and set the reachable radius as $2km$, consistent with existing approaches (Stanojevic et al. 2018).

Baselines. We compare DeepMG with 5 representative approaches for map reconstruction listed as follows.

- **Edelkamp** (Edelkamp and Schrödl 2003), which is a clustering-based algorithm. It first generates seed locations as initial centers according to distance and bearing similarity, and then k-Means algorithm is used to adjust cluster centers. At last, trajectories are used to form road segments from those locations.
- **Chen** (Chen et al. 2016), which is another clustering-based algorithm. It first generates local small edges based on centroids, then links small edges using trajectories.
- **Kharita** (Stanojevic et al. 2018), which can also be categorized as a clustering-based method. After vertices are inferred by clustering and edges are created by trajectories, a graph sparsification step is used to simplify the graph. To the authors' knowledge, this is the state-of-the-art method for map reconstruction.
- **Cao** (Cao and Krumm 2009), which is a trace-merging based algorithm. The main idea is to reduce GPS noises based on particle simulation. It uses a strong but short-range force to pull together nearby traces, and a weaker but long-range force to keep traces from straying too far.
- **Biagioni** (Biagioni and Eriksson 2012b), which is a KDE based approach. It converts the trajectory points into a road network skeleton image in different confidence levels based on KDE, and then a density-aware map matching algorithm is used to remove low support edges.

We also compare the performance of DeepMG without the topology construction, denoted as DeepMG-nt.

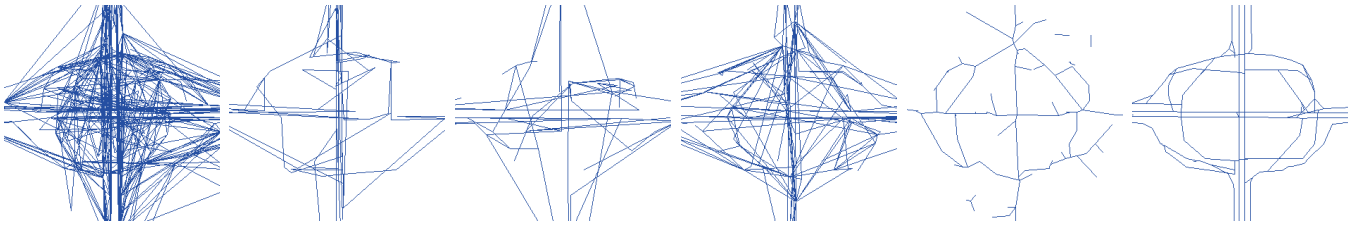
Implementations. DeepMG is fully written in Python. We use the author's implementation of the baseline algorithms, except for Chen, whose source code is not available for us. The T2RNet model in geometry translation is implemented by PyTorch, and trained with one NVIDIA Tesla V100 GPU.

Parameter Settings. For feature extraction in transition view, we set $T = 8$, and the neighborhood distance as 300m for TaxiBJ and 50m for TaxiJN due to different sampling rates. T2RNet has 64 and 8 hidden units respectively in the first and the second dense layer in transition embedding, the first ConvBlock of the encoder has $F = 64$ filters, and the network performs downsampling for $M = 4$ times. During the training phase, we leverage Adam (Kingma and Ba 2014) to perform network training with a learning rate $2e-4$ and batch size 8. We also apply a staircase-like schedule by halving the learning rate every 10 epochs. For topology construction, $R_{link} = 100m$, $\alpha = 1.4$, $S = 5$ for TaxiBJ and $S = 2$ for TaxiJN due to different number of trajectories.

Performance Comparison

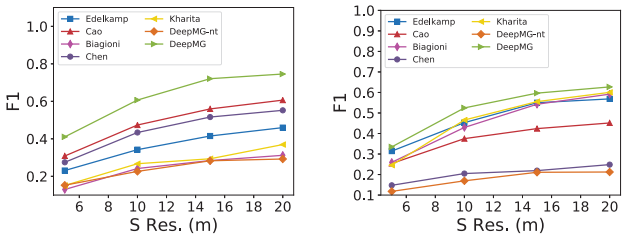
Quantitative Comparison. We report the F1 score for 5 baseline solutions, DeepMG and its variant in Fig 6. The results show DeepMG consistently outperforms the baselines in different spatial resolutions on two trajectory datasets. In the finest resolution, DeepMG outperforms the best baseline algorithm by 32.3%, 6.5% on TaxiBJ and TaxiJN, respectively. The improvement on TaxiBJ is much larger due to the low sampling rate issue, which is not well handled by existing methods. It also can be observed by comparing DeepMG-nt and DeepMG. Before topology construction, the connectivity of the map is poor, which leads to limited reachable grids and lowers the F1 score. After we construct the topology of the map, the performance of the DeepMG ranked the top over all baselines, which not only shows the importance of the second step, but also demonstrates the effectiveness of the first step.

Visual Comparison. Due to space limitations, we only visualize DeepMG, as well as baselines at a roundabout in



(a) Edelkamp. (b) Chen. (c) Kharita. (d) Cao. (e) Biagioni. (f) DeepMG.

Figure 5: Seven algorithm at a roundabout near Gongzhufen area in TaxiBJ.



(a) TaxiBJ. (b) TaxiJN.

Figure 6: Performance on F1.

Beijing in Fig 5 [1] , which is one of the most difficult cases for map reconstruction. The corresponding trajectories and the ground truth map is already shown in Fig 1. The results demonstrate that Biagioni, which is a KDE-based method, generates few redundant edges, and is robust to the sampling rate. However, it is difficult to distinguish two parallel roads. Although the direction is considered in other baselines, it is difficult to set global consistent parameters, which makes the edges too complicated. As shown in the figure, DeepMG significantly outperforms baselines, which can successfully identify parallel roads in the same direction without generating many redundant edges.

Geometry Translation Component

Model Comparison. To evaluate the effectiveness of T2RNet, we replace the encoder-decoder part of T2RNet with 6 models to directly predict road centerlines.

- FCN (Long, Shelhamer, and Darrell 2015): FCN replaces the FC layers of classification nets with convolutions, and multi-resolution features are combined before prediction.
- LinkNet (Chaurasia and Culurciello 2017): LinkNet uses ResNet (He et al. 2016) as the encoder block, and the feature maps from the encoder are summed with the upsampled feature maps from the decoder.
- DeepLabV3+ (Chen et al. 2017): DeepLabV3+ encodes multi-scale contextual information by applying atrous convolution at multiple scales, and then decodes the segmentation results along object boundaries.
- UNet (Ronneberger, Fischer, and Brox 2015): UNet consists of an encoding block which downsamples the original image, and a decoding block, which upsamples the encoded information. The encoding block and decoding block have linked shortcuts at corresponding layers.
- D-LinkNet (Zhou, Zhang, and Wu 2018): D-LinkNet is a variant of LinkNet, which introduces a bottleneck layer with dilated convolution between encoder and decoder.
- LinkNet+1D Decoder (Sun et al. 2019): It is also a variant of LinkNet, which replaces the traditional 2D filters of convolution layers in the decoder block with 1D. It is the SOTA method for aerial image segmentation.

We report the average pixel-wise Precision, Recall, and F1 of the test set in Tab 2. The performance of FCN is the worst in both two datasets since it predicts results by directly upsampling from a very small feature map, which makes the location information hard to reconstruct. DeepLabV3+ also doesn't perform very well, due to there is few information fusion between encoder and decoder. Other models following the encoder-decoder structure with skip-connections, shows relatively good performance. Experiments show that concatenation-based fusion (e.g., UNet) is more effective than summation-based fusion (e.g., LinkNet and its variants) for centerline inference, since the summation is usually inaccurate. T2RNet consistently outperforms all alternative models. An auxiliary task, i.e., road region inference, is introduced in T2RNet, so that the centerline decoder can fuse both the encoder and road region decoder information, and make more accurate predictions.

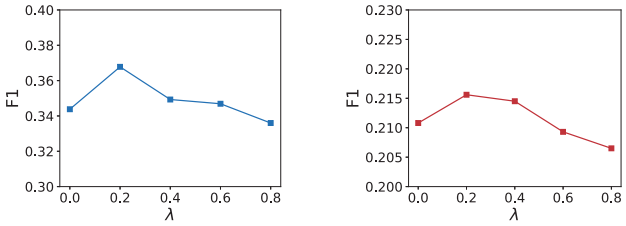
Effect of Multi-task Learning. To show the effectiveness of multi-task learning, we evaluate our model under different λ settings from 0 to 0.8. A small λ pays more attention to the centerline inference task, and a larger λ cares more about road region prediction. The result is shown in Fig 7. We find that when $\lambda = 0.2$, T2RNet achieves the best performance on both two datasets. A smaller λ or a larger λ decreases the centerline inference performance.

Effect of Different Features. The evaluation of different feature combinations is shown in Tab 3, where P, L, S, A means point feature, line feature, spatial view features, and all features, respectively. Comparing P and L, we find point feature has higher precision than line feature in both datasets, which is consistent with our common knowledge. However, line feature has a higher recall than point feature since it fills the distance gap between consecutive points. We also find that line feature ultimately has a higher F1 score than point feature in TaxiJN, while TaxiBJ is on the opposite.

[1] The final output of DeepMG is also shown in Fig 2.

Table 2: Model Comparisons on Different Datasets

Methods	#Params	TaxiBJ			TaxiJN		
		Precision	Recall	F1	Precision	Recall	F1
FCN	134.3M	0.1824	**0.6269**	0.2778	0.0255	**0.5026**	0.0482
LinkNet	21.7M	0.2652	0.4690	0.3325	0.1566	0.3238	0.2059
DeepLabV3+	50.5M	0.2199	0.5571	0.3107	0.1383	0.3471	0.1883
UNet	39.4M	0.2745	0.4760	0.3439	0.1749	0.2945	0.2079
D-LinkNet	31.1M	0.2637	0.4884	0.3387	0.1654	0.3116	0.2084
LinkNet+1D Decoder	21.8M	0.2576	0.5105	0.3384	0.1602	0.3248	0.2042
T2RNet	63.0M	**0.2879**	0.5245	**0.3678**	**0.1795**	0.3020	**0.2156**



(a) TaxiBJ. (b) TaxiJN.

Figure 7: Effect of λ.

The sampling rate of TaxiJN is much higher than TaxiBJ, so the line between two consecutive points is highly likely still on the road while this assumption is not held in TaxiBJ. Comparing S and A, we demonstrate the effectiveness of the transition view. The improvement of the transition view is much bigger in TaxiBJ than TaxiJN. This could because features in spatial view are already accurate enough when the sampling rate is high, while the transition view brings more information gains for low sampling rate trajectories.

Table 3: Effect of Different Features.

F	TaxiBJ			TaxiJN		
	Preci.	Recall	F1	Preci.	Recall	F1
P	0.2549	0.4840	0.3305	0.1619	0.2966	0.2030
L	0.2293	0.5000	0.3097	0.1580	**0.3364**	0.2096
S	0.2594	0.5196	0.3434	0.1753	0.2893	0.2123
A	**0.2879**	**0.5245**	**0.3678**	**0.1795**	0.3020	**0.2156**

Topology Construction Component

We have already quantitatively evaluated the effectiveness of topology construction in Fig 6. In this subsection, we further demonstrate a qualitative result. The generated maps under different α settings are given the Fig 8. The blue lines are the predicted edges, and the red lines are the links we generated. On one hand, if we do not add a penalty to the generated link ($\alpha = 1.0$), there are many shortcuts generated. On the other hand, if α is too large (e.g., $\alpha = 3.0$), road segments can be broken due to the map matching algorithm has fewer chances to match trajectories on those links. Therefore, we choose $\alpha = 1.4$, which has a good balance.

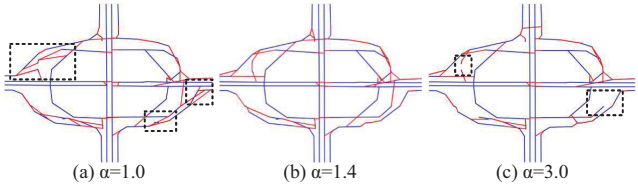


Figure 8: Effect of α.

Related Work

Map Reconstruction from Trajectories

Map reconstruction from trajectories is a field that has been extensively studied for a long time, and the main research focus is on how to reduce GPS noises and uncertainties. A survey is given in (Biagioni and Eriksson 2012a), which categorized existing works into 3 types: 1) Clustering-based method (Edelkamp and Schrödl 2003; Chen et al. 2016; Stanojevic et al. 2018) , which firstly identifies nodes (or short edges) from raw GPS points using spatial clustering algorithms based on location closeness and direction similarity, and then links those nodes (or short edges) using trajectories; 2) trace-merging based method (Cao and Krumm 2009; Niehöfer et al. 2009), which directly merges edges from every consecutive points in trajectories; 3) KDE based method (Biagioni and Eriksson 2012b; Wang, Wang, and Li 2015) , which performs kernel density estimation over raw GPS points, and several image processing techniques (e.g., morphological dilation, closing, thinning) are applied to extract road centerlines. However, all existing methods either cannot handle low sampling rate trajectories or are not able to identify spatially near parallel roads. In this work, we use a deep learning-based approach to learn how to reduce noise and detect centerlines, and the connectivity is further refined using trajectories.

Road Detection from Aerial Images

Road detection from aerial images is an active field in the computer vision community, which essentially corresponds to a semantic segmentation problem. With the rising of deep learning, (Mnih and Hinton 2012; Mnih 2013) employ convolutional networks to classify each pixel from a larger image patch. Due to the performance issues and complicated dependencies with neighborhood locations, predicting all pixels at once becomes more popular recently, e.g.,

Fully Convolutional Network (Long, Shelhamer, and Darrell 2015), U-Net (Ronneberger, Fischer, and Brox 2015), and DeepLab (Chen et al. 2017). CasNet (Cheng et al. 2017) uses a cascaded network to extract road region and road centerline from aerial images. D-LinkNet (Zhou, Zhang, and Wu 2018) won top places in recent aerial image segmentation challenge (Demir et al. 2018). The closest work to us is (Sun et al. 2019), which leverages both satellite images and GPS trajectories to detect road regions. Instead of treating trajectory only from spatial view in (Sun et al. 2019), we introduce another transition view to improve the prediction quality. Moreover, most aforementioned works are mainly focused on detecting road regions from remote sensing data, while we focus on the road centerline inference from trajectories, and reconstructing a routable and directed map.

Conclusion

In this paper, we study the problem of generating maps from trajectories. We propose a deep learning-based map generation framework DeepMG. DeepMG can handle trajectories with different sampling rates and distinguish parallel roads that are spatially close without empirical parameter tuning. Experiments on two real-world datasets show DeepMG outperforms baselines by at least 32.3% for low sampling rate trajectories and 6.5% for high sampling rate trajectories.

Acknowledgments

This work was supported by the NSFC Grant No. 61672399, No. U1609217, and National Key R&D Program of China (2019YFB2101800). The research of Cheng Long was supported by the NTU Start-Up Grant and Singapore MOE Tier 1 Grant RG20/19 (S).

References

Biagioni, J., and Eriksson, J. 2012a. Inferring road maps from global positioning system traces: Survey and comparative evaluation. *TRANSPORT RES REC* 2291(1):61–71.

Biagioni, J., and Eriksson, J. 2012b. Map inference in the face of noise and disparity. In *SIGSPATIAL*, 79–88. ACM.

Cao, L., and Krumm, J. 2009. From gps traces to a routable road map. In *SIGSPATIAL*, 3–12. ACM.

Chaurasia, A., and Culurciello, E. 2017. Linknet: Exploiting encoder representations for efficient semantic segmentation. In *VCIP*, 1–4. IEEE.

Chen, C.; Lu, C.; Huang, Q.; Yang, Q.; Gunopulos, D.; and Guibas, L. 2016. City-scale map creation and updating using gps collections. In *SIGKDD*, 1465–1474. ACM.

Chen, L.-C.; Papandreou, G.; Kokkinos, I.; Murphy, K.; and Yuille, A. L. 2017. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *TPAMI* 40(4):834–848.

Cheng, G.; Wang, Y.; Xu, S.; Wang, H.; Xiang, S.; and Pan, C. 2017. Automatic road detection and centerline extraction via cascaded end-to-end convolutional neural network. *TGRS* 55(6):3322–3337.

Demir, I.; Koperski, K.; Lindenbaum, D.; Pang, G.; Huang, J.; Basu, S.; Hughes, F.; Tuia, D.; and Raska, R. 2018. Deepglobe 2018: A challenge to parse the earth through satellite images. In *CVPRW*, 172–17209. IEEE.

Edelkamp, S., and Schrödl, S. 2003. Route planning and map inference with global positioning traces. In *Computer science in perspective*. Springer. 128–151.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *CVPR*, 770–778.

Ioffe, S., and Szegedy, C. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

Kingma, D. P., and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Long, J.; Shelhamer, E.; and Darrell, T. 2015. Fully convolutional networks for semantic segmentation. In *CVPR*, 3431–3440.

Máttyus, G.; Luo, W.; and Urtasun, R. 2017. Deeproadmapper: Extracting road topology from aerial images. In *ICCV*, 3438–3446.

Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; and Dean, J. 2013. Distributed representations of words and phrases and their compositionality. In *NIPS*, 3111–3119.

Milletari, F.; Navab, N.; and Ahmadi, S.-A. 2016. V-net: Fully convolutional neural networks for volumetric medical image segmentation. In *3DV*, 565–571. IEEE.

Mnih, V., and Hinton, G. E. 2012. Learning to label aerial images from noisy data. In *ICML*, 567–574.

Mnih, V. 2013. *Machine learning for aerial image labeling*. Citeseer.

Niehöfer, B.; Burda, R.; Wietfeld, C.; Bauer, F.; and Lueert, O. 2009. Gps community map generation for enhanced routing methods based on trace-collection by mobile phones. In *SPACOMM*, 156–161. IEEE.

Ronneberger, O.; Fischer, P.; and Brox, T. 2015. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, 234–241. Springer.

Shi, W.; Shen, S.; and Liu, Y. 2009. Automatic generation of road network map from massive gps, vehicle trajectories. In *ITSC*, 1–6. IEEE.

Stanojevic, R.; Abbar, S.; Thirumuruganathan, S.; Chawla, S.; Filali, F.; and Aleimat, A. 2018. Robust road map inference through network alignment of trajectories. In *ICDM*, 135–143. SIAM.

Sun, T.; Chen, Z.; Yang, W.; and Wang, Y. 2018. Stacked u-nets with multi-output for road extraction. In *CVPR Workshops*, 202–206.

Sun, T.; Di, Z.; Che, P.; Liu, C.; and Wang, Y. 2019. Leveraging crowdsourced gps data for road extraction from aerial imagery. In *CVPR*, 7509–7518.

Wang, S.; Wang, Y.; and Li, Y. 2015. Efficient map reconstruction and augmentation via topological methods. In *SIGSPATIAL*, 25. ACM.

Yuan, J.; Zheng, Y.; Zhang, C.; Xie, W.; Xie, X.; Sun, G.; and Huang, Y. 2010a. T-drive: driving directions based on taxi trajectories. In *SIGSPATIAL*, 99–108. ACM.

Yuan, J.; Zheng, Y.; Zhang, C.; Xie, X.; and Sun, G.-Z. 2010b. An interactive-voting based map matching algorithm. In *MDM*, 43–52. IEEE Computer Society.

Zheng, Y.; Capra, L.; Wolfson, O.; and Yang, H. 2014. Urban computing: concepts, methodologies, and applications. *TIST* 5(3):38.

Zheng, Y. 2015. Trajectory data mining: an overview. *TIST* 6(3):29.

Zhou, L.; Zhang, C.; and Wu, M. 2018. D-linknet: Linknet with pretrained encoder and dilated convolution for high resolution satellite imagery road extraction. In *CVPR Workshops*, 182–186.