Empowering A* Search Algorithms with Neural Networks for Personalized Route Recommendation

Jingyuan Wang*, Ning Wu*
Beijing Advanced Innovation Center for BDBC, School of Computer Science and Engineering, Beihang University, Beijing, China
{jywang,wuning}@buaa.edu.cn

Wayne Xin Zhao†
School of Information, Renmin University of China
Beijing, China
batmanfly@gmail.com

Fanzhang Peng, Xin Lin
MOE Engineering Research Center of ACAT, School of Computer Science and Engineering, Beihang University
Beijing, China
{pengfanzhang,sweeneylin}@buaa.edu.cn

ABSTRACT

Personalized Route Recommendation (PRR) aims to generate user-specific route suggestions in response to users' route queries. Early studies cast the PRR task as a pathfinding problem on graphs, and adopt adapted search algorithms by integrating heuristic strategies. Although these methods are effective to some extent, they require setting the cost functions with heuristics. In addition, it is difficult to utilize useful context information in the search procedure. To address these issues, we propose using neural networks to automatically learn the cost functions of a classic heuristic algorithm, namely A^* algorithm, for the PRR task. Our model consists of two components. First, we employ attention-based Recurrent Neural Networks (RNN) to model the cost from the source to the candidate location by incorporating useful context information. Instead of learning a single cost value, the RNN component is able to learn a time-varying vectorized representation for the moving state of a user. Second, we propose to use a value network for estimating the cost from a candidate location to the destination. For capturing structural characteristics, the value network is built on top of improved graph attention networks by incorporating the moving state of a user and other context information. The two components are integrated in a principled way for deriving a more accurate cost of a candidate location. Extensive experiment results on three real-world datasets have shown the effectiveness and robustness of the proposed model.

CCS CONCEPTS

• **Information systems → Information retrieval**.

KEYWORDS

Route Recommendation, A^* Search, Neural Networks, Attention, Personalized Recommendation

*Both authors contributed equally to this work.
†Corresponding author.

KDD '19, August 4–8, 2019, Anchorage, AK, USA

ACM ISBN 978-1-4503-6201-6/19/08…$15.00
https://doi.org/10.1145/3292500.3330824

ACM Reference Format:
Jingyuan Wang*, Ning Wu, Wayne Xin Zhao, and Fanzhang Peng, Xin Lin. 2019. Empowering A* Search Algorithms with Neural Networks for Personalized Route Recommendation. In *The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '19), August 4–8, 2019, Anchorage, AK, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3292500.3330824

1 INTRODUCTION

With the popularization of GPS-enabled mobile devices, a huge volume of trajectory data from users has become available in a variety of domains [27–29]. *Personalized Route Recommendation (PRR)* is one of the core functions in many online location-based applications, *e.g.,* online map. Given the road network, PRR aims to generate user-specific route suggestions on instant queries about the path planing from a source to a destination [6, 7]. It is challenging to perform effective pathfinding in a large and complex road network. For accurate route recommendation, it is necessary to consider rich context information, including personalized preference, spatial-temporal influence and road network constraint.

Early studies cast the route recommendation task as a pathfinding problem on graphs [17, 30]. These methods mainly focus on how to extend existing search algorithms (*e.g.,* Dijkstra shortest path algorithms and A^* search algorithm) for the studied task. With suitable heuristics, they can substantially reduce the search space and obtain high-quality responses. The key of heuristic search algorithms is to develop an effective cost function. Most of previous studies heuristically set the cost function, making their applicability highly limited. In addition, it is difficult to utilize various kinds of context information in the search process. To construct more flexible approaches, many studies have utilized machine learning methods for solving the PRR task [4, 32]. These methods are able to characterize the location dependencies or spatial-temporal information with principled models. While, most of them are shallow computational models, and may have difficulties in capturing complex trajectory patterns. With the revival of deep learning, it sheds light on the development of more effective PRR models using neural networks. Especially, sequential neural models, *i.e.,* Recurrent Neural Networks (RNN), have been widely used for modeling sequential trajectory data [1, 31, 34]. However, to our knowledge, these models mainly focus on one-step or short-term location prediction, which may not be suitable for the PRR task.

Comparing the above approaches, we can see they have their own merits for the PRR task. On one hand, in terms of the problem

setting, heuristic search algorithms are specially suitable for the PRR task, which can be considered as a pathfinding problem on graphs given the source and destination. They are able to generate high-quality approximate solutions using elaborate heuristics. On the other hand, as a newly emerging direction of machine learning, deep learning methods are effective to capture the complex data characteristics using learnable neural networks. They are able to learn effective mapping mechanisms from input to output or expressive feature representations from raw data in an automatic way. For developing a more effective PRR method, is there a principled way to combine the merits of both kinds of approaches?

Inspired by recent progress of deep learning in strategy-based games (*e.g.,* Go and Atari) [18, 23] , we propose to improve search algorithms with neural networks for solving the PRR task. Especially, we adopt the A^* algorithm [10] as the base search algorithm, since it has been widely used in pathfinding and graph traversal. Previous studies have also shown that A^* algorithm is a promising approach to solving the route recommendation task [11, 20, 30]. The main idea of our solution is to automatically learn the cost functions in A^* algorithms, which is the key of heuristic search algorithms. For this purpose, there are three important issues to consider. First, we need to define a suitable form for the cost in the PRR task. Different from traditional graph search problems, a simple heuristic form can not directly optimize the goal of our task [11, 30], *e.g.,* the route based on the shortest distance may not meet the personalized needs of a specific user. Second, we need to design effective models for implementing cost functions with different purposes, and unify different cost functions for deriving the final cost. The entire cost function $f(\cdot)$ of A^* can be decomposed into two parts, *i.e.,* $f(\cdot) = g(\cdot) + h(\cdot)$. The two parts compute the *observable cost* from the source node to the evaluation node and the *estimated cost* from the evaluation node to the destination node respectively. Intuitively, the two parts require different modeling methods, and need to jointly work to compute the entire cost. Third, we need to utilize rich context or constraint information for improving the task performance. For example, spatial-temporal influence and road networks are important to consider in modeling trajectory patterns, and should be utilized to develop the cost functions.

To address these difficulties, we propose a novel neuralized A^* search algorithm for solving the PRR task. To define a suitable form for the search cost, we formulate the PRR task as a conditional probability ranking problem, and compute the cost by summing the negative log of conditional probabilities for each trajectory point in a candidate trajectory. We use this form of cost to instruct the learning of the two cost functions in A^* algorithm, namely $g(\cdot)$ and $h(\cdot)$. For implementing $g(\cdot)$, we propose to use attention-based RNNs to model the trajectory from the source location to the candidate location. We incorporate useful context information to better capture sequential trajectory behaviors, including spatial-temporal information, personalized preference and road network constraint. Instead of simply computing a single cost, our model also learns a time-varying vectorized representation for the moving state of a user. For learning $h(\cdot)$, we propose to use a value network for estimating the cost for unobserved part of a trajectory. In order to capture the complex characteristics of road networks, we build the value network on top of improved graph attention networks by incorporating useful context information. In these two different components, we share the same embeddings for locations, users and time. The learned moving state in the RNN component will be subsequently fed into the value network. The two components are integrated in a joint model for computing a more accurate cost of a candidate location. Since the estimated cost is associated with a multi-step decision process, we further propose to use the *Temporal Difference* method from Reinforcement Learning for model learning.

To the best of our knowledge, we are the first to use neural networks for improving A^* algorithm in the PRR task. Our approach is able to automatically learn the cost functions without handcrafting heuristics. It is able to effectively utilize context information and characterize complex trajectory characteristics, which elegantly combines the merits of A^* search algorithms and deep learning. The two components are integrated in a joint model for deriving the evaluation cost. Extensive results on the three datasets have shown the effectiveness and robustness of the proposed model.

2 RELATED WORK

Our work is related to the following research directions.

Route Recommendation. With the availability of user-generated trajectory information, route recommendation has received much attention from the research community [6, 7, 11], which aims to generate reachable paths between the source and destination locations. The task can be defined as either *personalized* [6, 7] or *non-personalized* [4, 11, 17, 35], and constructed based on different types of trajectory data, *e.g.,* GPS data [35] or POI check-in data [3, 22]. In the literature, various methods have been developed for route recommendation, including graph search algorithms [4, 15, 30], time-sensitive algorithms [17], A^* search algorithm [11], probabilistic POI transition/ranking models [3] and diver-direction based methods [35]. Overall, most of the studies focus on using search based algorithms or probabilistic models by considering additional constraints, *e.g.,* road networks or time. Our work is built on top of search based solutions, and the novelty lies in the automatic learning of the cost functions using neural networks. Our model is flexible to incorporate rich context or constraint information.

Deep Learning for Trajectory Data Mining. Recent years have witnessed the success of deep learning in modeling complex data relations or characteristics. In specific, Recurrent Neural Network (RNN) together with its variant Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) have been widely used for modeling sequential trajectory data. Typical works include hierarchical RNN [36], RNN with road network constraints [31], and multi-modal embedding RNN [9], spatial-temporal RNN [16] and space time feature-based RNN [1]. These studies mainly focus on short-term trajectory behaviors, *e.g.,* one-step location recommendation [16], which are not suitable for solving the current task.

Machine Learning for Heuristic Search. These studies in this direction aim to automatically improve or optimize the search algorithms with machine learning methods. Early works include the use of machine learning in creating effective, likely-admissible or improved heuristics [8, 13, 21]. More recently, deep learning has significantly pushed forward the research of this line. The main idea is to leverage the powerful modeling capacity of neural networks for

improving the tasks that require complicated solving strategies, including the Go game [23] and Atari games [18]. Our work is highly inspired by these pioneering works, but have a quite different focus on the studied task, *i.e.,* personalized route recommendation. Our task itself involves specific research challenges that make the reuse of previous works impossible.

3 PRELIMINARIES

In our task, we assume road network information is available for the pathfinding task, which is the foundation of the traffic communication for users.

Definition 1. ***Road Network.*** *A road network is a directed graph $\mathcal{G} = (\mathcal{L}, \mathcal{E})$, where $\mathcal{L}$ is a vertex set of locations and $\mathcal{E} \subset \mathcal{L} \times \mathcal{L}$ is an edge set of road segments. A vertex $l_i \in \mathcal{L}$ (*i.e., *a location) represents a road junction or a road end. An edge $e_{l_i,l_j} = \langle l_i, l_j \rangle \in \mathcal{E}$ represents a directed road segment from vertex l_i to vertex l_j.*

Definition 2. ***Route.*** *A route (*a.k.a., *a path) p is an ordered sequence of locations connecting the source location l_s with the destination location l_d with m intermediate locations,* i.e., *$p : l_s \rightarrow l_1 \rightarrow \ldots \rightarrow l_m \rightarrow l_d$, where each pair of consecutive locations $\langle l_i, l_{i+1} \rangle$ corresponds to a road segment $e_{l_i,l_{i+1}}$ in the road network.*

The moving trajectory of a user on the road network can be recorded using GPS-enabled devices. Due to instrumental inaccuracies, the sampled trajectory points may not be well aligned with the locations in $\mathcal{L}$. Following [33], we can preform the procedure of *map matching* for aligning trajectory points with locations in $\mathcal{L}$.

Definition 3. ***Trajectory.*** *A trajectory t is a time-ordered sequence of m locations (after map matching) generated by a user,* i.e., *$t : \langle l_1, b_1 \rangle \rightarrow \langle l_2, b_2 \rangle \rightarrow \ldots \rightarrow \langle l_m, b_m \rangle$, where b_i is the visit timestamp for location l_i.*

A trajectory is a user-generated location sequence with timestamps, while a route is pre-determined by the road network. For a route, the start and end points are important to consider, which correspond to the source and destination of a travel. In the task of PRR, a user can issue *travel queries.*

Definition 4. ***Query.*** *A query q is a triple $\langle l_s, l_d, b \rangle$ consisting of source location l_s, destination location l_d and departure time b.*

With the above definitions, we now define the studied task.

Definition 5. ***Personalized Route Recommendation (PRR).*** *Given a dataset $\mathcal{D}$ consisting of historical trajectories, for a query $q : \langle l_s, l_d, b \rangle$ from user $u \in \mathcal{U}$, we would like to infer the most possible route p^* from l_s to l_d made by user u, formally defined as solving the optimal path with the highest conditional probability:*

$$p^* = \arg\max_p Pr(p|q, u, \mathcal{D}). \tag{1}$$

The PRR task is formulated as a conditional ranking problem. For solving this task, we first present a traditional A^*-based algorithm in Section 4, and then present our proposed approach in Section 5.

4 A HEURISTIC A^* SOLUTION FOR PRR

The task of PRR can be framed as a graph-based search problem. In this setting, we view the road network as a graph, and study how to find possible route(s) that start from the source node and end at the destination node.

Review of A^* Algorithm. In the literature [10], A^* search algorithm is widely used in pathfinding and graph traversal due to its performance and accuracy. Starting from a source node of a graph, it aims to find a path to the given destination node resulting in the smallest *cost*. It maintains a tree of paths originating at the source node and extending those paths one edge at a time until its termination criterion is satisfied. At each extension, A^* evaluates a candidate node n based on a *cost function* $f(n)$

$$f(n) = g(n) + h(n), \tag{2}$$

where $g(n)$ is the cost of the path from the source to n (we call it *observable cost* since the path is observable), and $h(n)$ is an estimate of the cost required to extend the future path to the goal (we call it *estimated cost* since the actual optimal path is unknown). The key part of A^* is the setting of the heuristic function $h(\cdot)$, which has an important impact on the final performance.

A Simple A^*-based Approach for PRR. Considering our task, the goal is to maximize the conditional probability of $\Pr(p|q, u, \mathcal{D})$. We can equally minimize its negative log: $-\log \Pr(p|q, u, \mathcal{D})$. Given a possible path $p : l_s \rightarrow l_1 \rightarrow l_2 \cdots \rightarrow l_m \rightarrow l_d$, consisting of m intermediate locations, we can factorize the path to compute its cost according to the chain rule in probability in the form of

$$-\log \Pr(p|q, u, \mathcal{D}) = -\sum_{i=0}^{m} \log \Pr(l_{i+1}|l_s \rightarrow l_i, q, u), \tag{3}$$

where $l_0 = l_s$ and $l_{m+1} = l_d$, and $\mathcal{D}$ is dropped for simplifying notations. This formula motivates us to set the cost functions of A^* algorithm in a similar form. Assume a partial route has been generated, *i.e.,* $p : l_s \rightarrow l_1 \cdots \rightarrow l_{i-1}$, we can compute the observable cost of a candidate l_i for extension as

$$g(l_s \rightarrow l_i) = -\sum_{k=1}^{i-1} \log \Pr(l_{k+1}|l_s \rightarrow l_k, q, u). \tag{4}$$

To compute the conditional transition probabilities, the first-order Markov assumption is usually adopted, so we have $\Pr(l_{k+1}|l_s \rightarrow l_k, q, u) = \Pr(l_{k+1}|l_k, q, u)$. Following [4, 30], we can further use user-specific or overall transition statistics to estimate the probabilities (with smoothing). While, to compute the cost of $h(l_i \rightarrow l_d)$ is more difficult, since the optimal sub-route from l_i to l_d is unknown. We cannot directly apply the similar method in Eq. (4) for $h(\cdot)$. In practice, we can use different heuristics to set $h(\cdot)$, including the shortest spatial distance [10, 19] and the historical likelihood [30].

Analysis. For our task, the A^*-based approach is more appealing than a greedy best-first search algorithm. By decomposing the entire cost into two parts, it leaves room on the elaborated setting of $g(\cdot)$ and $h(\cdot)$ for different tasks. Although it has been shown that A^*-like algorithms perform well in the task of route recommendation [11, 19, 30], we see three weak points for improvement. First, A^* algorithm is a general framework in which cost functions have to be heuristically set. It is difficult to incorporate varying context information, *e.g.,* personalized preference and spatial-temporal influence. Second, the cost function usually relies on the heuristic computation or estimation, which is easy to suffer from data

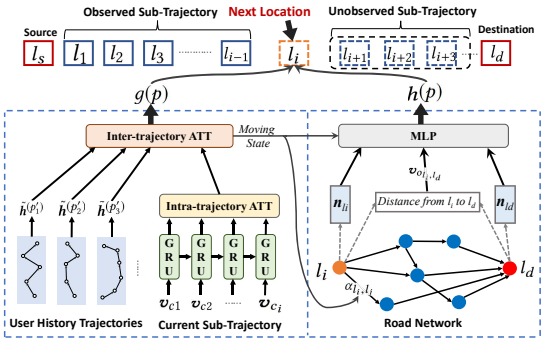


Figure 1: The overall architecture of the NASR model. $g(\cdot)$ learns the cost from the source to a candidate location, called *observable cost*; $h(\cdot)$ predicts the estimated cost from a candidate location to the destination, called *estimated cost.*

sparsity. For example, the estimation of transition probabilities in Eq. (4) may not be accurate when the historical transitions between two locations are sparse. In this case, even the computation of observable cost $g(\cdot)$ is likely to be problematic. Third, the PRR task is challenging, and a simple heuristic search strategy may not be capable of performing effective pathfinding in practice, *e.g.,* the route that has the shortest spatial distance may not be the final choice of a user [4, 17]. With these considerations, we next present our solution for addressing the above difficulties of A^* in PRR.

5 THE NASR MODEL

In the section, we present the proposed *Neuralized A-Star based personalized route Recommendation (NASR)* model.

5.1 Model Overview

Our model is developed based on the general A^* algorithm framework. For node evaluation, we decompose the entire cost function $f(\cdot)$ into two parts, namely *observable cost* and *estimated cost*, which correspond to the cost functions $g(\cdot)$ and $h(\cdot)$. Traditionally, both $g(\cdot)$ and $h(\cdot)$ are heuristically computed or set. While, our idea is to automatically learn the two functions with neural networks instead of using heuristics. Specially, we use Recurrent Neural Networks (RNN) to implement $g(\cdot)$ and another value network to implement $h(\cdot)$. In our neural network for function $g(\cdot)$, we not only compute a single cost value, but also learn a time-varying *moving state* for a specific user. The moving state encodes necessary trajectory information of a user till the evaluation time, which will be fed into the computation of $h(\cdot)$. Once the two networks are learned, we can compute the cost of a candidate location for path extension. We present the overall architecture for the proposed model in Fig. 1.

5.2 Modeling the Observable Cost with RNN

This part studies the learning of function $g(\cdot)$ for observable cost. Given an observed sub-route $l_s \rightarrow l_1 \rightarrow l_2 \cdots \rightarrow l_i$, as shown in Eq. (4), the problem becomes how to effectively learn the conditional transition probabilities $\Pr(l_{k+1}|l_s \rightarrow l_k, q, u)$. Simple frequency-based estimation method will suffer from data sparsity in large search space even with first-order Markov assumption. In this case, the computed observable cost will not be reliable to be used. In addition, PRR is a user-centric task, and a single cost value may not be enough to describe what has been observed. Instead, the *moving state* of a user and useful context information should be considered. To address these difficulties, we propose to use Recurrent Neural Networks to implement the cost function $g(\cdot)$.

5.2.1 Embedding Rich Context Information. As the prerequisite module, we embed rich context information into dense vectors, which will be subsequently used by other components. First, we set up an embedding vector $\boldsymbol{v}_u \in \mathbb{R}^{K_U}$ for user u, encoding necessary personalized user information. Then, for each location $l \in \mathcal{L}$, we set up a corresponding embedding vector $\boldsymbol{v}_l \in \mathbb{R}^{K_L}$. For trajectory behaviors, temporal information is also important to consider. Following [9], for each visit timestamp b, we use two embedding vectors $\boldsymbol{v}_{di(b)} \in \mathbb{R}^{K_D}$ and $\boldsymbol{v}_{hi(b)} \in \mathbb{R}^{K_H}$, where $di(b)$ and $hi(b)$ are functions transforming b into corresponding weekday index (1 to 7) and hour index (1 to 24) respectively. At i-th time step, we concatenate the above embedding vectors into a single embedding vector, and form an enhanced representation of context x_i as

$$\boldsymbol{v}_{x_i} = \boldsymbol{v}_u \| \boldsymbol{v}_{l_i} \| \boldsymbol{v}_{di(b_i)} \| \boldsymbol{v}_{hi(b_i)}, \tag{5}$$

where "||" is the concatenation operation. We can see that $\boldsymbol{v}_{x_i}$ contains information for user preference, current location and time.

5.2.2 Encoding the Observed Sub-Trajectory with RNN. For the PRR task, it is important to model the trajectory characteristics of users' moving behaviors, which can be considered as a sequential process. We utilize RNNs to model such sequential behaviors. Given an observed sub-trajectory $p : l_s \rightarrow l_1 \cdots \rightarrow l_i$ generated by u, we employ the widely used GRU network [5] to encode it into a vector

$$\boldsymbol{h}_i^{(p)} = \text{GRU}(\boldsymbol{v}_{x_i}, \boldsymbol{h}_{i-1}^{(p)}), \tag{6}$$

where $\boldsymbol{h}_i^{(p)} \in \mathbb{R}^{K_R}$ is the hidden vector produced by the GRU network and $\boldsymbol{v}_{x_i}$ is the context vector defined in Eq. (5). The vector $\boldsymbol{h}_i^{(p)}$ encodes the *moving state* of a user at the i-th time step. Note that we use the superscript to index the trajectory and the subscript to index locations. Unlike traditional A^* search algorithm, here, we learn an informative state representation of a user at each step, providing more information than a single cost value.

5.2.3 Enhanced Moving States with Attention Mechanism. An observed sub-trajectory can be short and noisy. We further propose to use two types of attention to improve the learning of moving state by leveraging data dependence.

Intra-Trajectory Attention. We first apply the method [2] to compute the attention between locations in the same trajectory as

$$\tilde{\boldsymbol{h}}_i^{(p)} = \sum_{k=1}^{i} \text{att}\left(\boldsymbol{h}_i^{(p)}, \boldsymbol{h}_k^{(p)}\right) \cdot \boldsymbol{h}_k^{(p)}, \tag{7}$$

where $\tilde{\boldsymbol{h}}_i^{(p)}$ denotes the improved state representation with intra-trajectory attention and $\text{att}(\cdot,\cdot)$ is an attention function as

$$\begin{aligned} \text{att}\left(\boldsymbol{h}_i^{(p)}, \boldsymbol{h}_k^{(p)}\right) &= \frac{\alpha_{i,k}}{\sum_{k'=1}^{i} \exp(\alpha_{i,k'})}, \\ \alpha_{i,k} &= \boldsymbol{w}_1^\top \cdot \tanh\left(\boldsymbol{W}_1 \cdot \boldsymbol{h}_k^{(p)} + \boldsymbol{W}_2 \cdot \boldsymbol{h}_i^{(p)}\right), \end{aligned} \tag{8}$$

where $\boldsymbol{w}_1$, $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ are the parameter vector or matrices to learn. With intra-trajectory attention, we can discover more important

characteristics by considering the entire trajectory. After intra-trajectory attention, we use the state representation of the last location for encoding the entire sub-trajectory, *i.e.,* $\tilde{\boldsymbol{h}}^{(p)} = \tilde{\boldsymbol{h}}_i^{(p)}$.

Inter-Trajectory Attention. The information from a single trajectory is usually limited. In order to capture overall moving patterns for a specific user, we further consider incorporating historical trajectories generated by the user. Given the current trajectory p, we attend it to each of the other historical trajectories as

$$\boldsymbol{h}^{(p)} = \sum_{p' \in \mathcal{P}^u} \text{att}\left(\tilde{\boldsymbol{h}}^{(p)}, \tilde{\boldsymbol{h}}^{(p')}\right) \cdot \tilde{\boldsymbol{h}}^{(p')}, \tag{9}$$

where $\mathcal{P}^u$ denotes the set of historical trajectories generated by u and $\text{att}(\cdot,\cdot)$ is the attention function which is similar to Eq. (8) but with different learnable parameters.

5.2.4 Observable Cost Computation with the Road Network Constraints. Once we have learned the hidden state for the current timestamp, we are able to compute the probability of the next location using a softmax function with road network constraint as

$$\Pr(l_i | l_s \to l_{i-1}, q, u) = \frac{\exp\left(z(p^{l_s \to l_i})\right)}{\sum_{l' \in \mathcal{L}_{l_{i-1}}} \exp\left(z(p^{l_s \to l'})\right)}, \tag{10}$$

where $z(p) = \boldsymbol{w}_2^\top \cdot \boldsymbol{h}^{(p)}$ is a linear transformation function taking as input the hidden state learned for a trajectory p in Eq. (9), and $\boldsymbol{w}_2$ is parameter vector to learn. Here, we compute the probability of a candidate location l_i by normalizing over all the neighboring locations of l_{i-1} in the road network. After defining $\Pr(l_i | l_s \to l_{i-1}, q, u)$ in Eq. (10), we sum the negative log probability of each location in a trajectory $l_s \to l_i$ as the value of $g(\cdot)$

$$g(l_s \to l_i) = -\sum_{j=2}^{i} \log \Pr\left(l_j | l_s \to l_{j-1}, q, u\right). \tag{11}$$

Note that we do not set $g(l_s \to l_i)$ using simple distance functions, since we would like to learn more useful information from the observed trajectories. Typically, a user would select a route based on many factors. Our computation form for $g(\cdot)$ naturally fits the defined goal of our task in Eq. (1). To learn the neural network component, given $g(\cdot)$ in Eq. (11), we set a loss for all the observed trajectories over all users

$$Loss_1 = \sum_{u \in \mathcal{U}} \sum_{p \in \mathcal{P}^u} g(p). \tag{12}$$

5.3 Modeling the Estimated Cost with Value Networks

Besides the observable cost, we need to learn the estimated cost from a candidate location to the destination. Specially, we introduce a value network to implement $h(\cdot)$. This part is more difficult to model since no explicit trajectory information is observed. In order to better utilize the road network information for estimation, we build the value network on top of an improved graph attention network with useful context information.

5.3.1 Improved Graph Attention Networks for Road Networks. We consider using graph neural networks for learning effective structural representations for summarizing graph nodes. Generally, the update of graph neural networks [26] can be given as

$$\boldsymbol{N}^{(z+1)} = \text{GNN}\left(\boldsymbol{N}^{(z)}\right) \tag{13}$$

where $\boldsymbol{N}^{(z)} \in \mathbb{R}^{K_G \times |\mathcal{L}|}$ denotes the matrix consisting of node representations at the z-th iteration, and the l_k-th column $\boldsymbol{n}_{l_k} \in \mathbb{R}^{K_G}$ corresponds to the representation of node l_k, *i.e.,* location $l_k \in \mathcal{L}$. For initialization, we set $\boldsymbol{n}_{l_k}^{(0)} = \boldsymbol{v}_{l_k}$ with the learned location embeddings in Section 5.2.1. Here, we adopt the recently proposed Graph ATTention network (GAT) [26] for a good balance between capacity and efficiency. In GAT, the key part is to compute the attention importance of a node on another. Original attention scores are computed using the node representations alone, which cannot well adapt to our task.

Context-aware Graph Attention. Intuitively, a node has a larger impact on a nearby node than a faraway node. For modeling this factor, we first discretize the distance o_{l_i,l_j} between nodes l_i and l_j into consecutive value bins. Then we set a unique embedding $\boldsymbol{v}_o \in \mathbb{R}^{K_O}$ for each discretized distance value o. Besides, it is likely the previous part of a trajectory will affect the subsequent part. Recall that we use RNN to encode observable sub-trajectory $p : l_s \to l_i$ into an embedding vector $\boldsymbol{h}^{(p)}$, modeling the moving state of a user. The attention scores should take current moving state of a user into consideration. With the two factors considered, we compute the attention weights between two locations l_j and $l_{j'}$ as

$$\alpha_{l_j,l_{j'}} = \frac{\exp\left(\boldsymbol{w}_2^\top \cdot \left(\boldsymbol{W}_3 \boldsymbol{n}_{l_j} + \boldsymbol{W}_4 \boldsymbol{n}_{l_{j'}} + \boldsymbol{W}_5 \boldsymbol{h}^{(p)} + \boldsymbol{W}_6 \boldsymbol{v}_{o_{l_j,l_{j'}}}\right)\right)}{\sum_{k \in \mathcal{L}_{l_j}} \exp\left(\boldsymbol{w}_2^\top \cdot \left(\boldsymbol{W}_3 \boldsymbol{n}_{l_j} + \boldsymbol{W}_4 \boldsymbol{n}_{l_k} + \boldsymbol{W}_5 \boldsymbol{h}^{(p)} + \boldsymbol{W}_6 \boldsymbol{v}_{o_{l_j,l_k}}\right)\right)}, \tag{14}$$

where $\boldsymbol{W}_{(\cdot)}$ and $\boldsymbol{w}_2$ are learnable parameters, and $\boldsymbol{v}_{o_{l_j,l_{j'}}}$ is the embedding vector for the discretized distance value. Since our attention mechanism involves more kinds of information, we use the multi-head attention for stabilizing the learning process. We combine the results of A attention heads as

$$\boldsymbol{n}_{l_i}^{(z+1)} = \Big\|_{a=1}^{A} \text{relu}\left(\sum_{l_j \in \mathcal{L}_{l_i}} \alpha_{l_i,l_j}^{(a)} \boldsymbol{W}^{(a)} \boldsymbol{n}_{l_j}^{(z)}\right), \tag{15}$$

where $\alpha_{i,j}^{(a)}$ are the normalized attention scores computed by the a-th attention head, "$\|$" denotes the concatenation operation and $\boldsymbol{W}^{(a)}$ is weight matrix of the corresponding input linear transformation.

Predicting the Estimated Cost with MLP. After obtaining the node representations, we are ready to define the cost function for estimating future cost. We use a Multi-Layer Perceptron component to infer the cost from the candidate location l_i to the destination l_d. Formally, we have

$$h\left(l_i \to l_d\right) = \text{MLP}\left(\boldsymbol{h}^{(p)}, \boldsymbol{n}_{l_i}, \boldsymbol{n}_{l_d}, \boldsymbol{v}_{o_{l_i,l_d}}\right), \tag{16}$$

where the MLP component takes as input the moving state $\boldsymbol{h}^{(p)}$, the node representations $\boldsymbol{n}_{l_i}$ and $\boldsymbol{n}_{l_d}$ for locations of l_i and l_d, and the embedding $\boldsymbol{v}_{o_{l_i,l_d}}$ for their spatial distance.

5.3.2 Temporal Difference Learning for the Estimated Cost. The computation of $h(l_i \to l_d)$ relies on the optimal sub-route from l_i to l_d, which is a multi-step decision process and difficult to be directly

optimized. Inspired by recent progress in reinforcement learning, we follow the framework of Markov Decision Process (MDP) [25] for solving our problem. In an MDP, an agent behaves in an environment according to a policy that specifies how the agent selects actions at each state of the MDP. In our task, a state consists of the information of the query q and the current sub-sequence $l_s \rightarrow l_{i-1}$; an action is to select a location l_i to extend in the route. The standard MDP aims to maximize the future reward, while our task aims to minimize the future cost. Mathematically, maximizing the future reward is equal to minimizing the future cost for the PRR task. To be consistent with our task setting, in what follows, we model the *cost* instead of *reward*. When a location l_i is selected, an immediate cost c_i will be yielded according to

$$c_i = -\log \Pr(l_i | l_s \rightarrow l_{i-1}, q, u), \tag{17}$$

where $\Pr(l_i | l_1 \rightarrow l_{i-1}, q, u)$ is the probability computed in Eq. (10).

Since our purpose is to estimate the future cost, we do not need to explicitly learn the policy here. Hence, we adopt a popular value-based learning method for optimizing the value networks, *i.e., Temporal Difference (TD).* TD [24] is an approach to learning how to predict a quantity that depends on future values of a given signal. In our model, the estimated cost $h(l_i \rightarrow l_d)$ and the future costs in $l_i \rightarrow l_d$ have a relationship as

$$h(l_i \rightarrow l_d) = \sum_{j=i+1}^{T} \gamma^{j-i-1} c_j, \tag{18}$$

where γ is discount rate to discounting future cost to the current and T is the timestamp arriving at l_d. If we look forward n steps from the current location l_i, the prediction error of the value network is

$$y_{l_i} = \gamma^n h(l_{i+n} \rightarrow l_d) + \sum_{j=i+1}^{i+n} \gamma^{j-i-1} c_{l_j}, \tag{19}$$

$$\delta_{l_i} = \sum_{i=1}^{T-1} \left\| h(l_i \rightarrow l_d) - y_{l_i} \right\|^2, \tag{20}$$

where the term y_{l_i} is the estimated cost using the temporal difference approach. To learn the value network component, we set a loss for all the observed trajectories over all users as

$$Loss_2 = \sum_{u \in \mathcal{U}} \sum_{p \in \mathcal{P}^u} \sum_{l_i \in p} \delta_{l_i}. \tag{21}$$

5.4 Model Analysis and Learning

Integrating the two components in Section 5.2 and 5.3, we obtain the complete NASR model for the PRR task. NASR follows the similar search procedure of A^* algorithm but uses the learned cost for node evaluation.Specially, it has fulfilled the cost functions of A^* algorithms with neural networks, namely $g(\cdot)$ and $h(\cdot)$. Given a candidate location, the first component utilizes RNNs to characterize the currently generated sub-trajectory for learning *observable cost*, while the second component incorporates a value network to predict the *estimated cost* to arrive at the destination. Finally, the two cost values are summed as the final evaluation cost of a candidate location.

Compared with traditional heuristic search algorithms, NASR has the following merits. First, it does not require to manually set functions with heuristics, but automatically learns the functions from data. Second, it can utilize various kinds of context information and capture more complicated personalized trajectory characteristics. Third, it is able to coordinate and integrate the two components by sharing useful information or parameters in a principled way. Note that traditional search algorithms neglect the importance of $g(\cdot)$, which computes the cost of observed sub-trajectories. In our model, the implementation of $g(\cdot)$ not only learns the cost but also a vectorized user state representation, *i.e.,* the moving state of a user. This state vector is subsequently used for the learning of $h(\cdot)$ function by providing useful information from current sub-trajectory. Besides, as we discussed in Section 3, not all the observable cost can be directly computed, usually requiring estimation or approximation. Neural networks are helpful to improve the computation of $g(\cdot)$ by producing more robust results.

To learn the model parameters, we first pre-train the RNN component. Then, we jointly learn the two components using alternative optimization by iterating over the trajectories in training set. After model learning, we follow the search procedure of A^* algorithm for the PRR task with the evaluation cost computed by our model.

6 EXPERIMENTS

6.1 Experimental Setup

Construction of the Datasets. To measure the performance of our proposed model, we use three real-world trajectory datasets. The *Beijing taxi* trajectory data is sampled every minute, while the *Beijing bicycle* dataset is sampled every 10 seconds. The *Porto taxi* dataset is originally released for a Kaggle trajectory prediction competition with a sampling period of 15 seconds. For the three datasets, we collect corresponding road network information from open street map.We further perform map matching [33] by aligning GPS points with locations in the road network. In this way, we transform the trajectory data into timestamped location sequences. With the boundary indicators provided by the three datasets, we split the location sequence into multiple trajectories.

Evaluation Metrics. For the PRR task, we adopt a variety of evaluation metrics widely used in previous works [6, 12, 14]. Given an actual route p, we predict a possible route p' with the same source and destination. Following [6, 14], we use *Precision*, *Recall* and *F1-score* as evaluation metrics: $Precision = \frac{|p \cap p'|}{|p'|}$, $Recall = \frac{|p \cap p'|}{|p|}$ and $F1 = \frac{2*P*R}{P+R}$. *Precision* and *Recall* compute the ratios of overlapping locations *w.r.t.* the actual and predicted routes respectively. Besides, we use the *Edit distance* as a fourth measure [12], which is the minimum number of edit operations required to transform the predicted route into the actual route. Note the source and destination locations are excluded in computing evaluation metrics.

Task Setting. For each user, we divide her/his trajectories into three parts with a ratio of 7 : 1 : 2, namely training set, validation set and test set. We train the model with training set, and optimize the model with validation set. Instead of reporting the overall performance on all test trajectories, we generate three types of queries *w.r.t.* the number of locations in the trajectories, namely *short* (10 to 20 locations), *medium* (20 to 30 locations) and *long* (more than 30 locations). In test set, given a trajectory, the first and last locations are treated as the source and destination respectively, and the

Table 1: Performance comparison using four metrics on three datasets. All the results are better with larger values except the EDT measure. With paired t-test, the improvement of the NASR over all the baselines is significant at the level of 0.01.

Datasets	Metric	Precision						Recall					
	Length	RICK	MPR	CTRR	STRNN	DeepMove	NASR	RICK	MPR	CTRR	STRNN	DeepMove	NASR
Beijing Taxi	Short	0.712	0.347	0.558	0.491	0.742	**0.821**	0.723	0.372	0.164	0.384	0.756	**0.848**
	Medium	0.638	0.253	0.276	0.446	0.642	**0.757**	0.651	0.261	0.067	0.350	0.654	**0.773**
	Long	0.586	0.169	0.194	0.359	0.562	**0.684**	0.589	0.173	0.045	0.214	0.575	**0.709**
Porto Taxi	Short	0.697	0.359	0.701	0.442	0.721	**0.804**	0.705	0.381	0.358	0.372	0.726	**0.832**
	Medium	0.622	0.271	0.416	0.403	0.619	**0.729**	0.634	0.293	0.106	0.326	0.628	**0.754**
	Long	0.565	0.184	0.305	0.340	0.547	**0.657**	0.578	0.198	0.036	0.218	0.568	**0.671**
Beijing Bicycle	Short	0.652	0.303	0.587	0.559	0.673	**0.788**	0.670	0.313	0.272	0.330	0.685	**0.802**
	Medium	0.568	0.217	0.603	0.461	0.582	**0.715**	0.574	0.226	0.142	0.304	0.589	**0.724**
	Long	0.503	0.129	0.613	0.297	0.487	**0.641**	0.519	0.139	0.045	0.206	0.492	**0.663**
Datasets	Metric	F1-score						EDT					
	Length	RICK	MPR	CTRR	STRNN	DeepMove	NASR	RICK	MPR	CTRR	STRNN	DeepMove	NASR
Beijing Taxi	Short	0.717	0.359	0.253	0.431	0.749	**0.834**	4.594	8.287	9.082	7.551	4.362	**3.376**
	Medium	0.644	0.257	0.108	0.392	0.648	**0.765**	8.273	16.321	23.110	14.725	8.730	**5.728**
	Long	0.587	0.171	0.073	0.268	0.568	**0.703**	11.283	25.873	27.493	22.705	12.059	**8.314**
Porto Taxi	Short	0.701	0.370	0.474	0.404	0.723	**0.818**	4.801	8.104	6.935	8.790	4.496	**3.563**
	Medium	0.628	0.282	0.169	0.360	0.623	**0.741**	8.619	15.032	18.294	13.368	8.930	**5.949**
	Long	0.571	0.191	0.065	0.266	0.557	**0.687**	11.379	21.349	31.745	19.603	12.297	**8.572**
Beijing Bicycle	Short	0.661	0.308	0.372	0.414	0.679	**0.795**	5.183	8.924	7.784	7.092	4.629	**3.719**
	Medium	0.571	0.221	0.229	0.367	0.585	**0.720**	8.972	17.497	20.966	14.503	9.039	**6.253**
	Long	0.511	0.134	0.084	0.243	0.489	**0.671**	11.891	22.028	57.997	21.324	12.692	**8.794**

rest locations are hidden. Each comparison method is required to recover the missing route between the source and destination.

Methods to Compare. We consider the following comparisons:

•*RICK* [30]: It builds a routable graph from uncertain trajectories, and then answers a users online query (a sequence of point locations) by searching top-k routes on the graph.

•*MPR* [4]: It discovers the most popular route from a transfer network based on the popularity indicators in a breadth-first manner.

•*CTRR* [6]: It proposes collaborative travel route recommendation by considering user's personal travel preference.

•*STRNN* [16]: Based on RNNs, it models local temporal and spatial contexts in each layer with transition matrices for different time intervals and geographical distances.

•*DeepMove* [9]: It is a multi-modal embedding RNN that can capture the complicated sequential transitions by jointly embedding the multiple factors that govern the human mobility.

Among these baselines, RICK and MPR are heuristic search based methods, CTRR is a machine learning method, and STRNN and DeepMove are deep learning methods. The parameters in all the models have been optimized using the validation set.

6.2 Results and Analysis

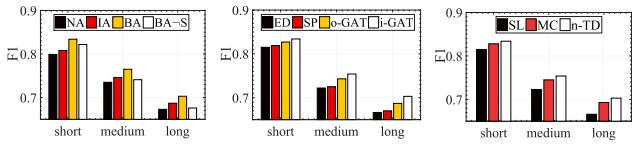
We present the results of all the comparison methods in Table 1. First, heuristic search methods, *i.e.,* RICK and MPR, perform very well, especially the RICK method. RICK fully characterizes the road network information and adopts the informed A^* algorithm. As a comparison, MPR mainly considers the modeling of transfer network and uses a relatively simple BFS search procedure. Second, the matrix factorization based method CTRR does not perform better than RICK and MPR. A possible reason is that CTRR can not well utilize the road network information. Besides, it has limited capacities in learning complicated trajectory characteristics. In our experiments, CTRR tends to generate short route recommendations, giving very bad recall results for medium and long queries. Third, deep learning method DeepMove performs very well among all the baselines, while STRNN gives a worse performance. Compared with STRNN, DeepMove considers more kinds of context information and designs more advanced sequential neural networks. Finally, the proposed model NASR is consistently better than all the baselines in all cases, yielding very good performance even on long queries.

By summarizing these results, we can see heuristic search methods are competitive to solve the PRR task, especially when suitable heuristics are used and context information is utilized. Besides, deep learning is also able to improve the performance by leveraging the powerful modeling capacity. Our proposed model NASR is able to combine both the benefits of heuristic search and neural networks, and hence it performs best among the comparison methods.

6.3 Detailed Analysis on Our Model NASR

In this section, we perform a series of detailed analysis on NASR for further verifying its effectiveness. Due to space limit, we only report the results of F1 scores on the *Beijing taxi* dataset. The rest results show the similar findings, and are omitted here.

Effect of the RNN Component. We first examine the effect of the RNN component with different variants. We have incorporated two kinds of attentions, namely inter- and intra-trajectory attention in Section 5.2. Here, we consider three variants of the attention mechanism for implementing $g(\cdot)$: *without attention (NA)*, *using only intra-trajectory attention (IA)* and *using both intra- and inter-trajectory attention (BA)*. Recall our RNN component is also able to learn a vectorized representation for the moving state of users. We further prepare a variant for verifying the effect of the learned



(a) Examining the RNN component. (b) Examining the value network. (c) Examining the TD learning method.

Figure 2: Detailed analysis of our model on the dataset of Beijing taxi using F1 measure.

moving state in the value network, namely the model that does not provide the moving state to the $h(\cdot)$ function, denoted by ($\underline{BA_{\neg S}}$). In Fig. 2(a), it can be seen that the performance rank is as follows: $NA < IA < BA$ and $BA_{\neg S} < BA$. It shows that both inter- and intra-trajectory attention are important to improve the performance of the PRR task. Especially, the learned moving state from the RNN component is useful for the value network. When the moving state is incorporated, the performance of the joint model has been substantially improved.

Effect of the Value Network. Predicting the estimated cost (*i.e.*, $h(\cdot)$) of a candidate location is especially important for our task. We use a value network for implementing $h(\cdot)$, which replaces the traditional heuristics. We now examine the performance of different variants for the value network. In this part, we fix the RNN component as its optimal setting. Then we prepare four variants for the value network as comparisons, including (1) *ED* using Euclid distance as heuristics, (2) *SP* using the scalar product between the embeddings of the candidate and destination locations, (3) *o-GAT* using the original implementation of graph attention networks, and (4) *i-GAT* using our improved GAT by incorporating context information. Both variants (3) and (4) are trained using the same TD learning method. In Fig. 2(b), it can be observed that the performance rank is as follows: *ED* < *SP* < *o-GAT* < *i-GAT*. We can see that the simplest spatial distance baseline *ED* gives the worst performance, which indicates simple heuristics may not work well in our task. Graph attention networks are more effective to capture structural characteristics from graphs. When incorporating context information, our value network is able to outperform the variant using original implementation.

Effect of Temporal Difference Learning Method. To learn our model, an important technique we apply is the Temporal Difference (TD) method. For verifying the effectiveness of the n-step TD method, we consider four variants for comparison, including (1) *SL* which directly learns the actual distance between the candidate location and the destination in a supervised way, (2) *MC* which applies Monte Carlo method to generate sampled sequences and trains the model with the cost of these sampled sequences [25], (3) *n-TD* which uses a TD step number of 5. From Fig. 2(c), we can see that the simplest supervised learning method performs worst. Since the prediction involves multi-step moving process, it is not easy to directly fit the distance using traditional supervised learning methods. Compared with all the methods, we can see that the 5-TD learning method is the most effective in our task. In our experiments, we find that using a step number of 5 produces the optimal performance.

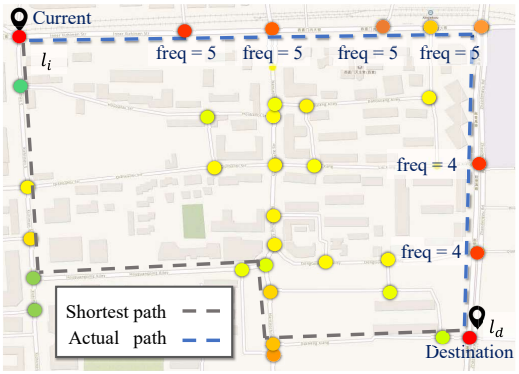


Figure 3: Visualization of the learned association scores using improved graph attention networks. The colored circles denote locations in the road network. A darker color indicates a larger importance degree *w.r.t.* current location l_i and destination l_d. "*freq*" denotes the visit frequency by the user in historical trajectories.

6.4 Qualitative Analysis

Previously, we have shown the effectiveness of our model in the PRR task. In this part, we qualitatively analyze why NASR is able to yield a good performance.

In NASR, the improved graph attention network is the core component for modeling road network information. It can generate informative node representations for encoding structural characteristics. To see this, we present an illustrative example in Fig. 3. A user is currently located at l_i and moving towards the destination l_d. For a candidate location l_j, we compute a simple scoring formula: $\boldsymbol{n}_{l_j}^\top \cdot \boldsymbol{n}_{l_i} + \boldsymbol{n}_{l_j}^\top \cdot \boldsymbol{n}_{l_d}$, where $\boldsymbol{n}_{(\cdot)}$s are the node representations learned in Eq. (15). This formula measures the association degree of l_j with both current location and destination. For comparison, we plot both the actual and shortest route. As we can see, the locations on the actual route has a larger association weight than those on the shortest route. By inspecting into the dataset, we find the shortest route contains several side road segments that are possibly in traffic congestion at the visit time. Another interesting observation is that the user indeed visits the locations in the actual route more times in historical trajectories. These observations indicate that our model is able to learn effective node representations for identifying more important locations to explore for the PRR task.

Next, we continue to study how the learned cost function helps the search procedure in NASR. Figure 4 presents a sample trajectory from a specific user. Given the source and destination, we need to predict the actual route. By comparing Fig. 4(a) (the original search space) and Fig. 4(b) (the reduced search space by NASR), it can be seen that our model is able to effectively reduce the search space. When zooming into a subsequence of this route, we further compare the estimated cost values for two candidate locations (green points) in Fig 4(c). Although the second location has a longer distance with the explored locations, it is located on the main road that is likely to lead to a better traffic condition. Our model is able to predict a lower cost for the second location by effectively learning such trajectory characteristics from road network and historical data.

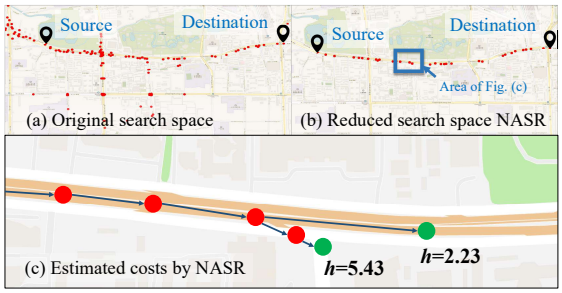


Figure 4: Visualization of the search procedure with the estimated costs by the NASR model. In (c), red points have been already explored and green points are candidate locations to extend in A^* search algorithm.

7 CONCLUSIONS

In this paper, we took the initiative to use neural networks to automatically learn the cost functions in A^* for the PRR task. We first presented a simple A^* solution for solving the PRR task, and formally defined the suitable form for the search cost. Then, we set up two components to learn the two costs respectively, *i.e.,* the RNN component for $g(\cdot)$ and the value network for $h(\cdot)$. The two components were integrated in a principled way for deriving a more accurate cost of a candidate location for search. We constructed extensive experiments for verifying the effectiveness and robustness of the proposed model.

Since road network information is not always available, as future work, we will consider extending our model to solve the PRR task without road networks. Currently, we focus on the PRR task. We will also study whether our solution can be generalized to solve other complex search tasks.

ACKNOWLEDGMENTS

The work was partially supported by National Natural Science Foundation of China under the Grant Number 61572059, 71531001, 61872369 and the Fundamental Research Funds for the Central Universities. Wang's work was supported by the National Key Research and Development Program of China under Grant No. 2016YFC1000307. Zhao's work was supported by the Research Funds of Renmin University of China under Grant No. 18XNLG22. Wu's work was supported by the Science and Technology Project of Beijing under Grant No.Z181100003518001, and the Open Foundation of TUCSU under Grant No.TUCSU-K-17002-01.

R.I.P. to Prof. Nils Nilsson, the co-inventor of the A^* algorithm. This work is based on his groundbreaking work.

REFERENCES

[1] Abdulrahman Al-Molegi, Mohammed Jabreel, and Baraq Ghaleb. 2016. STF-RNN: Space Time Features-based Recurrent Neural Network for predicting people next location. In *SSCI*. 1–7.
[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural Machine Translation by Jointly Learning to Align and Translate. (2014). arXiv:1409.0473
[3] Dawei Chen, Cheng Soon Ong, and Lexing Xie. 2016. Learning points and routes to recommend trajectories. In *CIKM*. ACM, 2227–2232.
[4] Zaiben Chen, Heng Tao Shen, and Xiaofang Zhou. 2011. Discovering popular routes from trajectories. In *ICDE*. 900–911.
[5] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. 2015. Gated feedback recurrent neural networks. In *ICML*. 2067–2075.
[6] Ge Cui, Jun Luo, and Xin Wang. 2018. Personalized travel route recommendation using collaborative filtering based on GPS trajectories. *IJED* 11, 3 (2018), 284–307.
[7] J. Dai, B. Yang, C. Guo, and Z. Ding. 2015. Personalized route recommendation using big trajectory data. In *ICDE*. 543–554.
[8] Marco Ernandes and Marco Gori. 2004. Likely-admissible and sub-symbolic heuristics. In *ECAI*. Citeseer, 613–617.
[9] Jie Feng, Yong Li, Chao Zhang, Funing Sun, Fanchao Meng, Ang Guo, and Depeng Jin. 2018. DeepMove: Predicting Human Mobility with Attentional Recurrent Networks. In *WWW*. 1459–1468.
[10] Peter E Hart, Nils J Nilsson, and Bertram Raphael. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE T-SSC* 4, 2 (1968), 100–107.
[11] Evangelos Kanoulas, Yang Du, Tian Xia, and Donghui Zhang. 2006. Finding fastest paths on a road network with speed patterns. In *ICDE*. IEEE, 10–10.
[12] Takeshi Kurashima, Tomoharu Iwata, Go Irie, and Ko Fujimura. 2010. Travel route recommendation using geotags in photo sharing sites. In *CIKM*. 579–588.
[13] Levi Lelis, Roni Stern, and Shahab Jabbari Arfaee. 2011. Predicting solution cost with conditional probabilities. In *ASSC*.
[14] Kwan Hui Lim, Jeffrey Chan, Christopher Leckie, and Shanika Karunasekera. 2015. Personalized Tour Recommendation Based on User Interests and Points of Interest Visit Durations.. In *IJCAI*, Vol. 15. 1778–1784.
[15] Hechen Liu, Ling Yin Wei, Yu Zheng, and M Schneider. 2011. Route Discovery from Mining Uncertain Trajectories. (2011), 1239–1242.
[16] Qiang Liu, Shu Wu, Liang Wang, and Tieniu Tan. 2016. Predicting the next location: a recurrent model with spatial and temporal contexts. In *AAAI*. 194–200.
[17] Wuman Luo, Haoyu Tan, Lei Chen, and Lionel M. Ni. 2013. Finding time period-based most frequent path in big trajectory data. In *SIGMOD*. 713–724.
[18] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529.
[19] Karl Nachtigall. 1995. Time depending shortest-path problems with applications to railway networks. *EJOR* 83, 1 (1995), 154–166.
[20] Stefano Pallottino and Maria Grazia Scutella. 1998. Shortest path algorithms in transportation models: classical and innovative aspects. In *Equilibrium and advanced transportation modelling*. Springer, 245–281.
[21] Mehdi Samadi, Ariel Felner, and Jonathan Schaeffer. 2008. Learning from Multiple Heuristics.. In *AAAI*. 357–362.
[22] Samia Shafique and Mohammed Eunus Ali. 2016. Recommending most popular travel path within a region of interest from historical trajectory data. In *SIGSPATIAL*. ACM, 2–11.
[23] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587 (2016), 484.
[24] Richard S Sutton. 1988. Learning to predict by the methods of temporal differences. *Machine learning* 3, 1 (1988), 9–44.
[25] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.
[26] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* 1, 2 (2017).
[27] Jingyuan Wang, Chao Chen, Junjie Wu, and Zhang Xiong. 2017. No longer sleeping with a bomb: a duet system for protecting urban safety from dangerous goods. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1673–1681.
[28] Jingyuan Wang, Xu He, Ze Wang, Junjie Wu, Nicholas Jing Yuan, Xing Xie, and Zhang Xiong. 2018. CD-CNN: a partially supervised cross-domain deep learning model for urban resident recognition. In *Thirty-Second AAAI Conference on Artificial Intelligence*.
[29] Jingyuan Wang, Junjie Wu, Fei Gao, and Zhang Xiong. 2019. Understanding Urban Dynamics via Context-aware Tensor Factorization with Neighboring Regularization. *arXiv preprint arXiv:1905.00702* (2019).
[30] Ling Yin Wei, Yu Zheng, and Wen Chih Peng. 2012. Constructing popular routes from uncertain trajectories. In *SIGKDD*. 195–203.
[31] Hao Wu, Ziyang Chen, Weiwei Sun, Baihua Zheng, and Wei Wang. 2017. Modeling Trajectories with Recurrent Neural Networks. In *ICJAI*. 3083–3090.
[32] Hao Wu, Jiangyun Mao, Weiwei Sun, Baihua Zheng, Hanyuan Zhang, Ziyang Chen, and Wei Wang. 2016. Probabilistic Robust Route Recovery with Spatio-Temporal Dynamics. In *SIGKDD*. 1915–1924.
[33] Can Yang and Gyozo Gidofalvi. 2018. Fast map matching, an algorithm integrating hidden Markov model with precomputation. *IJGIS* 32, 3 (2018), 547–570.
[34] Cheng Yang, Maosong Sun, Wayne Xin Zhao, Zhiyuan Liu, and Edward Y. Chang. 2017. A Neural Network Approach to Jointly Modeling Social Networks and Mobile Trajectories. *ACM T-IS* 35, 4 (2017), 36:1–36:28.
[35] 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 *SIGSPATIAL*. ACM, 99–108.
[36] Stephan Zheng, Yisong Yue, and Patrick Lucey. 2017. Generating Long-term Trajectories Using Deep Hierarchical Networks. In *NIPS*. 1543–1551.