ProgRPGAN: Progressive GAN for Route Planning

Tao-yang Fu
The Pennsylvania State University
University Park, PA, 16802, USA
txf225@cse.psu.edu

Wang-Chien Lee
The Pennsylvania State University
University Park, PA, 16802, USA
wlee@cse.psu.edu

ABSTRACT

Learning to route has received significant research momentum as a new approach for the route planning problem in intelligent transportation systems. By exploring global knowledge of geographical areas and topological structures of road networks to facilitate route planning, in this work, we propose a novel Generative Adversarial Network (GAN) framework, namely *Progressive Route Planning GAN (ProgRPGAN)*, for route planning in road networks. The novelty of ProgRPGAN lies in the following aspects: 1) we propose to plan a route with levels of increasing map resolution, starting on a low-resolution grid map, gradually refining it on higher-resolution grid maps, and eventually on the road network in order to *progressively* generate various realistic paths; 2) we propose to transfer parameters of the previous-level generator and discriminator to the subsequent generator and discriminator for parameter initialization in order to improve the efficiency and stability in model learning; and 3) we propose to pre-train embeddings of grid cells in grid maps and intersections in the road network by capturing the network topology and external factors to facilitate effective model learning. Empirical result shows that ProgRPGAN soundly outperforms the state-of-the-art learning to route methods, especially for long routes, by 9.46% to 13.02% in F1-measure on multiple large-scale real-world datasets. ProgRPGAN, moreover, effectively generates various realistic routes for the same query.

CCS CONCEPTS

• **Theory of computation** → **Sequential decision making**; • **Information systems** → **Geographic information systems**; • **Computing methodologies** → *Neural networks*.

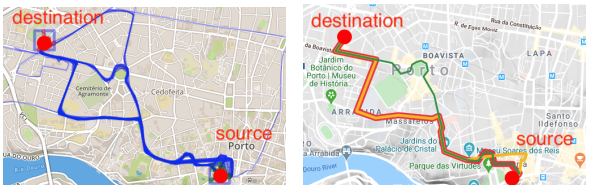
KEYWORDS

route planning, generative adversarial networks, neural networks

ACM Reference Format:
Tao-yang Fu and Wang-Chien Lee. 2021. ProgRPGAN: Progressive GAN for Route Planning. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '21), August 14–18, 2021, Virtual Event, Singapore.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3447548.3467406

KDD '21, August 14–18, 2021, Virtual Event, Singapore.

ACM ISBN 978-1-4503-8332-5/21/08…$15.00
https://doi.org/10.1145/3447548.3467406



(a) Real routes (b) Generated paths

Figure 1: Routes for the same source and destination

1 INTRODUCTION

With the proliferation of GPS-enabled devices and location-based services, enormous amount of trajectory data are generated and collected [2, 3]. The large volume of trajectory data provides opportunities to study and enhance various mining tasks in urban planning and intelligence transportation systems (ITS), such as trajectory clustering, travel time estimation, route planning, etc. Among them, *route planning* (RP) is a fundamental task in many ITS applications, e.g., navigation, ride sharing, and online map [8, 25]. Conventional studies on route planning focus on developing efficient *graph-search* algorithms [6, 11, 12, 15, 17, 26] to find top-k paths for a given *route planning query* (RPQ) based on various cost estimates associated with road segments, such as distance, travel time, gasoline consumption, traversed frequency, etc. These algorithms, assuming that mobile road users prefer paths with the optimal cost, focus on finding routes in certain aspects, e.g., shortest distance or fastest routes, based on some explicitly identified cost factors. However, the returned routes do not necessarily agree with the choices of road users (who may consider some implicit factors such as road safety or the number of traffic lights encountered) [4]. Moreover, the usefulness of the returned routes highly depends on the quality of the cost estimates adopted in RPQ.

Recently, arguing that trajectories have holistically captured various factors considered by road users, a growing number of *learning to route (LTR)* methods have been proposed to mine the routing behaviours and decisions in trajectories for route planning. Given a road network and a trajectory dataset collected from the road network, these methods learn models from the trajectories to generate a number of routes (i.e., *paths* on the road network) from the source to the destination (as specified by a given RPQ) that mobile users most likely travel on. Mostly exploring variants of Recurrent Neural Network (RNN), these LTR methods generate a path on road network by sequentially predicting the next road segment from the source to the destination [16, 27, 32]. However, without exploiting global knowledge of geographical areas and topological structures of road networks in routing decisions, they do not generate good results, especially for long paths. Moreover, while some existing works may generate multiple paths, e.g., by searching paths with top-k minimal cost or applying beam search [22] to generate the

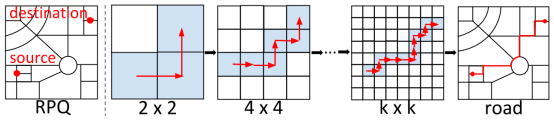


Figure 2: Progressive Route Planning

top-k most likely paths, the generated paths are often very similar, i.e., only different slightly in a small portion of road segments. Consequently, some acceptable but less frequently traveled paths are not generated. For example. Figure 1(a) shows the real routes of taxi drivers in Porto city between a source and a destination. As shown in Figure 1(b), in contrast to the real routes, some similar high-frequent paths may be generated (i.e., red and yellow paths), missing low-frequent paths (e.g., green path).[1]

Indeed, LTR is challenging because 1) various factors, e.g., spatial-temporal information, road types and road network constraints, need to be considered; 2) a proper objective, taking these factors into account, to measure the quality of a generated route is needed to guide the model learning, but conventional metrics based on individual road segments fail to consider the route as a whole; and 3) it's desirable to return a variety of acceptable routes for a given RPQ.

To address these challenges, in this paper, we propose to explore *Generative Adversarial Networks* (GANs) for route planning. GANs have attracted significant research interests due to their ability in generating realistic high-dimensional complex data, such as images [10**?**], videos [**?**] and texts [30]. A GAN model typically consists of two neural networks: a *generator* G and a *discriminator* D, which are iteratively trained against each other. Here, G takes a RPQ and a noise vector z as input to generate a *reachable path* from the source to the destination as the output. By feeding different noise vectors, G is able to non-deterministically generate various paths for the same query. On the other hand, instead of directly designing an objective to train G (e.g., minimal cost or maximum likelihood to travel on a road segment, as previous works do), the discriminator D is trained to assess whether a generated path from G is *realistic* (i.e., indifferentiable from those in the real data). In other words, D learns directly from the data to measure the quality of generated paths (i.e., the more similar to the real data, the more realistic) as the objective to train G. We exploit the proven power of RNN-based models, e.g., LSTM, in modeling sequential data for both G and D so they can effectively generate a path by sequentially predicting the next step from the source to the destination and classify the path as a generated path or a real path, respectively. However, directly generating a path from the source to the destination in the road network is difficult as it's important for G to capture the global information and topological structure of road networks to generate realistic paths. To address this issue, inspired by existing GANs for high-resolution image generation [13], as shown in Figure 2, we propose to generate the path *progressively*, starting by generating a "low-resolution" path, consisting of a sequence of coarse-partitioned grid cells, which gradually grows into "higher-resolution" paths consisting of sequences of finer-partitioned cells, and eventually produce a realistic path on the road network, i.e., consisting of a sequence of intersections. This progressive generation process

[1]NASR [25], a state-of-the-art for LTR, generates only red and yellow paths, while our method generates all red, yellow and green paths.

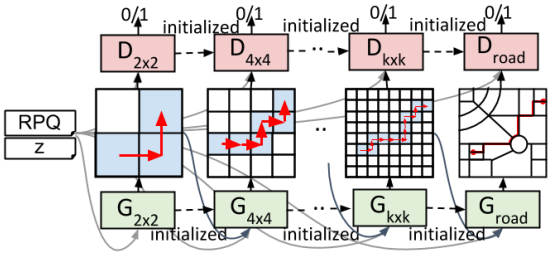


Figure 3: ProgRPGAN

not only effectively captures the global information in geometrical maps and the road network, but also speed up the training and improve stability [13].

To realize this idea, we propose the *Progressive Route Planning GAN* (ProgRPGAN) framework. As shown in Figure 3, ProgRPGAN starts training the generator $G_{S\times S}$ and discriminator $D_{S\times S}$ (e.g., $S = 2$), which takes a query to generate a low-resolution path on the $S\times S$ grid map and classifies if this path is generated or real, respectively. As the training advances, it progressively trains the generators and discriminators for generating higher-resolution paths on 4×4, 8×8, ..., $K\times K$ grid maps. We propose to exploit LSTM for both generators and discriminators. More specifically, a generator $G_{k\times k}$ generates a sequence of cells in a $k\times k$ grid map by sequentially predicting the next cell to move towards the destination. Moreover, $G_{k\times k}$ takes not only the RPQ but also the path generated by $G_{\frac{k}{2}\times\frac{k}{2}}$ as the input to guide the generation of a higher-resolution path. Each $G_{k\times k}$ and $D_{k\times k}$ is initialized by $G_{\frac{k}{2}\times\frac{k}{2}}$ and $D_{\frac{k}{2}\times\frac{k}{2}}$, respectively, to speed up the training (by avoiding learning from scratch) and improve stability. Finally, G_{road} and D_{road} (which are also LSTM models) are trained respectively to generate the highest-resolution path (which consists of a sequence of road segments) as the final output and to classify if the final path is generated or real. Moreover, we propose to pre-train embeddings of grid cells in grid maps and intersections in the road network in order to capture the network topology and external factors, e.g., types of road segments, to facilitate model learning. To the best of our knowledge, this is the first attempt to explore the GAN framework for route planning.

The major contributions of this work are listed below.

- **Novel ideas for learning to route**. We analyze the challenges in learning to route on road networks and propose to overcome these challenges by exploring the GAN framework to progressively generate realistic paths for route planning.
- **A new framework for route planning**. We propose ProgRPGAN for route planning on road networks, which consists of a sequence of (generator, discriminator) pairs. While capturing the global knowledge in grid maps and the road network, generators progressively generate paths on grid maps of low to higher resolutions, and eventually a realistic path on the road network. By feeding noise vectors, generators are able to generate a variety of paths for the given query. Instead of relying on a pre-designed objective in generators to guide the learning, discriminators are used to assess if a path generated from generators is realistic.
- **Empirical evaluation using real-world data**. We evaluate ProgRPGAN by conducting a comprehensive evaluation

using two large-scale real-world trajectory datasets in comparison with several baselines, including graph-search based and learning based methods. Empirical result shows ProgRPGAN soundly outperforms all existing methods and effectively generates various realistic paths for the given query.

The rest of this paper is structured as follows. We review the related work in Section 2 and provide problem definition and analysis in Section 3. We detail the proposed ProgRPGAN framework in Section 4 and show experiment results in Section 5. Finally, we conclude the paper in Section 6.

2 RELATED WORK

Here we review related works on route planning and GAN.

2.1 Route Planning

Route planning is a fundamental task in ITS, which can be examined in the following aspects: 1) types of trajectory data, such as GPS data [6, 16, 26, 31], POI check-in data [5, 23], etc.; 2) personalization, i.e., personalized [7, 11, 25] or non-personalized [6, 12, 17, 31]; and 3) physical constraints, e.g., moving distance [6], grid maps [14, 26] or road networks [25, 27]. In this work, we focus on *non-personalized route planning on road networks by exploiting GPS data*, aiming to train machine learning models to generate realistic paths on road network for a route planning query. As mentioned, existing works on route planing generally fall into two categories: graph-search based and learning based approaches.

Graph-search Based Route Planning. Early works on route planning are graph-search based. By modeling the travel cost in different aspects (e.g., moving distance, travel time, fuel consumption, etc.) on individual road segments, find the path with minimal cost by applying graph-search algorithms, e.g., Dijkstra and A* algorithm on the road networks [12, 31]. BBS [15] encodes multiple costs to form a multi-attributed road network to generate multiple skyline paths. STHMM and PACE [28, 29] model the travel time on each road segment as a distribution by using hidden markov models to generate the path with minimal travel time. MPR, RICK and TPMFP [6, 17, 26] consider the travel frequency derived from historical trajectories along with other costs (e.g., distance and travel time) to find the top-k paths with lowest cost or highest transition probability. To address the data sparsity issue faced in estimating the cost on road networks, L2R [11] clusters frequently traversed intersections and road segments as groups, called *regions*, using historical trajectory data to form a region graph, and then finds paths with minimal cost on the region graph. Assuming that mobile users prefer a path with optimal cost, these approaches highly depend on the quality of cost estimate on road segments.

Learning Based Route Planning. Recently, deep learning methods are developed to model the spatial-temporal dynamics in trajectory data for path generation or route planning. Particularly, Recurrent Neural Network (RNN) based models, such as Long Short-Term Memory (LSTM), have been widely used for modeling trajectories, aiming to sequentially predict the next step (in terms of intersection or road segment) to generate most likely paths. Various RNN-based models are proposed, e.g., hierarchical RNN [32], RNN with spatial-temporal contexts [16], RNN with road network constraints [27] and RNN with grid map constraints [14]. However, these works focus on predicting short-term movement, e.g., the next step recommendation, without paying attention on issues of route planning, although some of them [14, 27] do suggest to feed the destination as an input in order to apply their models on route planning. One recent work, NASR [25] models i) the cost of current path from the source, and ii) additional cost required to approach the destination, to guide the next step prediction for route planning. These methods, directly generating a path by predicting the next step consecutively without exploiting global knowledge of geographical areas and topological structure of road networks, do not fare well, especially for long paths. Moreover, although both graph-search based and learning based approaches may be able to generate multiple paths by searching paths with top-k minimal cost or applying beam search [22] to generate top-k most likely paths, the generated paths usually are different only in a small portion of road segments, failing to generate a variety of realistic routes.

2.2 Generative Adversarial Network

Generative adversarial networks (GANs) have achieved great success recently in generating realistic complex data, such as images [10], videos [18] and texts [30]. A typical GAN model consists of two submodels, a *generator* G and a discriminator D. G aims to generate realistic data by learning a transformation from a noise prior to the data distribution of the real dataset, while D learns to decide, by classification, whether a sampled data is from the real dataset or the synthetic dataset generated by G [10]. G and D are trained iteratively in an adversarial manner. Eventually G generates a variety of realistic data (using different input noises) such that a discriminator could not differentiate the generated samples from the real data. To improve the quality, stability and variation of GAN for image generation, *progressive GAN* is proposed to build the generator and discriminator progressively. It starts with low-resolution images and progressively adds more layers of generators and discriminators to generate and assess images of increasing resolutions [13], respectively. To our best knowledge, GAN has not been explored for trajectory data and route planning.

3 RESEARCH PROBLEM AND CHALLENGES

In this section, we introduce important terms, describe the targeted research problem and discuss the challenges.

Definition 1. ***Road Network.*** *A road network is a directed graph $G = (V, E, \Psi)$, where V is a set of nodes denoting intersections and each node $v \in V$ contains a geographic location $(v.lng, v.lat)$ (i.e., longitude and latitude); $E \subseteq V \times V$ is a set of directed edges denoting road segments; and $\Psi : E \rightarrow R$ is a type mapping function for edges, e.g., a road segment is a highway.*

Definition 2. ***Path.*** *A path p is a sequence of connected nodes on a road network, i.e., $p = \{v_1, ..., v_{|p|}\}$.*

Definition 3. ***Trajectory.*** *A trajectory is a sequence of spatio-temporal sample points generated from the movement of a road user on a path, where a sample point contains a location (i.e., longitude and latitude) and a timestamp.*

Definition 4. ***Learning to Route on Road Networks.*** *Given a road network G and a path dataset $D = \{(p_i, d_i)\}_{i=1}^{|D|}$, where p_i is*

the i-th path and d_i is the departure time, a learning-to-route model is learned for a given route planning query (RPQ) q, specified as a 3-tuple (source s_q, destination d_q, departure time b_q), to generate k realistic paths on G, to be traveled from s_q to d_q, departing at b_q.

To implement ProgRPGAN, we face several new challenges: (1) *Progressive route planning.* A well-designed framework, consisting of a sequence of (generator, discriminator) pairs, is critical for generating realistic paths. Several issues arise in the framework design. While we propose to exploit LSTM for both of generators and discriminators, how to transfer the knowledge of previous-level paths to facilitate efficient generation and assessment of a high-resolution path? How to incorporate the physical constraints, e.g., the topology of the road network, for generation and assessment? (2) *Representation learning for grid cells and intersections.* To facilitate the generation and assessment of ProgPRGAN, we propose to pre-learn embeddings of grid cells in different resolution grid maps and intersections in the road network. A number of technical issues arise. How to pre-learn embeddings of grid cells and intersections simultaneously? How to encode various information into embeddings, e.g., the geometrical correlation between grid cells, the types of road segments, etc.? (3) *Progressive learning process.* How to design an effective and efficient learning process in a progressive manner? How to initialize a pair of a generator and a discriminator by the previous-level generator and discriminator to avoid learning from scratch? These are research questions to be addressed in our design of ProgRPGAN.

4 THE PROGRPGAN FRAMEWORK

The ProgRPGAN framework progressively generates realistic paths for a given query in an adversarial manner. As shown in Figure 3, we train a sequence of (generator, discriminator) pairs, $\{(G_{S\times S}, D_{S\times S}), (G_{2S\times 2S}, D_{2S\times 2S}), ..., (G_{K\times K}, D_{K\times K}), (G_{road}, D_{road})\}$, to generate and assess paths, respectively, on grip maps of increasing resolutions and eventually the road network to produce the final output. In this section, we detail our design of ProgRPGAN, including the generator, the discriminator, and the learning process.

4.1 Generator

We propose to exploit LSTM for the generator model which generates a path by sequentially predicting the next step (in terms of grid cell or intersection) from the source to the destination of an RPQ. The generation process not only takes the query and a grid map (or a road network) as the inputs, but also a random noise vector (for variety of realistic paths) and its previous-level path to guide the current path generation. In the following, we first introduce the design of the generator and the generation process for a $k \times k$ grid map and then present a similar design for the road network.

Given an RPQ $q = (s_q, d_q, b_q)$, a noise vector z and a previous-level path $p' = \{c'_1, c'_2, ..., c'_{|p'|}\}$ in the $\frac{k}{2} \times \frac{k}{2}$ grid map, a generator $G_{k\times k}$ generates a path consisting of a sequence of cells in the $k \times k$ grid map by sequentially predicting the next cell to move towards the destination. At the beginning, it first maps s_q and d_q to the corresponding grid cells in the $k \times k$ grid map, denoted as c_1 and c_d, respectively. If c_1 and c_d are the same cell, it directly returns $\{c_1\}$, a path with only one cell, as the output. Otherwise, the generation process continues. As shown in Figure 4, at the

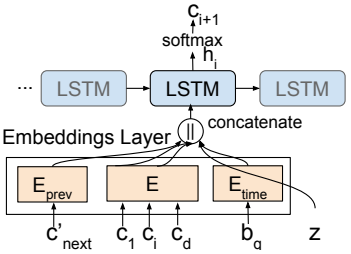


Figure 4: i-th state of generator

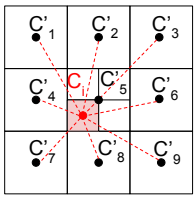


Figure 5: Embeddings transferring

i-th state of the generation process, to predict the next grid cell c_{i+1}, $G_{k\times k}$ takes several items as input, including the current cell c_i, the source cell c_1, the destination cell c_d, the departure time b_q, the noise vector z and the next cell c'_{next} in p', where c'_{next} is set as the grid cell c'_{j+1} in p' whereas c_i is located in c'_j of p'. If there is no next grid cell (i.e., c_i is in the last grid cell of p'), c'_{next} is set to zero. $G_{k\times k}$ first projects c_i, c_1, c_d, b_q and c'_{next} into a latent embedding space and then concatenates all these embeddings with z as the input for stacked LSTM cells. More specifically, c_i, c_1 and c_d are replaced by their embeddings recorded in the projection/embedding matrix $E \in \mathcal{R}^{H\times k^2}$ by table lookup, where k^2 is the number of grid cells in the $k \times k$ grid map, H is the dimensionality of embeddings, and each column of E records the embeddings of its corresponding grid cell. Note that the embedding of c'_{next} is obtained by lookup in the embedding matrix E_{prev} learned in the previous level. The embedding of c'_{next} is set as a zero vector (i.e., zeros in all dimension) for the lowest-resolution (e.g., 2×2) grid map as there is no previous-level path. Regarding the embedding of departure time b_q, we first generate a vector of time information, i.e., $\vec{b_q} = \{day\_in\_a\_week, week\_day\_or\_weekend, hour, minute\}$, from b_q and then transform $\vec{b_q}$ into its embedding by $E_{time} \times \vec{b_q}$, where $E_{time} \in \mathcal{R}^{H_{time}\times|\vec{b_q}|}$ is the projection matrix and H_{time} is the dimensionality of the embedding space.

After feeding the concatenated input to the stacked LSTM cells, to predict the next cell c_{i+1}, the output of the i-th state of the LSTM cells, h_i, is projected by a matrix $W \in \mathcal{R}^{k^2\times|h_i|}$ followed by a softmax function to estimate the *transition probability* of each grid cell in the $k \times k$ grid map. However, the transition in a grid cell is constrained by the topology of the grid map, i.e., the next grid cell to move to should be adjacent to the current grid cell. Moreover, it is proved that RNN-based models hardly learn the topology information well automatically [27]. Thus, we enforce the topological constraint by deriving $P(c_{i+1}|c_{1:i})$, i.e., the conditional probability for a grid cell c_{i+1} to be selected as the next step, given the previous grid cells $c_{1:i}$ have been passed by, as follows.

$$P(c_{i+1} = c_j|c_{1:i}) = P(c_{i+1} = c_j|h_i) = \frac{exp(W\cdot h_i \cdot adj_{c_i,c_j})}{\sum_{c_j\in\mathcal{N}(c_i)} exp(W\cdot h_i)}$$

$$\text{where } adj_{c_i,c_j} = \begin{cases} 1, & c_j \in \mathcal{N}(c_i) \\ 0, & otherwise \end{cases}$$

Here $\mathcal{N}(c_i)$ denotes the 8 adjacent grid cells of c_i, i.e., at up, down, left, right and four diagonal corners to c_i. Setting the transition probability of grid cells not adjacent to c_i to zero allows the model to focus on updating only the weights of those cells adjacent to c_i.

The generation process continues until it reaches the destination cell c_d or exceeds the maximal length of the generation process.

For path generation on road network, G_{road} acts the same way as generating a path of grid cells but instead predicting a sequence of intersections in the road network. Moreover, it embeds the road network topology to constrain the transitions on intersections.

Transfer Learning in Generators. To avoid learning a generator from scratch to improve training efficiency and stability, and to transfer global knowledge learned from geometrical maps to facilitate path generation, we propose to initialize a generator's parameters based on those of its previous-level generator, including embeddings matrices E_{prev}, E, E_{times}, LSTM cells and W. In the following, we introduce the parameter transfer from $G_{\frac{k}{2}\times\frac{k}{2}}$ to the generator $G_{k\times k}$ and then to G_{road}.

Among the parameters to be initialized in $G_{k\times k}$, E_{prev}, E_{time} and LSTM cells are copied from E, E_{time} and LSTM cells of $G_{\frac{k}{2}\times\frac{k}{2}}$, respectively. W is initialized by replicating W of $G_{\frac{k}{2}\times\frac{k}{2}}$ four times, i.e., the i-th row of W in $G_{\frac{k}{2}\times\frac{k}{2}}$ is copied to $(4\times i+j)$-th rows of W in $G_{k\times k}$ where $j = 0, .., 3$. For the embedding matrix E, we propose to not only initialize it from E in $G_{\frac{k}{2}\times\frac{k}{2}}$ but also the pre-trained embeddings of grid cells (and intersections), derived as follows.

$$E = \alpha E_{trans} + (1-\alpha)E_{pre-trained} \tag{1}$$

where E_{trans} denotes the embeddings transferred from E of $G_{\frac{k}{2}\times\frac{k}{2}}$, $E_{pre-trained}$ is the pre-trained embeddings of grid cells in $k\times k$ grid map and α is for weighting. The pre-training of $E_{pre-trained}$ is detailled later. On the other hand, E_{trans} is aggregated from E of $G_{\frac{k}{2}\times\frac{k}{2}}$ based on the distance between cells. More specifically, as shown in Figure 5, each column of E_{trans}, i.e., the transferred embeddings of c_i in the $k\times k$ grid map, is calculated by aggregating the embeddings of its corresponding surrounding grid cells in $\frac{k}{2}\times\frac{k}{2}$ map (denoted as $c'_1, ..., c'_9$). The aggregation is derived as follows.

$$\vec{c_{i_{trans}}} = \sum_{c'_j \in Sur(c_i)} w_{ij} \cdot \vec{c'_j}; \quad w_{ij} = \frac{1/dis(c_i, c'_j)}{\sum_{c'_j \in Sur(c_i)} 1/dis(c_i, c'_j)}$$

where $Sur(c_i)$ is the surrounding grid cells of c_i in $\frac{k}{2}\times\frac{k}{2}$ grid map, and $dis(c_i, c'_j)$ is the geometrical distance between the centroids of c_i and c'_j. Finally, E_{trans} is set as $[c_1; ...; c_{k\times k}]$.

Next, we discuss the initialization of the generator G_{road} for road network, which is pretty much the same as the initialization of generators for grid maps, except that 1) W of G_{road} is initialized by copying W in $G_{K\times K}$ (the generator for the highest-resolution grid map) where the number of copies depends on how many intersections are covered by a grid cell, and 2) E_{trans} is initialized by considering the geometrical distance between an intersection and the centroids of its surrounding grid cells in the highest-resolution $K\times K$ grid map. Note that, during training, all parameters of a generator are fine-tuned, except E_{prev} is fixed because it is well trained in the previous level.

Pre-train Embeddings of Grid Cells and Intersections. To facilitate the learning of generators and discriminators, we propose to pre-train embeddings of grid cells in different resolution levels and intersections in the road network by encoding 1) the geometrical information of grid maps and the topological structure of road network; and 2) external information of the road network, e.g., types of road segments. Then, the pre-trained embeddings of cells and intersections, denoted as $E_{pre-trained}$, are used to initialize generators as derived in Eq. (1). To achieve this goal, a naive way is to model each grid map as a *grid graph* (where grid cells are denoted as nodes and two adjacent grid cells are linked by an undirected edge) and then apply network representation learning methods, such as DeepWalk [21] on the grid graphs and road network separately to learn embeddings of cells and intersections. However, in this way, the learned embeddings of nodes in different graphs are irrelevant because they are learned separately, which may hurt the initialization of generators while aggregating E in Eq. (1).

To address this issue, we propose to learn the embeddings of grid cells in grid graphs of all resolution levels and intersections in the road network jointly. To achieve this goal, we first model all the grid maps and the road network into a *unified graph*. An example of the unified graph is shown in Figure 6. The unified graph not only consists of the above-mentioned grid graphs in each resolution level and the road network, but also additional edges between cells in adjacent levels (i.e., if a cell is covered by another cell in the previous level) and edges between cells in the highest-resolution grid map and intersections (i.e., if an intersection is covered by a cell in the grid graph of the highest resolution). Moreover, each edge in the unified graph is labeled by an edge type which indicates the relationship between its two end nodes. For example, in the example unified graph, edge (c_1, c_2) is labeled by "2 − 2" indicating they are both in the 2×2 grid map, edge (c_2, c_3) is labeled by "2 − 4" indicating a refinement from the 2×2 grid map to the 4×4 grid map, and edge (v_4, v_5) is labeled by "primary" indicating the type of road segment between the two intersections. In other words, the unified graph is a *heterogeneous information network (HIN)* which contains nodes and edges labeled by various types.

After constructing the unified graph, we propose to exploit an existing network representation learning method for HIN, called *HIN2Vec* [9], to learn embeddings of nodes (i.e., grid cells and road segments here). HIN2Vec aims to not only encode the topology of the graph (i.e., if two nodes have more paths between them, their embeddings are more similar) but also the type of path between nodes (i.e., capturing the sequences of edge types between nodes) into the learned embeddings. Specifically, HIN2Vec applies random walks in the graph to sample training data with negative sampling mechanism [19]. For the unified graph, which consists of multiple graphs (representing grid maps of different resolutions and the road network) linked by interconnecting edges, we propose a *parameterized random walks* to sample the training data for learning embeddings. Specifically, we introduce a *leaping parameter* β, representing the probability of leaping into a node not in the current grid graph, in order to control the walking process. With a large β, the walking process tends to select the next node in a grid graph different from the current one (or in the road network). On the other hand, a small β tends to guide the walking process staying in the current grid graph. Based on the generated random walks, we applies a sliding window over the walks to generate various training data samples consisting of two nodes and a path type. For instance, using a sliding window of size 3 (in terms of nodes), the example random walk in Figure 6, {node c_1, edge "2 − 2", node c_2, edge "2−4", node c_3, ... }, may generate $(c_1, c_2$, "2−2"), $(c_2, c_3$, "2−4"), $(c_1, c_3$, "2 − 2 − 4"), ... as data samples.

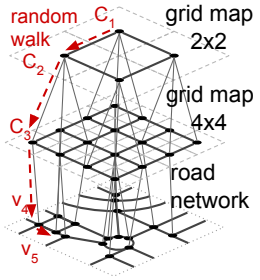


Figure 6: Unified graph

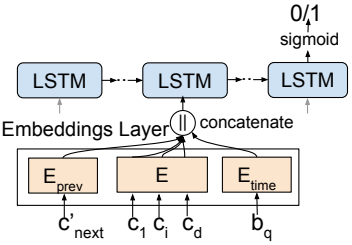


Figure 7: Discriminator

4.2 Discriminator

We also propose to exploit LSTM for the discriminator model. In the following, we introduce the design of the discriminator and the classification process for a $k \times k$ grid map. The design of the discriminator model and the classification process for the road network is the same as for a grid map.

Given a query $q = (s_q, d_q, b_q)$, a path $C = \{c_1, c_2, ..., c_{|C|}\}$ and a previous-level path $p' = \{c'_1, c'_2, ..., c'_{|p'|}\}$ in the $\frac{k}{2} \times \frac{k}{2}$ grid map, a discriminator $D_{k\times k}$ aims to classify if C is a generated or real path. At the beginning, it first maps s_q and d_q to the corresponding grid cells in the $k \times k$ grid map, denoted as c_1 and c_d, respectively. Then, as shown in Figure 7, at the i-ith state, $D_{k\times k}$ takes the current cell c_i, the source cell c_1, the destination cell c_d, the departure time b_q and the next cell c'_{next} in the previous $\frac{k}{2} \times \frac{k}{2}$ grid map as the inputs, then also transforms c_i, c_1, c_d, b_q and c'_{next} as embeddings (as $G_{k\times k}$ does) and concatenates them as the input to stacked LSTM cells. $D_{k\times k}$ sequentially encodes the whole path, and finally the output of the latest state, h_d, is projected by a matrix $W_D \in 1 \times |h_d|$ followed by a sigmoid function to classify if it is generated or real.

To avoid learning the discriminator $D_{k\times k}$ from scratch, we propose to initialize $D_{k\times k}$ by transferring parameters from both $D_{\frac{k}{2}\times\frac{k}{2}}$ and $G_{k\times k}$, including embedding matrices E_{prev}, E, E_{times}, LSTM cells and W_D. More specifically, E_{prev}, E and E_{time} are initialized by copying the E_{prev}, E and E_{time} in $G_{k\times k}$, respectively. On the other hand, LSTM cells and W_D are initialized by copying from the LSTM cells and W_D in $D_{\frac{k}{2}\times\frac{k}{2}}$, respectively. Note that, during training, E_{prev}, E and E_{time} are fixed (because they are trained by $G_{k\times k}$ for the generation), but LSTM cells and W_D are fine-tuned.

4.3 Learning Process

In ProgRPGAN, we progressively train a sequence of (generator, discriminator) pairs in an adversarial manner, where G aims to maximize the estimated "reward" from D, i.e., the probability of a generated path to be classified as a real path, and D aims to minimize the error of the classification among generated and real paths. To train a G, we apply the policy gradient method [24] to maximize the estimated rewards of generated paths, with a Monte-Carlo rollouts (MCR) mechanism [30] to estimate the individual reward for each next-step prediction (i.e., a grid cell or an intersection) during the path generation process. The estimated reward $\bar{R}$ and its gradient $\nabla\bar{R}$ of a training epoch is derived as follows.

$$\bar{R} = \sum_{m=1}^{M} R(p^m) P(p^m|\theta, q, z)$$
$$\nabla\bar{R} \approx \frac{1}{M} \sum_{m=1}^{M} R(p^m) dlogP(p^m|\theta, q, z)$$

where M is the number of samples of a training epoch, $R(p^m)$ is the reward for the whole path p^m from D, $P(p^m|\theta, q, z)$ is the probability to generate p^m by current G given a query q and a noise vector z, and θ denotes the trainable parameters of G. Then, we apply MCR to estimate the reward of the next prediction at i-th state of a path p, denoted as $R_{p_{1:i}}$. The idea behind MCR is to additionally generate N paths starting from i-th state of p to the destination, and then get the average of N rewards from D to estimate $R_{p_{1:i}}$, which is derived as $R_{p_{1:i}} = \frac{1}{N} \sum_{p^n \in MC(p_{1:i}, N)} D(p^n)$ where $MC(p_{1:i}, N) = \{p^1, ..., p^N\}$ denotes the generation of N additional paths starting from $p_{1:i}$ to the destination. Finally, $\nabla\bar{R}$ is modified as follows.

$$\nabla\bar{R} \approx \frac{1}{M} \sum_{m=1}^{M} \sum_i R_{p^m_{1:i}} dlogP(p^m_{1:i}|\theta, q, z)$$

We update G's parameters by gradient ascent as follows.

$$\theta \leftarrow \theta + \eta\nabla\bar{R} \quad (2)$$

where η is the learning rate. On the other hand, D aims to minimize the classification error (i.e., the cross entropy) and we update D's parameters by gradient descent as follows.

$$J \approx -\frac{1}{M} \sum_{m=1}^{M} [logD(p^m|\phi)] - \frac{1}{M} \sum_{m=1}^{M} [log(1 - D(\tilde{p^m}|\phi))]$$
$$\phi \leftarrow \phi - \eta\nabla J \quad (3)$$

where p^m is a path sampled from the real paths, $\tilde{p^m}$ is a generated path from G and ϕ denotes the trainable parameters of D.

Algorithm 1 Train a pair of G and D

Require: generator G; discriminator D; a path dataset T
1: Initialize G and D;
2: **repeat**
3: **for** *d-step* **do**
4: Sample M paths $\{p^1, ..., p^M\}$ from T and generate corresponding queries $\{q^1, ..., q^M\}$
5: Sample M noise vectors $\{z^1, ..., z^M\}$
6: Generate M paths $\{\tilde{p^1}, ..., \tilde{p^M}\}$, $\tilde{p^m} = G(q^m, z^m)$
7: Update D by Eq. (3)
8: **end for**
9: **for** *g-step* **do**
10: Sample M paths from T to generate queries $\{q^1, ..., q^M\}$
11: Sample M noise vectors $\{z^1, ..., z^M\}$
12: Generate M paths $\{\tilde{p^1}, ..., \tilde{p^M}\}$, $\tilde{p^m} = G(q^m, z^m)$
13: Update G by Eq. (2)
14: **end for**
15: **until** G and D converge

Algorithm 1 details the learning process for a pair of generator G and discriminator D. At the beginning of the training, we initialize G and D by their corresponding generator and discriminator in the previous level. For the lowest-resolution G and D, we initialize them with random weights. Then, G and D are trained alternatively with *g-step* and *d-step*, respectively, until they converge.

5 EXPERIMENTS

In this section, we conduct extensive experiments using two large-scale real-world trajectory datasets to evaluate the performance of ProgRPGAN against several existing methods. We also perform sensitivity tests on parameters of ProgRPGAN, examine several issues in ProgRPGAN.

Table 1: Statistics of road networks and trajectories

Road network data	Porto	Chengdu
# of intersections	143,877	101,528
# of road segments	342,838	229,599
Trajectory data	**Porto**	**Chengdu**
# of paths	596,943	1,878,854
moving distance mean	5.836 km	4.226 km
# of road segments mean	74.211	22.480

5.1 Datasets

Two trajectory datasets and corresponding road networks extracted from OpenStreetMap [1] are used in the evaluation:
Porto taxi data [2] collects trajectories of 442 taxis running from January 2013 to June 2014 in Porto, Portugal.
Chengdu taxi data [3] collects 1.4 billion GPS points of 14,864 taxis running in August 2014 in Chengdu, China. We segment the sample points of each taxi into trajectories based on a 60-second time gap.

For each road network data, we extract intersections and road segments from raw OpenStreetMap data to form a road network. We leave the details in Appendix A. For each trajectory dataset, we first select trajectories with travel time within 1 to 60 minutes and match the trajectories to the road network by existing map matching techniques [20] to obtain the corresponding sequence of intersections (i.e., paths). Then we filter mismatched paths which have much longer (+10%) or shorter (-10%) moving distances comparing with their raw trajectories. Some statistics of the extracted road networks and paths are summarized in Table 1.

5.2 Baseline Methods for Comparison

The route planning methods we evaluate for comparison include baselines, graph-search based methods and learning based methods.
Short and **Fast**, graph-search based baselines that use Yen's algorithm to find top-k shortest/fastest paths in terms of geometrical distance and travel time (estimated by spd-LSTM, a state-of-the-art method predicting the speed of each road segment), respectively.
MPR [6], a graph-search based method that models a transfer graph based on the traversed frequency and the geometrical distance to find the most popular path.
RICK [26], a graph-search based method that finds top-k paths based on geometrical distance, travel time and traversed frequency.
STRNN [16], an RNN based learning method that sequentially predicts the next step by considering the spatial and temporal contexts.
CSSRNN [27], an RNN based learning method that sequentially predicts the next step by considering the road network constraint.
NASR [25], an RNN based learning method that adopts neural network based reinforcement learning techniques to learn two costs, including a cost of the current path from the source and a cost required to reach the destination, to guide the next step prediction for route planning.

5.3 Experimental Setup

In each experiment, for a dataset, we randomly split it into three folds, 80%, 10% and 10% as the training set, the validation set and the test set, respectively. We use the training set to train models while using the validation set to select the best models, and evaluate the performance using the test set. For each test sample, for a model, we generate three paths and report the best performance among them (i.e., top-3 minimal cost paths from a graph-search based method, top-3 most likely paths from a learning based method by applying beam search and three paths from ProgRPGAN by feeding different random noise vectors). We repeat each experiment for 5 times and report the mean of the different runs. We use micro *precision*, *recall*, *F1-measure* and *Jaccard index* between the generated paths and the ground truth among road segments of paths weighted by the road segment lengths as the performance metrics. We leave the equation of each metric in Appendix B. To report more details of the performance, we further split the test set into three folds by the lengths, namely *short* (less then 3km), *medium* (3km to 6km) and *long* (more than 6km). The distributions among the three folds of Porto and Chengdu datasets are (31.4%, 33.9% and 34.7%) and (51.6%, 20.3% and 19.1%), respectively.

Regarding the default parameter settings of ProgRPGAN, the dimensionality of cell/intersection embeddings H, time embeddings H_{time} and LSTM cell are set to 128, the weight α for the generator initialization is set to 0.8. The dimensionality of the noise vector z is 16. We adopt a double layer LSTM model. For both datasets, we consider a 64km × 64km region and generate a sequence of generator and discriminator pairs, i.e., $\{(G_{S\times S}, D_{S\times S}), (G_{2S\times 2S}, D_{2S\times 2S}), ..., (G_{K\times K}, D_{K\times K}), (G_{road}, D_{road})\}$, where S is set to 8 and K is set to 64. For pre-training embeddings, the sliding window size for training data preparation is set to 3 and the leaping parameter β for random walks is set to 0.5. For model learning, the number of sampling M for an epoch is set to 64, the number of sampling N for MCR is set to 16, the learning rate η is set to 0.0001, the number of training steps of generates *g-step* and discriminators *d-step* are set to 1 and 20, respectively. The maximal length of the cell path generation in each grid map and the path generation in the road network is determined by data such that it covers more than 99% cell paths and final paths. To obtain converged result, the number of iterations for model training varies for individual models and different datasets. For all compared methods, we tune the best parameter settings. Our model is implemented in python with Tensorflow. We train and evaluate all models on a server with NVIDIA GTX 1080 GPUs and one Intel Core i5-8400 CPU on Ubuntu 18.04.

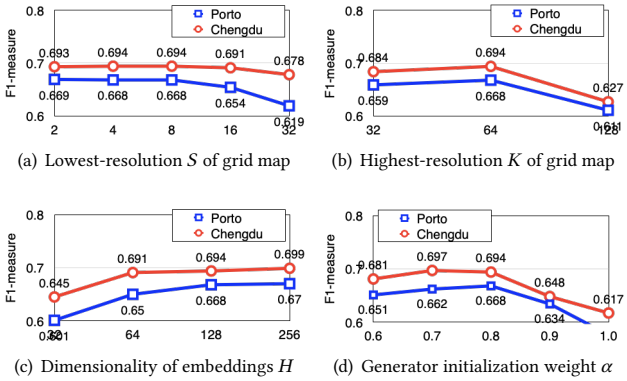
5.4 Evaluation of Models

The performance obtained by all evaluated methods is summarized in Table 2.[2] ProgRPGAN outperforms all the compared methods on all three test folds, especially on the *long* fold which is the hardest. As shown, the improvement ratio of F1-measure (compared with the best of these existing models, marked by '*') are 13.02% and 9.46% for the *long* folds in Porto and Chengdu datasets, respectively. We have the following observations from the comparison.
Travel time is an important criteria. Comparing with other methods, Fast, RICK, NASR and ProgRPGAN which directly or potentially consider travel time on road segments have better performance. Among them, Fast and RICK directly consider travel

[2]Due to space constraint, "P" and "C" denote Porto and Chengdu datasets, respectively. "s", "m" and "l" denote the short, medium and long test folds, respectively. "ST" and "CSS" denote STRNN and CSSRNN, respectively

Table 2: Performance Evaluation

Data-set		Precision								Recall							
		Short	Fast	MPR	RICK	ST	CSS	NASR	**Ours**	Short	Fast	MPR	RICK	ST	CSS	NASR	**Ours**
P	s	0.659	0.754	0.671	0.770	0.698	0.691	*0.791	**0.822**	0.593	0.708	0.638	0.775	0.681	0686	*0.781	**0.815**
	m	0.505	0.610	0.549	0.622	0.512	0.514	*0.670	**0.718**	0.434	0.545	0.476	0.601	0.476	0.464	*0.661	**0.713**
	l	0.247	0.528	0.263	0.547	0.333	0.370	*0.598	**0.677**	0.182	0.424	0.224	0.496	0.271	0.312	*0.585	**0.661**
C	s	0.790	0.790	0.799	0.808	0.789	0.790	*0.818	**0.830**	0.755	0.761	0.793	0.800	0.763	0.761	*0.812	**0.825**
	m	0.650	0.675	0.656	0.691	0.660	0.662	*0.714	**0.761**	0.601	0.632	0.633	0.665	0.621	0.631	*0.701	**0.744**
	l	0.448	0.552	0.461	0.572	0.483	0.499	*0.642	**0.704**	0.376	0.481	0.404	0.541	0.412	0.438	*0.626	**0.685**
Data-set		F1-measure								Jaccard							
		Short	Fast	MPR	RICK	ST	CSS	NASR	**Ours**	Short	Fast	MPR	RICK	ST	CSS	NASR	**Ours**
P	s	0.623	0.731	0.655	0.764	0.679	0.674	*0.786	**0.821**	0.453	0.577	0.485	0.624	0.526	0.506	*0.650	**0.695**
	m	0.467	0.575	0.496	0.611	0.491	0.488	*0.667	**0.715**	0.303	0.405	0.331	0.442	0.325	0.323	*0.501	**0.557**
	l	0.209	0.470	0.243	0.519	0.301	0.337	*0.591	**0.668**	0.115	0.303	0.139	0.351	0.175	0.201	*0.419	**0.501**
C	s	0.773	0.775	0.794	0.804	0.776	0.775	*0.816	**0.827**	0.630	0.635	0.659	0.675	0.633	0.633	*0.688	**0.705**
	m	0.623	0.652	0.643	0.680	0.638	0.645	*0.707	**0.750**	0.453	0.485	0.475	0.514	0.471	0.476	*0.546	**0.598**
	l	0.404	0.513	0.431	0.555	0.443	0.466	*0.634	**0.694**	0.252	0.345	0.273	0.383	0.286	0.304	*0.464	**0.531**



(a) Lowest-resolution S of grid map (b) Highest-resolution K of grid map

(c) Dimensionality of embeddings H (d) Generator initialization weight α

Figure 8: Parameter Sensitivity of ProgRPGAN

time on road segments. NASR learns a latent cost potentially capturing travel time by feeding geometrical distance and timestamp, and ProgRPGAN encodes the road segment types in pre-trained embeddings which reflect the travel time.

Simply applying next step prediction models does not fare well. Comparing with other methods, STRNN and CSSRNN have the worst performance (except for Short). This suggests that simply applying RNN-based models, which fails to capture the global knowledge of geographical maps and the road network, to sequentially predict next steps for route planning is impractical.

Progressive GAN based path generation is effective for route planning. Comparing with Fast, RICK and NASR, the performance improvement of ProgRPGAN is clear and impressive, because ProgRPGAN exploits discriminators to learn a cost of a generated path rather than manual-craft costs (as Fast and RICK do) and progressively generates a path on low-resolution to higher-resolution grid maps rather than directly on the road network (as NASR does).

5.5 Parameter Sensitivity in ProgRPGAN

We examine the impact of important parameter settings in ProgRPGAN on its performance. We only show the result on the long test fold in F1-measure, as the other test folds have a similar trend with the long test fold. Due to the lack of space, we leave the details in tuning other parameters in Appendix C.

Resolution of grid maps. Generally speaking, grid maps of very low resolution are not informative as most of paths are in the same cell, e.g., a 2×2 grid map consists of four 32km $\times$ 32km cells. On the other hand, grid maps of too high a resolution, e.g., a 128×128 grid map consists of 0.5km $\times$ 0.5km cells, leads to long cell sequences and unnecessary complexity for learning. Figure 8(a) and (b) show that the best performance is achieved when the lowest-resolution S is set to 8 and the highest-resolution K is set to 64 for both datasets.

Dimensionality of embeddings H. In general, a small dimensionality is not capable of capturing the patterns between cells and intersections to fit the training data, but a large dimensionality may require a large amount of training data. Figure 8(c) shows that setting H as 128 for both Porto and Chengdu datasets achieves converged performance.

Impact of pretrained embedding. By varying α between 0.6 to 1.0, as shown in Figure 8(d), we observe that setting α as 1.0 (i.e., does not incorporate pre-trained embeddings) has the worst performance, indicating that the information encoded in pre-trained embeddings is very useful for route planning. On the other hand, setting α as 0.8 achieves the best performance, suggesting that transferring embeddings from the previous-level is also useful while incorporating pre-trained embeddings.

5.6 Study of Unique Issues in ProgRPGAN

In this section, we examine the following unique issues arising in the design of ProgRPGAN: i) the effect of progressive path generation, ii) the effect of incorporating graph constraints in the grid maps and road network topology for path generation, and iii) the effect of time information in path generation. The results are shown in Figure 9 where the `Best` setting denotes ProgRPGAN with the default settings described in Section 5.3.

Regarding the effect of progressive path generation, we compare `Best` with the `No Progressive` setting which directly generates a path on the road network without incorporating any grid map. Figure 9 shows that `Best` significantly outperforms `No Progressive`

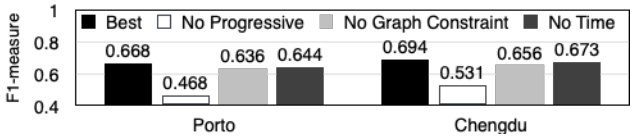


Figure 9: Comparison of approaches to issues

in both datasets, indicating that the progressive generation of ProgPRGAN is effective for route planning. Next, we compare `Best` with the `No Graph Constraint` setting which does not enforce graph constraints in path generation. As shown in Figure 9, `Best` outperforms `No Graph Constraint` about 4.8% and 5.6% in Porto and Chengdu datasets, respectively. This suggests that considering the topology constraints of grid graphs and road network is useful for route planning. Finally, Figure 9 shows that `Best` outperforms `No Time`, which does not feed time information in path generation, for about 3.3% in both Porto and Chengdu datasets. This suggests that the time information is also useful.

6 CONCLUSION

This study focuses on learning to route on road networks. Prior works fail to capture the global knowledge in geographical maps and road networks, and thus do not generate realistic paths. To fill in this gap, we explore the GAN technique to capture the various information on geographical maps and road network to progressively plan routes on grid maps of low to high resolutions and finally generate realistic paths on the road network. To achieve the goal, we propose a novel progressive GAN framework, namely *ProgRPGAN*, which consists of a sequence of (generator, discriminator) pairs, aiming to generate and assess realistic paths in adversarial manner. Moreover, we exploit network representation learning methods to simultaneously pre-learn embeddings of grid cells in grid maps and intersections in road network by capturing the network topology and external factors, e.g., road segment types, to facilitate model learning. Empirical result shows that ProgRPGAN soundly outperforms all existing models in multiple large-scale real-world datasets especially for long routes. ProgRPGAN, moreover, effectively generates various realistic paths for a route planning query.

ACKNOWLEDGMENTS

This work is supported in part by the National Science Foundation under Grant No. IIS-1717084.

REFERENCES

[1] 2004. OpenStreetMap. https://www.openstreetmap.org/.
[2] 2015. Taxi Service Trajectory (TST) Prediction Challenge. http://www.geolink.pt/ecmlpkdd2015-challenge/.
[3] 2016. Taxi Travel Time Prediction Challenge. http://www.dcjingsai.com.
[4] Vaida Ceikute and Christian S Jensen. 2013. Routing service quality–local driver behavior versus routing services. In *Proceedings of IEEE International Conference on Mobile Data Management*, Vol. 1. IEEE, 97–106.
[5] Dawei Chen, Cheng Soon Ong, and Lexing Xie. 2016. Learning points and routes to recommend trajectories. In *Proceedings of the ACM International on Conference on Information and Knowledge Management*. ACM, 2227–2232.
[6] Zaiben Chen, Heng Tao Shen, and Xiaofang Zhou. 2011. Discovering popular routes from trajectories. In *Proceedings of IEEE International Conference on Data Engineering*. IEEE, 900–911.
[7] Jian Dai, Bin Yang, Chenjuan Guo, and Zhiming Ding. 2015. Personalized route recommendation using big trajectory data. In *Proceedings of the IEEE International Conference on Data Engineering*. IEEE, 543–554.
[8] Eva Ericsson, Hanna Larsson, and Karin Brundell-Freij. 2006. Optimizing route choice for lowest fuel consumption–potential effects of a new driver support tool. *Transportation Research Part C: Emerging Technologies* 14, 6 (2006), 369–383.
[9] Tao-yang Fu, Wang-Chien Lee, and Zhen Lei. [n.d.]. Hin2vec: Explore meta-paths in heterogeneous information networks for representation learning. In *Proceedings of the ACM Conference on Information and Knowledge Management*.
[10] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Proceedings of the Annual Conference on Neural Information Processing Systems*. 2672–2680.
[11] Chenjuan Guo, Bin Yang, Jilin Hu, and Christian Jensen. 2018. Learning to route with sparse trajectory sets. In *Proceedings of the IEEE International Conference on Data Engineering*. IEEE, 1073–1084.
[12] Evangelos Kanoulas, Yang Du, Tian Xia, and Donghui Zhang. 2006. Finding fastest paths on a road network with speed patterns. In *Proceedings of IEEE International Conference on Data Engineering*. IEEE, 10–10.
[13] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. 2017. Progressive growing of gans for improved quality, stability, and variation. *CoRR* abs/1710.10196 (2017).
[14] Jiangtao Kong, Jian Huang, Hongkai Yu, Hanqiang Deng, Jianxing Gong, and Hao Chen. 2019. RNN-based default logic for route planning in urban environments. *Neurocomputing* 338 (2019), 307–320.
[15] Hans-Peter Kriegel, Matthias Renz, and Matthias Schubert. 2010. Route skyline queries: A multi-preference path planning approach. In *Proceedings of IEEE International Conference on Data Engineering*. IEEE, 261–272.
[16] Qiang Liu, Shu Wu, Liang Wang, and Tieniu Tan. 2016. Predicting the next location: A recurrent model with spatial and temporal contexts. In *Proceedings of the AAAI Conference on Artificial Intelligence*.
[17] Wuman Luo, Haoyu Tan, Lei Chen, and Lionel M Ni. 2013. Finding time period-based most frequent path in big trajectory data. In *Proceedings of the ACM International Conference on Management of Data*. ACM, 713–724.
[18] Behrooz Mahasseni, Michael Lam, and Sinisa Todorovic. 2017. Unsupervised video summarization with adversarial LSTM networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 202–211.
[19] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. [n.d.]. Distributed representations of words and phrases and their compositionality. In *Proceedings of the Annual Conference on Neural Information Processing Systems*.
[20] Paul Newson and John Krumm. 2009. Hidden Markov map matching through noise and sparseness. In *Proceedings of the ACM SIGSPATIAL international conference on advances in geographic information systems*. ACM, 336–343.
[21] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining*. 701–710.
[22] Michael Pinedo. 2012. *Scheduling*. Vol. 5. Springer.
[23] Samia Shafique and Mohammed Eunus Ali. 2016. Recommending most popular travel path within a region of interest from historical trajectory data. In *Proceedings of the ACM International Workshop on Mobile Geographic Information Systems*. ACM, 2–11.
[24] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. 2000. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*. 1057–1063.
[25] Jingyuan Wang, Ning Wu, Wayne Xin Zhao, Fanzhang Peng, and Xin Lin. 2019. Empowering A* Search Algorithms with Neural Networks for Personalized Route Recommendation. In *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining*. 539–547.
[26] Ling-Yin Wei, Yu Zheng, and Wen-Chih Peng. 2012. Constructing popular routes from uncertain trajectories. In *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining*. ACM, 195–203.
[27] Hao Wu, Ziyang Chen, Weiwei Sun, Baihua Zheng, and Wei Wang. 2017. Modeling trajectories with recurrent neural networks. In *Proceedings of the International Joint Conferences on Artificial Intelligence Organization*. 3083–3090.
[28] Bin Yang, Jian Dai, Chenjuan Guo, Christian S Jensen, and Jilin Hu. 2018. PACE: a PAth-CEntric paradigm for stochastic path finding. *International Journal on Very Large Data Bases* 27, 2 (2018), 153–178.
[29] Bin Yang, Chenjuan Guo, and Christian S Jensen. 2013. Travel cost inference from sparse, spatio temporally correlated time series using Markov models. *the VLDB Endowment* 6, 9 (2013), 769–780.
[30] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. 2017. Seqgan: Sequence generative adversarial nets with policy gradient. In *Proceedings on AAAI Conference on Artificial Intelligence*.
[31] Jing Yuan, Yu Zheng, Chengyang Zhang, Wenlei Xie, Xing Xie, Guangzhong Sun, and Yan Huang. 2010. T-drive: driving directions based on taxi trajectories. In *Proceedings of the International Conference on Advances in Geographic Information Systems*. ACM, 99–108.
[32] Stephan Zheng, Yisong Yue, and Jennifer Hobbs. 2016. Generating long-term trajectories using deep hierarchical networks. In *Proceedings of the Annual Conference on Neural Information Processing Systems*. 1543–1551.

A ROAD NETWORK EXTRACTION

We extract road networks from the OpenStreetMap (OSM) [1], which is a publically accessible map website. For each road network data, we download the raw OSM data of the corresponding city equipped with a trajectory dataset (i.e., Porto and Chengdu) for our experiments. We then extract Nodes and Ways data from the raw OSM data as intersections (i.e., nodes) and road segments (i.e., edges), respectively, to form a road network. More specifically, we only extract Ways with a "highway" key in their tag sets (e.g., <key="highway" value="primary"/>) as road segments because OSM also records the boundaries of buildings and areas as Ways (without a "highway" key). For each extracted Way, represented as a sequence of Node ids in the raw OSM data, we only use the starting Node and end Nodes of the sequence as intersections (i.e., nodes in the road network), and remove other internal Nodes which are used to describe the shape of a Way, and then add a directed edge from the starting Node to the end Node in road network if the Way is a one-way road (by checking its "one_way" key); otherwise, add an additional directed edge from the end Node to the starting Node. The edges extracted from a Way is labeled a road segment type as the value of the Way's "highway" key (e.g., "primary" in <key="highway" value="primary"/>) and a geometrical length calculated by the length of the Way (by considering all Nodes of its Node sequence).

B METRICS IN PERFORMANCE EVALUATION

As mentioned, we use micro *precision*, *recall*, *F1-measure* and *Jaccard index* between the generated paths and the ground truth among road segments of paths weighted by the road segment lengths as the performance metrics. Given a tested result set Res consisting of a set of (an real route p and a corresponding generated path $\tilde{p}$) pairs, the four metrics are derived as follows.

$$precision(Res) = \frac{\sum_{(p,\tilde{p})\in Res}\sum_{e\in p\cap\tilde{p}} length(e)}{\sum_{(p,\tilde{p})\in Res}\sum_{e\in\tilde{p}} length(e)}$$

$$recall(Res) = \frac{\sum_{(p,\tilde{p})\in Res}\sum_{e\in p\cap\tilde{p}} length(e)}{\sum_{(p,\tilde{p})\in Res}\sum_{e\in p} length(e)}$$

$$F1-measure(Res) = \frac{2\times precision(Res)\times recall(Res)}{precision(Res)+recall(Res)}$$

$$Jaccard(Res) = \frac{\sum_{(p,\tilde{p})\in Res}\sum_{e\in p\cap\tilde{p}} length(e)}{\sum_{(p,\tilde{p})\in Res}\sum_{e\in p\cup\tilde{p}} length(e)}$$

C ADDITIONAL PARAMETER SENSITIVITY TESTS IN PROGRPGAN

Here we further examine the impact of other parameter settings in ProgRPGAN on its performance. To test the parameter sensitivity of ProgRPGAN, we vary the values of the parameters, 1) the model design of LSTM; 2) the data preparation for the embeddings pre-training; 3) the number of samples in learning ProgRPGAN; and 4) the dimensionality of the noise vector z and the embeddings of time H_{time}, to observe the changes in F1-measure on the long test fold, as shown in Figure 10.

Model design of LSTM. Regarding the model design of LSTM, generally speaking, a small dimensionality or a shallowly stacked

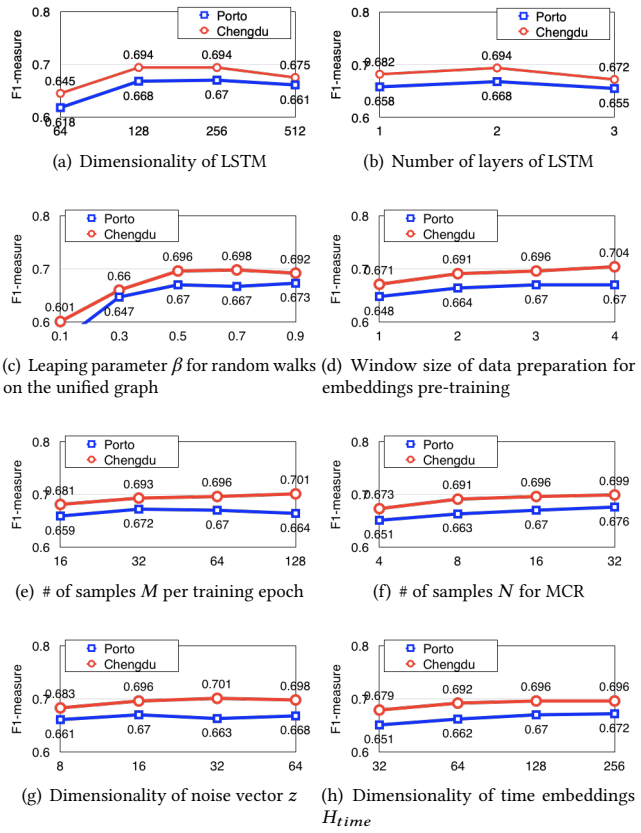
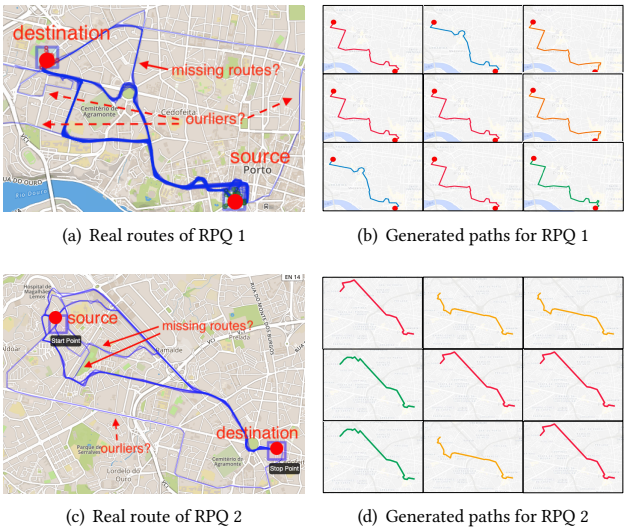


Figure 10: Parameter Sensitivity Tests of ProgRPGAN

LSTM is not capable of capturing the moving patterns along the paths to accurately predict the next step (i.e., a grid cell or an intersection), but a large dimeingsionality or a deeply stacked LSTM may lead to noises, causing overfitting or requiring a large amount of training data. Figure 10(a) shows that setting the dimensionality of LSTM to 128 for both Porto and Chengdu datasets achieves converged performance, while the performance is decreased after the dimensionality is set to larger than 512. On the other hand, for the number of stacked layers of LSTM, Figure 10(e) shows that the performance does not change much when it is set between 1 to 3. Thus, setting it to 2 is a good choice.

Parameterized random walks in embedding pre-training. We study the impact of leaping parameter β for random walks on the unified graph and the sliding window size on random walks to sample training data for HINVec. Figure 10(c) shows that when β is increased (resulting more leaping between adjacent levels of grid graphs and road network), the performance continues to improve and coverage when β is set to 0.5 or 0.7. It suggests that simultaneously learning embeddings of grid cells in grid maps of different resolutions and intersections (i.e., $\beta > 0$) is useful, but sampling more cells/intersections in the same grid graph/road network is better than between adjacent levels of grid graphs and road network (i.e., $\beta = 0.5$ 0.7 achieves converged performance). On the other hand, regarding the sliding window size on random walks to sample training data for HINVec, Figure 10(d) shows that the



(a) Real routes of RPQ 1 (b) Generated paths for RPQ 1

(c) Real route of RPQ 2 (d) Generated paths for RPQ 2

Figure 11: Real routes and generated paths

performance does not change significantly when it is set at between 2 to 4. Therefore,it is set to 3.

Learning process of ProgRPGAN. We examine the number of samples M for a training epoch and the number of sampls N for Monte-Carlo rollouts (MCR). As shown in Figure 10(e), M does not affect the performance significantly while it is greater than 32, and thus we set it to 64 which achieves the converged performance. On the other hand, for N, Figure 10(f) shows that the performance continues to improve while the N is increased because the more number of samples, the more accurate estimation of the individual reward for a next step prediction, but it leads to longer sampling time for training. Thus, N is set to 16 due to the limited improvement made by setting N to 32.

Dimensionality of noise vector and time embeddings. Finally, we study the impact of the dimensionality of noise vector z and time embeddings H_{time}. Figure 10(g) shows that the performance converges while the dimensionality of $z > 16$. It is much smaller than a conventional setting (i.e., 100) for the other generation tasks such as image and text generation. This suggests that the variety of realistic routes is smaller than images or texts, i.e., given a RPQ, there may exist a relatively small number of acceptable routes from users compared with image generation, e.g., given an image class, says "dog", to generate different realistic images for "dog". On the other hand, Figure 10(h) shows that setting H_{time} between 64 to 256 achieves good performance.

D VISUALIZED GENERATED PATHS

In this section, we investigate the generated paths of ProgRPGAN by visualized case studies. Figure 11 illustrates two example queries RPQ 1 and RPQ 2 (where RPQ 1 is the same as we show in Section 1), both of which have a variety of acceptable routes from road users (i.e., blue lines) between the source and destination, as shown in Figure 11(a) and (c). Then, we use ProgPRGAN to generate sets of paths for the both queries by giving random noise vectors z. The

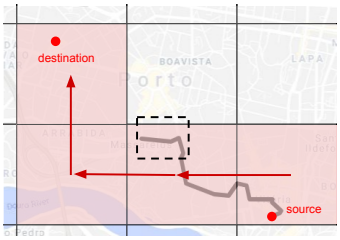


Figure 12: Generated path in the $K \times K$ grid map

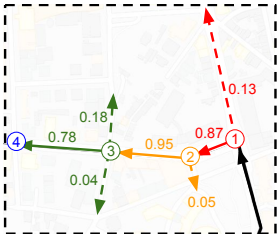


Figure 13: Next road segment prediction

generated paths by ProgRPGAN are shown in Figure 11(b) and (d) (for RPQ 1 and RPQ 2, respectively), distinguished by color that generated paths which are exactly the same have the same color. As shown, ProgRPGAN is effective to generate realistic paths from the given source to destination compared with the real routes from road users. Moreover, ProgRPGAN is able to generate a variety of realistic paths for the same query. It is worthy to note that the distribution of generated paths is correlated to the frequency of the real routes taken by users (i.e., the darker lines indicate the higher frequency in Figure 11(a) and (c)). However, we can still observe that some real routes do not appear in the set of generated paths (marked as "outliers?" and "missing paths?" in Figure 11). ProgRPGAN may not have learned to generate these missing routes due to the low frequency.

E ILLUSTRATION OF NEXT ROAD SEGMENT PREDICTION

In this section, we further investigate the path generation process of ProgRPGAN in the road network by a visualized case study. Given the RPQ 1 in Appendix D as the query (its source and destination are labeled in Figure 12) and a generated cell path in the highest-resolution grid map (illustrated as red squares in Figure 12), ProgRPGAN aims to generate a path on the road network by sequentially predicting the next intersection to move toward the destination. Assume that ProgRPGAN already generates a partial path starting from the source (i.e, the black path in Figure 12), we illustrate the further path generation process within the dotted box of Figure 12 in Figure 13. As shown in Figure 13, at intersection 1, ProgRPGAN infers a transition probability to intersection 2 (i.e., 0.87) higher than 0.13 to another intersection up north Thus ProgRPGAN selects intersection 2 as the next step. Note that, while selecting the intersection up north of intersection 1 is also a potential good decision to generate a realistic path (as the blue path shown in Figure 11(b)), the current generation is guided by the given cell path that the next cell to reach is the one on left of the current intersection 1, and thus ProgRPGAN correctly infers intersection 2 as the next intersection to move to. Then, ProRPGAN sequentially to generate the path to move to intersection 3 and then intersection 4, following the direction of the generated cell path.