Effective and Efficient Route Planning Using Historical Trajectories on Road Networks

Wei Tian
Hong Kong Polytechnic University
wei.tian@connect.polyu.hk

Jieming Shi*
Hong Kong Polytechnic University
jieming.shi@polyu.edu.hk

Siqiang Luo
Nanyang Technological University
siqiang.luo@ntu.edu.sg

Hui Li
Xiamen University
hui@xmu.edu.cn

Xike Xie
University of Science and Technology of China
xkxie@ustc.edu.cn

Yuanhang Zou
Tencent Co. Ltd.
yuanhangzou@tencent.com

ABSTRACT

We study route planning that utilizes historical trajectories to predict a realistic route from a source to a destination on a road network at given departure time. Route planning is a fundamental task in many location-based services. It is challenging to capture latent patterns implied by complex trajectory data for accurate route planning. Recent studies mainly resort to deep learning techniques that incur immense computational costs, especially on massive data, while their effectiveness are complicated to interpret.

This paper proposes DRPK, an effective and efficient route planning method that achieves state-of-the-art performance via a series of novel algorithmic designs. In brief, observing that a route planning query (RPQ) with closer source and destination is easier to be accurately predicted, we fulfill a promising idea in DRPK to first detect the key segment of an RPQ by a classification model KSD, in order to split the RPQ into shorter RPQs, and then handle the shorter RPQs by a destination-driven route planning procedure DRP. Both KSD and DRP modules rely on a directed association (DA) indicator, which captures the dependencies between road segments from historical trajectories in a surprisingly intuitive but effective way. Leveraging the DA indicator, we develop a set of well-thought-out key segment concepts that holistically consider historical trajectories and RPQs. KSD is powered by effective encoders to detect high-quality key segments, without inspecting all segments in a road network for efficiency. We conduct extensive experiments on 5 large-scale datasets. DRPK consistently achieves the highest effectiveness, often with a significant margin over existing methods, while being much faster to train. Moreover, DRPK is efficient to handle thousands of online RPQs in a second, *e.g.*, 2768 RPQs per second on a PT dataset, *i.e.*, 0.36 milliseconds per RPQ.

PVLDB Reference Format:
Wei Tian, Jieming Shi, Siqiang Luo, Hui Li, Xike Xie, and Yuanhang Zou. Effective and Efficient Route Planning Using Historical Trajectories on Road Networks. PVLDB, 16(10): 2512-2524, 2023.
doi:10.14778/3603581.3603591

*Corresponding Author.

Proceedings of the VLDB Endowment, Vol. 16, No. 10 ISSN 2150-8097.
doi:10.14778/3603581.3603591

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/derekwtian/DRPK.

1 INTRODUCTION

With the popularity of location-based services, massive trajectory data become highly available, which attracts much research attention to support important applications [17, 23–27, 42]. Given the historical trajectory data $\mathcal{D}$ on a road network, we focus on *non-personalized route planning*, aiming to predict a realistic path from a source to a destination (an SD pair) at certain departure time, specified by a route planning query (an RPQ). A route (a.k.a. path) is a sequence of connected road segments in the road network. Route planning has important use cases in navigation [37], food delivery [18], ride-sharing [17], etc. The non-personalized setting does not require extra user information, and thus provides convenience to new cold-start users and also the users sensitive to such information. For instance, route planning can recommend a promising route towards a destination, where a new ride-sharing driver has never been before. The route is generated based on the underlying mainstream travel patterns implied by historical trajectories.

Effective route planning is a highly challenging task, especially for large-scale data with millions of trajectories on large road networks. It is non-trivial to efficaciously capture the latent pivotal patterns among road segments from the complicated data involving trajectory sequences on network topology. Given an RPQ, an early wrong prediction near the source could cause a great divergence between the predicted and the real routes, especially for long RPQs with faraway source and destination. A collection of existing studies formulate route planning as path finding or graph search problems based on certain cost functions over various factors [7, 9, 26, 36]. The design of cost functions relies on expert knowledge. Besides, real trajectories do not always match the paths with lowest costs, *e.g.*, distance and time [12, 30]. As shown in Section 6.2, in an SF dataset, the average percentage of overlapping length of the shortest paths over the corresponding real trajectories is just 49.1%.

Thus, another plethora of existing solutions learn travel patterns from historical trajectories to plan routes [11, 12, 22, 24, 25, 34, 37], by adopting learning models, especially Recurrent Neural Networks (RNNs) [8, 16]. However, these methods incur immense overheads to train complex models in the offline stage, and are also slow for online RPQ inference. Moreover, their effectiveness is elusive to interpret. Specifically, as revealed in [17], a significant portion of the predicted routes cannot even reach the destinations of the processed RPQs, which significantly comprises their real-world utility. To recapitulate, existing solutions are facing either efficiency or effectiveness issues for route planning using big trajectory data.

Adhering to making the complicated simple, we propose DRPK (short for Directed Association based Route Planning with Key Segment Detection), a novel solution that yields state-of-the-art route planning accuracy, while being highly efficient, which is fulfilled through several thoughtful designs. In a nutshell, we observe that RPQs with closer SD pairs are easier to be accurately planned, and hence, carry out an intriguing idea to let DRPK first invoke a key segment detection model KSD that identifies the key segment e_{key} of an RPQ q, to split q into shorter RPQs (q_1 from the source to e_{key} and q_2 from e_{key} to the destination); both q_1 and q_2 are then solved by a destination-driven route planning procedure DRP that relies on a new directed association (DA) indicator σ to plan routes.

The superiority of DRPK is non-trivial to achieve. The first challenging task is to capture segment dependencies implied by massive historical trajectories $\mathcal{D}$. To deal with it, we propose the DA indicator σ, in which the DA strength between two segments quantifies the historical trend of going from one to the other. Compared with expensive RNNs used in the literature, the construction of σ only requires efficient statistical counting. Then starting from the source of an RPQ, the DRP procedure always picks the adjacent segment with the highest DA strength towards the destination to expand and plan the route (*i.e.*, destination-driven), assisted by an auxiliary traffic popularity technique. In experiments, DRP itself already exhibits comparable performance over existing methods, validating the effectiveness of the DA indicator.

As for the KSD model, the challenges are twofold. It is unclear how to properly measure if a segment is key or not *w.r.t.* an RPQ. Further, it is inefficient to inspect every segment in a large road network to detect key segments. To tackle the challenges, we exploit the DA indicator σ again and develop a complete set of key segment concepts that consider RPQs and trajectories on road networks as a whole. In particular, given an RPQ, we devise its candidate key segment pool of small size and formulate KSD as a binary classification model over the pool. Thus, we avoid inspecting all segments in a road network for efficiency. We develop effective and efficient encoders and loss function to build and train the KSD model in Section 5.

Extensive experiments over 5 real-world trajectory datasets on road networks demonstrate that DRPK consistently outperforms its competitors in terms of result quality, at a fraction of their training costs. For instance, on a PT dataset with trajectories in millions, DRPK only takes 0.65 hour to train, while a fast competitor require 7.59 hours and the other competitors need more than a day.

To sum up, we make the following contributions in our paper.

- We propose a new solution DRPK for route planning using historical trajectories on road networks. The main technical designs in DRPK include the DA indicator σ, the destination-driven procedure DRP, and the key segment detection model KSD.
- We devise the DA indicator to preserve the segment associations implied by historical trajectories in an intuitive, effective and efficient way. Then we design DRP to perform destination-driven route planning guided by the DA indicator as well as an auxiliary traffic popularity technique.

Table 1: Frequently used notations

Notation	Description
$\mathcal{D}$	Historical trajectory data $\mathcal{D}$.
$T, T.s, T.d,$ $T.seq$	A trajectory T from its source $T.s$ to destination $T.d$ with a sequence of traveled segments $T.seq$.
$G = (V, E)$, n	A road network G with road segment set E and intersection set V. And n is the number of segments.
$e_i, N_o(e_i)$	A road segment e_i and its out-going adjacent segments.
$q = \langle s, d, t_q \rangle$	A route planning query q with source and destination GPS coordinates s and d and departure time t_q.
e_s, e_d	The source and destination segments where s and d are respectively, obtained by map-matching.
$\hat{t}_q$	The time slot of timestamp t_q.
$\omega(e_i, \langle e_s, e_d \rangle)$	The importance of segment e_i *w.r.t.* SD pair $\langle e_s, e_d \rangle$.
$\mathcal{K}_T$	The ground-truth key segment set of trajectory T.
C_{sd}, k_c	The candidate key segment pool of the SD pair of an RPQ q, with size k_c.
$P_{key}(e_i\|q)$, $e_i \in C_{sd}$	The key segment probability of a candidate e_i in the candidate pool C_{sd} of the RPQ q.
$\mathcal{R}$	The predicted route of an RPQ.

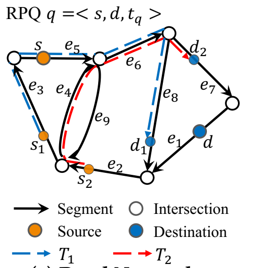
- Utilizing the DA indicator again, we derive a holistic set of key segment concepts that properly take RPQs and historical trajectories into consideration. We then develop KSD, a classification model to effectively detect key segments over a small candidate pool of an online RPQ.
- The superiority of DRPK, in terms of efficiency and effectiveness, is evaluated over massive trajectory data on road networks.

2 PRELIMINARIES

2.1 Problem Formulation

Road Network. A road network is modeled as a directed graph $G = (V, E)$, where V is a set of nodes, and E is a set of directed edges. A node $v \in V$ represents an intersection or a road end. A directed edge $e = (v_i, v_j) \in E$ is a road segment from entrance point v_i to exit point v_j. Denote $n = |E|$ and $m = |V|$ as the number of road segments and intersections, respectively. Road network data are from OpenStreetMap [2]. A segment e_i has length $l(e_i)$ in meters. Figure 1a shows a road network with black arrows as segments and white circles as intersections. Let $N_o(e_i)$ be the set of out-going adjacent segments of e_i, *e.g.*, $N_o(e_6) = \{e_7, e_8\}$ in Figure 1a.

Trajectory. A raw trajectory is a sequence of GPS points in the form of ([*latitude, longitude*], *time*), meaning that the moving object is at location [*latitude,longitude*] at the *time*. In this work, we consider trajectories generated in a road network. In other words, the GPS points of raw trajectories are on road segments. Therefore, we map raw trajectories onto road segments by a popular map-matching algorithm [41], also adopted in existing studies [17, 34]. After map-matching, a trajectory T consists of (i) a sequence of road segments $T.seq = \langle e_1, e_2, ..., e_\ell \rangle$ with length $T.\ell$, (ii) a source location $T.s$ on the source segment e_1 and a destination location $T.d$ on the destination segment e_ℓ, associated with respective departure time and arrival time, and (iii) the entry and exit timestamps $et(e_i, T)$ and $xt(e_i, T)$ of T on every segment e_i in $T.seq$. Particularly, $et(e_i, T)$ and $xt(e_i, T)$ are obtained by linear interpolation after map-matching [42], and the travel time of T over e_i is $t(e_i, T) = xt(e_i, T) - et(e_i, T)$. Given historical trajectories $\mathcal{D}$, we can obtain the average travel time $t(e_i)$ on segment e_i by averaging



(a) Road Network

$\sigma(e_6, e_1) = 6$, $\sigma(e_7, e_1) = 9$,
$\sigma(e_8, e_1) = 4$, $\sigma(e_9, e_1) = 2$,
$\sigma(e_7, e_2) = 3$, $\sigma(e_8, e_2) = 3$,
$\sigma(e_2, e_3) = 8$, $\sigma(e_2, e_4) = 2$,
$\sigma(e_2, e_5) = 3$, $\sigma(e_2, e_6) = 5$,
$\sigma(e_3, e_7) = 1$, $\sigma(e_4, e_7) = 3$,
$\sigma(e_5, e_7) = 4$, $\sigma(e_6, e_7) = 7$

(b) DA Strength Values

Figure 1: Running Example

the travel time $t(e_i, T)$ of all $T \in \mathcal{D}$. In the following, the map-matched trajectories are referred to as trajectories. Figure 1a shows two trajectories T_1 and T_2 in dashed lines. T_1 has its source s_1 on segment e_3, destination d_1 on segment e_8, and $T_1.seq = \langle e_3, e_5, e_6, e_8 \rangle$.

Route Planning Query. Given a source s and a destination d, one can apply the map-matching method [41] again to get the source segment e_s and destination segment e_d, where s and d reside respectively. Note that s can be anywhere on e_s. Let position ratio r_s be the ratio of the distance between s and the entrance of e_s over segment length $l(e_s)$. Similarly, let r_d be the position ratio of d on e_d. A *route* from s to d is a *path* of connected road segments from e_s to e_d on the road network G. Then an RPQ is defined as follows. When the context is clear, we refer to either $\langle s, d \rangle$ or $\langle e_s, e_d \rangle$ as an SD pair. Table 1 shows the frequently used notations.

Definition 2.1. (Route Planning Query). Given historical trajectory data $\mathcal{D}$ on a road network G, an RPQ $q = \langle s, d, t_q \rangle$, consisting of a source location s, a destination location d, and departure time t_q, asks for the most likely route $\mathcal{R}$ with high probability $Pr(\mathcal{R}|q, \mathcal{D}, G)$ based on the travel patterns implied in $\mathcal{D}$.

2.2 Overview of Current Approaches

We overview the main competitors here, while discussing other related work in Section 7.

A vital task in route planning is to capture the transition probabilities among road segments *w.r.t.* RPQs. Recent studies attempt to adopt complicated models to preserve such transition patterns over historical trajectories. NMLR [17] is a latest method that utilizes Lipschitz embeddings [5] and Graph Convolutional Networks (GCN) [21] together to learn the probabilities from historical trajectories on road networks and then generates routes by either Dijkstra or greedy search. CSSRNN [37] is based on RNN [8, 16], a sequential deep learning model, to learn the transition probabilities. CSSRNN integrates the constraints of road networks (*e.g.*, reachability) into RNN to capture long-term dependencies of road segments, and is trained to minimize the error between the predicted and the real transition probability distributions implied by historical trajectories. NASR [34] automatically learns the cost functions in A* algorithm by deep learning models, including using RNN for observable cost and adopting Graph Attention Networks (GAT) [33] for estimated cost, and then utilizes A* for route planning. DeepMove [11] also uses RNNs to capture sequential transition patterns in personalized historical trajectories, and has an attention model to capture multi-level periodicity of mobility patterns. In experiments, it is extended to road network without personalization.

Algorithm 1: DRPK (Online)

Input: RPQ $q = \langle s, d, t_q \rangle$ where s and d are on segments e_s and e_d of the input road network G respectively, the KSD model, the DA indicator σ

Output: The predicted route $\mathcal{R}$

1 Get the candidate key segment pool C_{sd} of q (Definition 5.2, Section 5.1);
2 Get the key segment probability $P_{key}(e_i|q)$ of every candidate $e_i \in C_{sd}$ by the KSD model (Sections 5.2 and 5.3);
3 Predicted key segment $e_{key} \leftarrow \arg\max_{\forall e_i \in C_{sd}} P_{key}(e_i|q)$;
4 RPQ $q_1 = \langle s, v^+_{key}, t_q \rangle$, where v^+_{key} is the entrance of e_{key};
5 Predicted route and arrival time $\mathcal{R}_1, t_1 \leftarrow$ Invoke DRP on q_1 (Algorithm 3);
6 RPQ $q_2 = \langle v^+_{key}, d, t_1 \rangle$ from source segment e_{key} to e_d;
7 Predicted route and arrival time $\mathcal{R}_2, t_2 \leftarrow$ Invoke DRP on q_2 (Algorithm 3);
8 $\mathcal{R} \leftarrow concat(\mathcal{R}_1, \mathcal{R}_2)$;
9 **return** $\mathcal{R}$;

Novelty. Our technical novelty lies in three aspects. First, instead of the complicated models (*e.g.*, RNN, GCN, and GAT) commonly used in the approaches above, we propose a new DA indicator to preserve transition patterns via efficient statistical counting. Second, to the best of our knowledge, for the route planning problem in Definition 2.1, it is the first time to formulate KSD as a classification task from scratch with a complete set of concepts developed, without inspecting all segments. Third, DRPK assembles the novel components above to first split RPQs by KSD and then handle them by DRP, for superior effectiveness and efficiency on real data.

3 THE DRPK SOLUTION

The DRPK solution has two phases. In the offline phase, we build the DA indicator using historical trajectories (Section 4.1), and train the key segment detection model KSD by historical trajectories (Section 5). In the online phase, for an RPQ q to be issued in the future, DRPK invokes the KSD model and the destination-driven route planning procedure DRP (Section 4.2) to get the predicted route $\mathcal{R}$ for RPQ q. In this section, we present the pseudocode of DRPK in Algorithm 1 for online RPQ processing.

Recall that DRPK first detects the key segment of an RPQ q by KSD, and then splits q into two shorter RPQs q_1 and q_2 which are handled by DRP. Given the input RPQ in Algorithm 1, $q = \langle s, d, t_q \rangle$, its route usually involves only a small part of the whole input road network G. In other words, most road segments in G are less relevant or even irrelevant to the query. Hence, to detect the key segment of q, it is not necessary to inspect every road segment in G in details. Therefore, at Line 1, DRPK first gets a small pool C_{sd} of candidate key segments *w.r.t.* q (defined in Section 5.1), and only focuses on the pool for key segment detection, rather than on all road segments in G. According to our definitions, C_{sd} contains the top segments that are important to *both* the source and the destination of q, which is measured by the DA indicator σ. A segment that is less important to either the source or the destination of q is unlikely to be in C_{sd}, and subsequently cannot be a key segment of q. Then, at Line 2, we execute the KSD model forwardly to predict the key segment probability of every candidate $e_i \in C_{sd}$ *w.r.t.* RPQ q, and select the segment e_{key} with the highest probability as the predicted key segment of q (Line 3).

At Line 4, the first RPQ q_1 is formed as $\langle s, v^+_{key}, t_q \rangle$, where v^+_{key} is the entrance point of e_{key} and e_{key} is the destination segment of q_1. RPQ q_1 has the same source s and departure time t_q as RPQ q. Then DRPK invokes DRP on q_1 to get the predicted route $\mathcal{R}_1$ and predicted arrival time t_1 as a byproduct (Line 5). The predicted arrival time t_1 of q_1 is useful at Line 6, as the departure time of the second RPQ q_2. RPQ q_2 starts from v^+_{key} with source segment e_{key} towards the final destination d of the original RPQ q. At Line 7, DRPK invokes DRP on q_2 to get the predicted route $\mathcal{R}_2$ and arrival time t_2. At Line 8, the final predicted route $\mathcal{R}$ is obtained by merging $\mathcal{R}_1$ and $\mathcal{R}_2$, and returned at Line 9.

Remark. In Algorithm 1, DRPK only invokes KSD once for an RPQ, to explain our main ideas. In Section 6.5, we also experimentally evaluate the extension of DRPK with multiple key segments per RPQ and analyze its performance trade off.

4 DIRECTED ASSOCIATION

In Section 4.1, we present the directed association (DA) indicator σ to capture the relationships between road segments based on historical trajectories. Then, in Section 4.2, we develop the DRP procedure that leverages the DA indicator to perform destination-driven route planning, assisted by a traffic popularity technique.

4.1 DA Construction

As mentioned, a vital task is to extract transition patterns from historical trajectories $\mathcal{D}$. Existing methods mainly resort to deep learning with tremendous overheads. We propose the DA indicator σ that is intuitive and effective, while being efficient to construct.

Given a historical trajectory T with sequence $T.seq = \langle e_1, e_2, ..., e_\ell \rangle$, $T.seq$ reflects the driver's intention that every segment e_i $(1 \le i < \ell)$ is chosen with the goal to be directed to the destination segment e_ℓ. $T.seq$ also indicates that e_i is selected with the consideration to arrive at some intermediate segment e_j first $(1 \le i < j \le \ell)$, in order to finally arrive at the destination. The DA indicator is designed to preserve these associations. Specifically, the DA strength $\sigma(e_i, e_j)$ from segments e_i to e_j indicates the trend of going to e_j from e_i based on historical trajectories. Note that e_i and e_j are not necessarily adjacent in the road network. Also the indicator σ is directed, *i.e.*, the DA strength $\sigma(e_i, e_j)$ is different from $\sigma(e_j, e_i)$.

How to construct the DA indicator σ over historical trajectories $\mathcal{D}$ is presented at Lines 1-5 of Algorithm 2. We implement the DA indicator by a two-layer dictionary to store the DA strength values between road segments. Initially, σ is empty (Line 1). For every training trajectory $T \in \mathcal{D}$ with $T.seq = \langle e_1, e_2, ..., e_\ell \rangle$ (Lines 2-4), $T[e_i, e_j]$ represents a sub-trajectory of T starting from e_i and ending at e_j, for all possible $1 \le i < j \le \ell$. Therefore, we increase $\sigma(e_i, e_j)$ by one for every such sub-trajectory of T (Line 5). After going through all trajectories in $\mathcal{D}$, the DA indicator is constructed.

Discussion. One may feel that the DA indicator σ is intuitive to construct. We emphasize that this is actually an interesting finding made in this paper. As shown in our experiments, σ is important in DRP for accurate route planning. This demonstrates the simplicity and effectiveness of σ, compared with the complex learning designs in the literature. More importantly, based on σ, we develop the KSD model in Section 5, to further boost the performance of DRPK.

Algorithm 2: Build DA indicator and Traffic Popularity

Input: Historical trajectories $\mathcal{D}$ as training data
Output: DA indicator σ and traffic popularity $\mathbf{P}$

1 Initialize σ as an empty dictionary;
2 **foreach** *Trajectory* $T \in \mathcal{D}$ **do**
3   **for** $i \leftarrow 1, 2, ..., T.\ell$ **do**
4   **for** $j \leftarrow i+1, ..., T.\ell$ **do**
5             $\sigma(e_i, e_j) \leftarrow \sigma(e_i, e_j) + 1$;
6 Initialize $\mathbf{P}$ as a zero matrix;
7 **foreach** *Trajectory* $T \in \mathcal{D}$ **do**
8   **for** $i \leftarrow 1, 2, ..., T.\ell$ **do**
9         $\hat{et}(e_i, T) \leftarrow$ time slot by Eq. (1) for entry time $et(e_i, T)$;
10         $\mathbf{P}[e_i, \hat{et}(e_i, T)] \leftarrow \mathbf{P}[e_i, \hat{et}(e_i, T)] + 1$;

Furthermore, the average sequence length $T.\ell$ of real trajectories is just in dozens, as shown in Table 2 in experiments. Thus, it is practically efficient to iterate historical trajectories to construct σ. In particular, it only costs just tens of seconds over millions of trajectories to build σ as reported in Table 10 of Section 6.3.

4.2 DRP: Destination-driven Route Planning

The majority of moving objects on road networks are with clear goals to arrive at their destinations. Hence, it is reasonable to deduce that most historical trajectories (and future RPQs) were generated (will be issued) with the objective to reach their destinations. Thus, we design DRP, a destination-driven method for route planning. In brief, starting from the source segment e_s of an RPQ q, DRP expands the predicted route by always appending the adjacent segment with the highest DA strength to the destination segment e_d of q.

Running example. In Figure 1a, an RPQ q asks for a route from the source s on segment e_5 to the destination d on segment e_1. Figure 1b provides example DA values σ built from historical trajectories on the road network in Figure 1a. Starting from source segment e_5 with adjacent segments $N_o(e_5) = \{e_6, e_9\}$, DRP adds e_6 into the planned route since e_6 has stronger DA strength towards the destination segment e_1, *i.e.*, $\sigma(e_6, e_1) = 6 > \sigma(e_9, e_1) = 2$. Then with adjacent segments $N_o(e_6) = \{e_7, e_8\}$, DRP expands the route by e_7, since $\sigma(e_7, e_1) = 9 > \sigma(e_8, e_1) = 4$. Then from e_7, DRP finds that the destination segment e_1 is in $N_o(e_7)$ (*i.e.*, destination reached), and returns the planned route ended at e_1.

Since DRP is destination-driven, only the DA values $\sigma(*, e_d)$ towards the destination e_d are used when processing an RPQ with destination segment e_d. There could be tie cases when comparing DA values. For instance, in Figure 1, if the destination segment of another RPQ is e_2 and DRP needs to pick a segment from $N_o(e_6) = \{e_7, e_8\}$, both $\sigma(e_7, e_2)$ and $\sigma(e_8, e_2)$ are 3, which is a tie case. It is also possible that no DA values exist between certain road segments and the destination, which is a tie too. A brute-force way is to break ties arbitrarily, which is ineffective as validated in experiments.

Therefore, in the following, we first present a traffic popularity technique $\mathbf{P}$ to assist DRP to effectively break ties, and then present the complete algorithm of DRP.

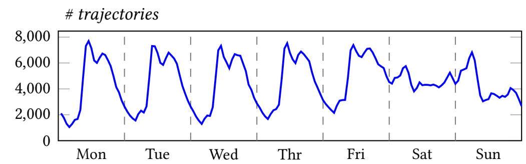


Figure 2: Traffic Periodicity Per Hour Per Day

Traffic popularity. In modern road networks, due to the increasingly mature urban planning, road segments are uncrowded for most of the time. The majority tend to choose relatively common road segments to travel. Hence, we suggest to break the tie cases of DA strengths by choosing the relative popular segment in the corresponding time period. Specifically, for every road segment e_i, we maintain its traffic popularity in n_t time slots. Given historical trajectory data $\mathcal{D}$, the traffic popularity $\mathbf{P}[e_i, \hat{t}_j]$ of segment e_i at time slot $\hat{t}_j$ is the number of trajectories entering e_i within the time slot $\hat{t}_j$. Then, we store the traffic popularity of all segments by $\mathbf{P} \in \mathbb{R}^{n \times n_t}$, where n_t is a small number, explained below.

Now the question is how to decide n_t. Figure 2 shows the number of trajectories per hour per day of a PT dataset. Observe that weekdays and weekends exhibit different traffic patterns. As an example, a road segment towards a resort could be more popular on weekends than weekdays. Also, within weekdays, the traffic exhibits periodicity. Therefore, we distinguish the traffic popularity of a road segment into weekdays and weekends (each counts for $n_t/2$ time slots). Let t_0 be the starting time of a day and δ be the time slot duration in seconds (*e.g.*, 3600s). $\delta \cdot n_t/2$ is 24 hours, which decides n_t. Given a timestamp t_j, we first parse the timestamp into the *day* of a week, and the time remainder tr_j in seconds. Then the time slot id $\hat{t}_j$ of timestamp t_j is calculated by

$$\hat{t}_j = \begin{cases} \lfloor \frac{tr_j}{\delta} \rfloor, & day \text{ is a weekday,} \\ \lfloor \frac{tr_j}{\delta} \rfloor + n_t/2, & \text{otherwise.} \end{cases} \tag{1}$$

Lines 6-10 in Algorithm 2 show the build of $\mathbf{P}$ based on historical trajectories $\mathcal{D}$. Initially, $\mathbf{P}$ is a zero matrix (Line 6). Then we scan every segment e_i in every trajectory T (Lines 7-8), get the time slot $\hat{et}(e_i, T)$ of its entry time $et(e_i, T)$ at e_i by Eq. (1) at Line 9, and increase $\mathbf{P}[e_i, \hat{et}(e_i, T)]$ by one at Line 10.

DRP algorithm. The pseudocode of DRP is presented in Algorithm 3. DRP takes as input an RPQ $q = \langle s, d, t_q \rangle$ with source and destination segments e_s and e_d, the DA indicator σ and traffic popularity $\mathbf{P}$. DRP returns the predicted route $\mathcal{R}$ and also the predicted arrival time t_a, which is useful in DRPK (Algorithm 1). From Lines 1-2 in Algorithm 3, the route $\mathcal{R}$ is initialized by e_s, and timestamp t' is set to be the departure time t_q. Then DRP performs destination-driven route planning from Lines 3 to 14, which terminates at Line 3 if the last segment $\mathcal{R}[-1]$ is the destination segment e_d or the sequence length of $\mathcal{R}$ is too long beyond a threshold parameter L (*e.g.*, 300). Within the while loop, DRP first gets the last segment e_i in $\mathcal{R}$ (Line 4). At Lines 5-6, DRP retrieves all the candidate adjacent segments $e_j \in N_o(e_i)$ with the highest DA strength $da_{\max}$ towards e_d into set E_{cand}. If there is only one element in E_{cand} (Line 8), then the new segment e_{app} to be appended into $\mathcal{R}$ is decided without a tie (Line 9). Otherwise, we have at least two segments in E_{cand}. If so, we further use the traffic popularity $\mathbf{P}$ to break tie: get the time slot $\hat{t'}$ by Eq. (1) (Line 11) and decide e_{app} from E_{cand} by the highest

---

Algorithm 3: DRP (Online)

Input: RPQ $q = \langle s, d, t_q \rangle$ where s and d are on segments e_s and e_d of the input road network G respectively, DA indicator σ, traffic popularity $\mathbf{P}$, max length L
Output: The predicted route $\mathcal{R}$ and arrival time t_a

1 Predicted route $\mathcal{R} \leftarrow \langle e_s \rangle$;
2 Predicted time $t' \leftarrow t_q$;
3 **while** $\mathcal{R}[-1] \neq e_d$ ***and*** *size of* $\mathcal{R} < L$ **do**
4   Segment $e_i \leftarrow \mathcal{R}[-1]$;
5     $da_{\max} \leftarrow \max_{\forall e_j \in N_o(e_i)} \sigma(e_j, e_d)$;
6     $E_{cand} \leftarrow \{e_j \in N_o(e_i) | \sigma(e_j, e_d) = da_{\max}\}$;
7     $e_{app} \leftarrow null$;
8   **if** $|E_{cand}| = 1$ **then**
9         $e_{app} \leftarrow E_{cand}[0]$;
10   **else**
11   Get the time slot $\hat{t'}$ of timestamp t' by Eq. (1) ;
12         $e_{app} \leftarrow \arg\max_{\forall e_j \in E_{cand}} (\mathbf{P}[e_j, \hat{t'}])$;
13     $\mathcal{R}$.append(e_{app});
14     $t' \leftarrow t' + t(e_{app})$;
15 $t_a \leftarrow t'$;
16 **return** $\mathcal{R}$ and t_a;

---

traffic popularity (Line 12). Note that, at Line 12, if more than one segment in E_{cand} has the highest traffic popularity, there is still a tie. In this case, we adopt a simple trick based on cosine similarity to break the tie. Specifically, for every segment e_j in E_{cand}, we regard it as a vector from its entrance to its exit point, compute its cosine similarity with the vector formed from its entrance point to the destination location d, and then choose e_j with the highest cosine similarity as e_{app}. Then at Lines 13-14, DRP appends segment e_{app} to the end of $\mathcal{R}$ and increases timestamp t' by the average travel time $t(e_{app})$ of segment e_{app}. After the while loop terminates, at Line 15, DRP updates the predicted arrival time t_a to be t' and returns $\mathcal{R}$ and t_a at Line 16. DRP in Algorithm 3 predicts at most L steps. Let $\tilde{deg}$ be the max value of $|N_o(e_i)|$ for any e_i in G. In every step, DRP iterates the neighbors of a road segment. The accesses of σ and $\mathbf{P}$ cost amortized constant time. $\tilde{deg}$ is also small, at most 6 in all datasets in experiments. Hence, DRP is efficient to plan routes.

Discussion. Currently we consider static road networks. It is possible to extend DRP to handle road changes, *e.g.*, road closure or new connections. In Algorithm 3, if DRP encounters a closed segment or dead end, before appending e_{app} to $\mathcal{R}$ (Line 13), a technique is to allow DRP *backtrack* one or multiple steps along $\mathcal{R}$ to *restart* the route planning by selecting another segment based on the next largest DA σ. For new segment e_{new}, if a route $\mathcal{R}$ planned by DRP contains a sub-sequence with the same starting and ending points as e_{new}, then it is possible to generate another route $\mathcal{R}'$ by replacing the sub-sequence with e_{new} and also recommend $\mathcal{R}'$ to users. If users adopt $\mathcal{R}'$, then we can obtain new trajectories containing e_{new} to further train our method.

5 KEY SEGMENT DETECTION

It is promising to apply the idea of key segment detection for route planning. However, as mentioned, the challenges are (i) how to properly measure if a segment is a key segment *w.r.t.* an RPQ, and (ii) how to efficiently detect high-quality key segments over massive data. Given an RPQ, its key segment should serve as a hub from its source to destination. Obviously, a segment that is only important to the source or the destination should not be a key segment of the RPQ. Further, trivially regarding segments with higher road types as the key neglects the travel patterns in historical trajectories, and different RPQs should have different key segments.

To deal with these issues, in Section 5.1, we utilize the DA indicator σ to define the importance of a segment *w.r.t.* an SD pair, and develop a series of key segment concepts considering historical trajectories and RPQs as a whole. For every RPQ, we derive a small candidate key segment pool and formulate key segment detection as a binary classification task over the pool. Then in Section 5.2, we present the KSD model consisting of a weighted binary cross entropy loss function and effective encoders, namely query encoder and candidate key segment encoder. Lastly, in Section 5.3, we elaborate how to train KSD and use it for online detection.

5.1 KSD Concepts and Problem Formulation

Given an RPQ $q = \langle s, d, t_q \rangle$ where s and d are on segments e_s and e_d respectively, its key segment should be important to *both* e_s and e_d. Therefore, as the first step, in Definition 5.1, we define the importance score $\omega(e_i, \langle e_s, e_d \rangle)$ of a segment e_i *w.r.t.* SD pair $\langle e_s, e_d \rangle$ by leveraging the DA indicator σ.

Definition 5.1. (Segment Importance *w.r.t.* an SD Pair). Given an SD pair $\langle e_s, e_d \rangle$ and the DA indicator σ built from historical trajectories, the importance of a road segment e_i *w.r.t.* $\langle e_s, e_d \rangle$ is $\omega(e_i, \langle e_s, e_d \rangle) = \min(\sigma(e_s, e_i), \sigma(e_i, e_d))$.

$\omega(e_i, \langle e_s, e_d \rangle)$ takes the minimum of $\sigma(e_s, e_i)$ and $\sigma(e_i, e_d)$, in order to quantify the importance *w.r.t.* both e_s and e_d. If e_i has high DA strengths from e_s to itself and also from itself to e_d, then e_i is with high importance score $\omega(e_i, \langle e_s, e_d \rangle)$. If e_i only has high DA strength with either e_s or e_d, but not both, then e_i is less important to the SD pair. For instance, in Figure 1a, for an SD pair $\langle e_2, e_7 \rangle$, the importance of e_6 is $\omega(e_6, \langle e_2, e_7 \rangle) = \min(\sigma(e_2, e_6), \sigma(e_6, e_7)) = \min(5, 7) = 5$, the importance of e_5 is $\omega(e_5, \langle e_2, e_7 \rangle) = \min(3, 4) = 3$. As a counter example, e_3 has large DA value $\sigma(e_2, e_3) = 8$ from e_2 but is with low $\sigma(e_3, e_7) = 1$ to e_7, and consequently its importance score $\omega(e_3, \langle e_2, e_7 \rangle) = \min(8, 1) = 1$ is low *w.r.t.* the SD pair.

Given an RPQ $q = \langle s, d, t_q \rangle$ with SD pair $\langle e_s, e_d \rangle$, the road segments with higher importance scores are more likely to be the key segment of q, since they have higher DA strengths to both e_s and e_d. Further, as mentioned, a route of an RPQ usually goes through a small part of a road network G, while the remaining part of G is less relevant to the query. It is not necessary to investigate all segments in G for key segment detection. Therefore, in Definition 5.2, given an SD pair $\langle e_s, e_d \rangle$, we define its candidate key segment pool C_{sd} to contain its top-k_c most important segments, and only focus on the pool to detect key segments, where $k_c \ll n$.

Definition 5.2. (Candidate Key Segment Pool of an SD Pair). Given an SD pair $\langle e_s, e_d \rangle$ and the DA indicator σ, the candidate key segment pool C_{sd} of $\langle e_s, e_d \rangle$ contains the top-k_c most important segments e_i ranked by importance scores $\omega(e_i, \langle e_s, e_d \rangle)$, where k_c is the pool size.

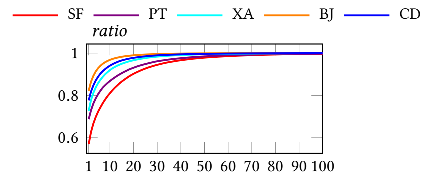


Figure 3: The ratio of trajectories T with candidate key segment pool C_{sd_T} containing at least one segment in $T.seq$, when varying pool size k_c from 1 to 100.

Then a question is how to decide the pool size k_c. Too large k_c will include too many less important segments into the pool C_{sd}, while too small k_c may miss out the true key segments of RPQs. In the following, we provide empirical evidence on how to choose k_c.

Given historical trajectory data $\mathcal{D}$ for training, every trajectory T corresponds to a training RPQ q_T formed by $T.s$, $T.d$, and its departure time $T.t_q$. For every q_T, we get its C_{sd_T} with size k_c. Then we calculate the ratio of trajectories $T \in \mathcal{D}$ with at least one segment in C_{sd_T} that is also in T's segment sequence $T.seq$. Figure 3 reports the ratio when varying k_c from 1 to 100 on the five datasets in experiments, each represented by a line. Observe that the ratio is almost 1 when k_c approaches 100 on all datasets, meaning that, for almost every trajectory $T \in \mathcal{D}$, its candidate key segment pool C_{sd_T} with size 100 contains at least one ground-truth segment from $T.seq$. Therefore, we set k_c to 100 ($\ll n$).

Another observation is that, in Figure 3, when $k_c = 1$, the ratio is not high on all datasets, which indicates that the top-1 most important segment of an RPQ q_T is not necessarily a true key segment really existing in the corresponding trajectory sequence $T.seq$. In other words, during online stage of processing RPQ q, blindly choosing the top-1 important segment of q's SD pair as its key segment is not effective for route planning, which is validated in experiments. Further, in Figure 3, observe that as k_c increases from 1 to 20, the ratio increases significantly, which indicates that the segment importance in Definition 5.1 exhibits positive correlations to the segments actually traveled by historical trajectories.

Remark that not every segment in C_{sd_T} appears in the corresponding trajectory T. In Figure 1b, according to the example DA values, when $k_c = 3$, for an SD pair $\langle e_2, e_7 \rangle$, its candidate pool contains $\{e_6, e_5, e_4\}$ with importance scores $\omega(e_6, \langle e_2, e_7 \rangle) = 5 > \omega(e_5, \langle e_2, e_7 \rangle) = 3 > \omega(e_4, \langle e_2, e_7 \rangle) = 2$. But for a trajectory T_2 with SD pair $\langle e_2, e_7 \rangle$ in Figure 1a, it actually traveled via e_6 and e_4, but not e_5. To facilitate the training of KSD, we define the ground-truth key segments of a historical trajectory T in Definition 5.3.

Definition 5.3. (Ground-truth Key Segments of a Historical Trajectory). Given a trajectory T with $T.s$, $T.d$ and segment sequence $T.seq = \langle e_1, ..., e_\ell \rangle$, after getting its C_{sd_T} by Definition 5.2, the segments in both $T.seq$ and C_{sd_T} are the ground-truth key segments of T. Denote $\mathcal{K}_T$ as the ground-truth key segment set of T.

In Figure 1, for trajectory T_2 with $T_2.seq = \langle e_2, e_4, e_6, e_7 \rangle$ and candidate key segment pool $C_{sd_{T_2}} = \{e_6, e_5, e_4\}$, we can get its ground-truth key segment set $\mathcal{K}_{T_2} = \{e_6, e_4\}$. Since candidate pool size is set to 100 in practice, long trajectories could have $\mathcal{K}_T$ with size close to 100. In this case, we truncate $\mathcal{K}_T$ to size $\lceil 0.2 \cdot T.\ell \rceil$ to exclude low-importance ground-truth key segments, and make $\mathcal{K}_T$ proportional to sequence length $T.\ell$.

Finally, we formulate the Key Segment Detection problem below.

Key Segment Detection (KSD) Problem. KSD is a binary classification task to classify road segments to be or not to be the key segments of RPQs.

(i) Labeled KSD data. For every training trajectory $T \in \mathcal{D}$ with $T.s, T.d, T.t_q$ forming a training RPQ q_T, its candidate key segment pool C_{sd_T} and ground-truth key segment set $\mathcal{K}_T$ are obtained by Definition 5.2 and Definition 5.3. Then every candidate $c \in C_{sd_T}$ has class label $y_c = 1$ if $c \in \mathcal{K}_T$, or class label $y_c = 0$ if $c \notin \mathcal{K}_T$.

(ii) KSD Problem Formulation. Given training trajectory data $\mathcal{D}$ with ground-truth key segment labels, the KSD problem is to train a binary classification model, so that, for an online RPQ $q = \langle s, d, t_q \rangle$ with candidate key segment pool C_{sd}, the model can accurately classify the candidates in C_{sd} into class 1 or 0 *w.r.t.* RPQ q.

5.2 KSD Model Architecture

Then we develop the KSD model illustrated in Figure 4, containing a query encoder, a candidate key segment encoder, and a weighted binary cross entropy loss as the objective.

In a nutshell, given an RPQ $q = \langle s, d, t_q \rangle$, the query encoder will generate a query representation vector $\mathbf{q}$ by considering $\langle e_s, e_d \rangle$, position ratio r_s and r_d, and road network via fully connected layer (FC) and Multi-layer Perceptron (MLP). Meanwhile, based on Definitions 5.1 and 5.2, KSD utilizes the DA strength values $\sigma(e_s, *)$ and $\sigma(*, e_d)$ to obtain the candidate key segment pool C_{sd} of RPQ q. Then C_{sd} is fed into the candidate key segment encoder to output a candidate representation vector $\mathbf{c}_i$ for every $c_i \in C_{sd}$, with the consideration of candidate segment id and traffic popularity of c_i at departure time t_q in RPQ via FC and MLP as well. With representation vectors $\mathbf{q}$ and $\mathbf{c}_i$, KSD computes $P_{key}(c_i|q)$, the probability that c_i is a key segment of RPQ q by Eq. (2). In particular, $P_{key}(c_i|q)$ is the inner product of $\mathbf{c}_i$ and $\mathbf{q}$, normalized by function $sigmoid(x) = \frac{1}{1+exp(-x)}$ to map a value to range $(0, 1)$.

$$P_{key}(c_i|q) = sigmoid(\mathbf{c}_i \cdot \mathbf{q}) \tag{2}$$

Training Objective. During offline training stage, for a trajectory T with candidate pool C_{sd_T} labeled by ground-truth key segment set $\mathcal{K}_T$ and training RPQ q_T, KSD employs a weighted binary cross entropy loss function to evaluate the $loss_T$ of predicting the key segment labels for every $c_i \in C_{sd_T}$, as in Eq. (3). Note that w_{c_i} is the weight of candidate c_i, which is calculated based on the importance score of c_i *w.r.t.* the SD pair of q_T, to be explained shortly.

$$loss_T(\Theta) = - \sum_{\forall c_i \in C_{sd_T}} w_{c_i} \left(y_{c_i} \log P_{key}(c_i|q_T) + (1 - y_{c_i}) \log (1 - P_{key}(c_i|q_T)) \right), \tag{3}$$

where Θ represents the model parameters in KSD, w_{c_i} is the weight of candidate c_i, and y_{c_i} is the class label of c_i in $\{0, 1\}$.

Then the total loss of KSD is the averaged $loss_T$ of all T in $\mathcal{D}$,

$$loss(\Theta) = \frac{1}{|\mathcal{D}|} \sum_{\forall T \in \mathcal{D}} loss_T(\Theta). \tag{4}$$

Recall that every candidate c_i in C_{sd_T} has segment importance score $\omega(c_i, \langle e_s, e_d \rangle)$ (Definition 5.1), which should be considered in the loss function above. Specifically, for every candidate c_i, we assign a weight w_{c_i} computed by Eq. (5). If c_i has class label 1

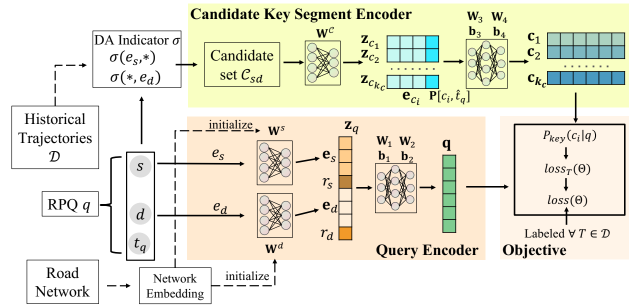


Figure 4: Key Segment Detection Model: KSD

(*i.e.*, ground-truth key segment), w_{c_i} is obtained by exponentially scaling the normalized $\omega(c_i, \langle e_s, e_d \rangle)$, and obviously $w_{c_i} > 1$. If c_i has class label 0, w_{c_i} is set to 1, regardless of its importance score.

$$w_{c_i} = \begin{cases} exp\left(\frac{\omega(c_i, \langle e_s, e_d \rangle)}{\sum_{\forall c_j \in C_{sd_T}} y_{c_j} \cdot \omega(c_j, \langle e_s, e_d \rangle)} \right), & y_{c_i} = 1 \\ 1, & y_{c_i} = 0 \end{cases} \tag{5}$$

Query Encoder. Given an RPQ $q = \langle s, d, t_q \rangle$, we encode it into a query representation $\mathbf{q}$ by considering $\langle e_s, e_d \rangle$, position ratios r_s and r_d, and road network G. Note that departure time t_q is considered in the candidate key segment encoder that is explained later. For segments e_s and e_d, a simple way is to encode their segment ids by n-dimensional one-hot vectors $\mathbf{1}_s$ and $\mathbf{1}_d \in \{0, 1\}^n$, in which all elements are 0, except 1 at the e_s-th and e_d-th dimension respectively. However, such one-hot vector ignores all other information of a segment, *e.g.*, its representation in a road network. Intuitively, a segment e_i influences all its out-going adjacent segments $N_o(e_i)$. The representations of nearby segments should be similar, while that of faraway segments should be less similar. There are classic network embedding techniques to get node representations in graphs, *e.g.*, [14, 28]. We simply adopt Node2Vec [14] as a basic preprocessing step. Node2vec maximizes the likelihood of preserving a node's topological neighborhood via biased random walks with a return likelihood parameter and an in-out parameter controlling depth-first and breath-first explorations respectively. We first convert road network G into its conjugate form where segments e_i are nodes, and an edge exists from e_i to e_j if $e_j \in N_o(e_i)$ in G [42]. Then Node2vec is applied on the conjugate network to learn segment embeddings $\mathbf{W}_G \in \mathbb{R}^{n \times d_0}$ ($d_0 \ll n$) of all segments in G.

Rather than directly using the corresponding rows in $\mathbf{W}_G$ as the representations of e_s and e_d, in the query encoder in Figure 4, we use $\mathbf{W}_G$ to initialize the learnable parameters $\mathbf{W}^s$ and $\mathbf{W}^d$ of two one-layer FCs, which are trained to output the representations $\mathbf{e}_s$ and $\mathbf{e}_d$. Intuitively, a segment should have different semantics when serving as source or destination, and thus, we have separate FCs for source and destination. Specifically, the FC of e_s (resp. e_d) takes as input the one-hot encoding $\mathbf{1}_s$ (resp. $\mathbf{1}_d$), and outputs the representation $\mathbf{e}_s$ (resp. $\mathbf{e}_d$) after trained in the KSD architecture in Figure 4. Eq. (6) shows the formula of the FCs.

$$\mathbf{e}_s = \mathbf{1}_s^\top \mathbf{W}^s; \mathbf{e}_d = \mathbf{1}_d^\top \mathbf{W}^d, \tag{6}$$

where $\mathbf{W}^s, \mathbf{W}^d \in \mathbb{R}^{n \times d_0}$ are the learnable parameters.

Then we generate the intermediate representation of RPQ q, $\mathbf{z}_q = concat(\mathbf{e}_s, r_s, \mathbf{e}_d, r_d)$ by further concatenating position ratios

r_s of s on e_s and r_d of d on e_d. $\mathbf{z}_q \in \mathbb{R}^{2d_0+2}$ is then fed into a two-layer MLP to generate the final query representation $\mathbf{q} \in \mathbb{R}^{d_2}$ shown in Eq. (7). Briefly, MLP is a fully connected feed-forward neural network and it often combines with nonlinear activation function (*e.g.*, $ReLU(x) = \max(0, x)$) to bring non-linearity into the model, to alleviate the vanishing gradient problem [13].

$$\mathbf{q} = \mathbf{W}_2 ReLU(\mathbf{W}_1 \mathbf{z}_q + \mathbf{b}_1) + \mathbf{b}_2, \tag{7}$$

where $\mathbf{W}_1 \in \mathbb{R}^{d_1 \times (2d_0+2)}$ and $\mathbf{b}_1 \in \mathbb{R}^{d_1}$ are the parameters of the first layer in MLP, and $\mathbf{W}_2 \in \mathbb{R}^{d_2 \times d_1}$ and $\mathbf{b}_2 \in \mathbb{R}^{d_2}$ are the parameters of the second layer, and the output is $\mathbf{q} \in \mathbb{R}^{d_2}$.

Candidate Key Segment Encoder. Then we develop the encoder that generates a representation $\mathbf{c}_i$ for every $c_i \in C_{sd}$ as shown in Figure 4. This encoder contains a one-layer FC and a two-layer MLP. In particular, for every $c_i \in C_{sd}$, the FC transforms its one-hot id vector $\mathbf{1}_{c_i}$ to a dense representation $\mathbf{e}_{c_i}$ via learnable weights $\mathbf{W}^C$,

$$\mathbf{e}_{c_i} = \mathbf{1}_{c_i} \mathbf{W}^C, \tag{8}$$

where $\mathbf{W}^C \in \mathbb{R}^{n \times d_3}$ contains the learnable parameters.

As shown in Figure 4, then for every $c_i \in C_{sd}$, we get its $\mathbf{z}_{c_i}$ by concatenating the traffic popularity of c_i at the departure time t_q in RPQ q with $\mathbf{e}_{c_i}$, so that the traffic popularity of a candidate key segment at the departure time is also considered. Specifically, after getting the time slot $\hat{t}_q$ of t_q by Eq. (1), we get $\mathbf{P}[c_i, \hat{t}_q]$ of all $c_i \in C_{sd}$, apply min-max normalization to convert the values to into range $[0, 1]$, and then concatenate to get $\mathbf{z}_{c_i}$. All $\mathbf{z}_{c_i}$ of $c_i \in C_{sd}$ are then fed into a two-layer MLP (Eq.(9)) to get the final candidate representation $\mathbf{c}_i \in \mathbb{R}^{d_2}$, with the same dimension as the query encoding $\mathbf{q}$ in Eq. (7).

$$\mathbf{c}_i = \mathbf{W}_4 ReLU(\mathbf{W}_3 \mathbf{z}_{c_i} + \mathbf{b}_3) + \mathbf{b}_4, \tag{9}$$

where $\mathbf{W}_3 \in \mathbb{R}^{d_4 \times (d_3+1)}$, $\mathbf{b}_3 \in \mathbb{R}^{d_4}$, $\mathbf{W}_4 \in \mathbb{R}^{d_2 \times d_4}$, and $\mathbf{b}_4 \in \mathbb{R}^{d_2}$ are the parameters of the MLP.

Finally, we obtain the query representation $\mathbf{q}$ and the candidate key segment representations $\mathbf{c}_i, \forall c_i \in C_{sd}$ in Figure 4. As explained, we then calculate the key segment probability $P_{key}(c_i|q)$ by Eq. (2), get $loss_T$ by Eq. (3) and subsequently total loss in Eq. (4) over historical trajectories $\mathcal{D}$, in order to train the whole KSD model, as elaborated in the following section.

5.3 KSD Offline Training and Online Inference

Model Training. We adopt mini-bath training in epochs and employ Adam Optimizer [20] to train the parameters Θ of KSD (*i.e.*, Eq. (6), (7), (8), and (9)) over the loss functions in Eq. (3) and (4). Algorithm 4 presents the training process of KSD. The input includes a historical trajectory dataset $\mathcal{D}$ for training, the corresponding DA indicator σ, learning rate lr, number of training epochs I, and batch size bs. In a nutshell, from Lines 1 to 8, we generate training samples $\mathcal{U}$, *i.e.*, labeled KSD data (Section 5.1); then from Lines 9 to 19, we train KSD over $\mathcal{U}$ with the loss function and the encoders of KSD (Section 5.2). Specifically, for each trajectory T (Line 2), we regard its $T.s$, $T.d$, $T.t_q$ as a training RPQ q_T (Line 3), get its candidate key segment pool C_{sd_T} with class labels labeled by its ground-truth key segments $\mathcal{K}_T$ (Lines 4-6), and compute the weight w_{c_i} of candidate c_i (Line 7). The training sample generated by T is inserted into $\mathcal{U}$ at Line 8. Then, after initializing all parameters at Line 9, we train KSD with at most I epochs (Line 10) until training loss convergence. In every epoch, we randomly shuffle and split $\mathcal{U}$ into batches (Line 11). In every batch, we handle every training sample $(q_T, C_{sd_T}, \mathcal{K}_T)$ by forward model execution to get predicted key segment probabilities (Lines 12-14) and get $loss_T$ by Eq. (3) at Line 15. The total loss of the batch is obtained by Eq. (4) at Line 16. Then the model parameters are updated via Adam Optimizer at Lines 17-18. After training, Algorithm 4 returns the trained KSD model with optimized parameters Θ at Line 19.

Online Inference. In the online stage, given an RPQ $q = \langle s, d, t_q \rangle$, KSD first gets its C_{sd} using the DA indicator σ, and then conducts forward execution as shown in Figure 4 to get the predicted key segment probability $P_{key}(c_i|q)$ of every candidate $c_i \in C_{sd}$. Note that KSD only outputs the key segment probabilities of the candidate segments in C_{sd}. This online inference of KSD is invoked at Line 1 of Algorithm 1 (DRPK) in Section 3.

Algorithm 4: KSD Training Algorithm

Input: Training trajectory data $\mathcal{D}$ and its DA indicator σ, learning rate lr, training epochs I, batch size bs
Output: KSD model

```
1  Training samples 𝒰 ← ∅;
2  foreach T ∈ 𝒟 do
3      Training RPQ q_T ← ⟨T.s, T.d, T.t_q⟩;
4      Get candidate key segment pool C_{sd_T} of q_T by Definition 5.2;
5      Get ground-truth key segments 𝒦_T by Definition 5.3;
6      ∀c_i ∈ C_{sd_T}, if c_i ∈ 𝒦_T, y_{c_i} ← 1; else y_{c_i} ← 0;
7      ∀c_i ∈ C_{sd_T}, get w_{c_i} by Eq. (5);
8      Insert training sample (q_T, C_{sd_T}, 𝒦_T) into 𝒰;
9  Initialize parameters in Eq. (6), (7), (8), and (9);
10 for i ← 1, 2, ..., I do
11     Shuffle and split 𝒰 into batches ℬ;
12     foreach batch in ℬ do
13         foreach training sample (q_T, C_{sd_T}, 𝒦_T) in batch do
14             Forward model execution to get P_key(c_i|q_T) of all
                 candidate c_i ∈ C_{sd_T};
15             Get loss_T by Eq. (3);
16         Get loss of the batch by Eq. (4);
17         Δθ ← AdamOpt(Θ, loss, lr);
18         Update model parameters Θ with Δθ;
19 return KSD model with Θ;
```

Table 2: Dataset Statistics

	San Francisco City (SF)	Porto District (PT)	Xi'an (XA)	Beijing (BJ)	Chengdu (CD)
# of trajectories	314,507	1,258,165	2,419,072	3,100,845	3,887,769
Avg # of segments	34.15	49.44	24.59	18.57	22.31
Avg length (m)	3,659.98	6,185.39	4,752.08	4,033.85	4,299.19
Avg travel time (s)	563.53	718.07	834.14	486.60	648.32
Time interval	2008/5/17-2008/6/10	2013/7/1-2014/6/30	2016/10/1-2016/10/31	2009/3/2-2009/3/25	2016/10/1-2016/10/31
# of segments	26,659	185,074	59,927	311,769	107,655
# of intersections	9,581	78,054	26,979	125,558	46,008

6 EXPERIMENTS

All experiments are conducted on a Linux machine powered by Intel Xeon® Gold 6226R 2.90GHz CPU and NVIDIA GTX 3090 GPU

Table 3: Overall Effectiveness Evaluation: Precision, Recall, F1, Jaccard Scores. (Best is in bold, runner up is underlined.)

	Precision							Recall						
Data	Short	Fast	CSSRNN	DeepMove	NASR	NMLR	DRPK	Short	Fast	CSSRNN	DeepMove	NASR	NMLR	DRPK
SF	0.515	0.589	0.408	0.472	0.532	<u>0.600</u>	**0.658**	0.491	0.574	0.542	0.543	0.471	<u>0.596</u>	**0.650**
PT	0.601	0.716	0.736	0.649	0.738	<u>0.763</u>	**0.791**	0.539	0.671	<u>0.740</u>	0.666	0.632	0.738	**0.757**
XA	0.693	0.784	0.834	0.741	0.766	<u>0.835</u>	**0.846**	0.667	0.763	<u>0.824</u>	0.772	0.744	0.822	**0.831**
BJ	0.791	0.817	0.786	0.450	0.734	<u>0.842</u>	**0.865**	0.749	0.779	0.803	0.588	0.674	<u>0.823</u>	**0.839**
CD	0.733	0.810	0.865	0.764	0.798	<u>0.866</u>	**0.876**	0.700	0.783	<u>0.850</u>	0.794	0.773	0.848	**0.856**

	F1-score							Jaccard						
Data	Short	Fast	CSSRNN	DeepMove	NASR	NMLR	DRPK	Short	Fast	CSSRNN	DeepMove	NASR	NMLR	DRPK
SF	0.500	0.578	0.430	0.486	0.476	<u>0.593</u>	**0.650**	0.410	0.486	0.362	0.408	0.397	<u>0.509</u>	**0.567**
PT	0.559	0.683	0.727	0.644	0.643	<u>0.741</u>	**0.765**	0.469	0.608	0.668	0.577	0.579	<u>0.681</u>	**0.707**
XA	0.678	0.772	<u>0.827</u>	0.748	0.752	<u>0.827</u>	**0.836**	0.591	0.702	<u>0.776</u>	0.681	0.684	0.774	**0.785**
BJ	0.764	0.792	0.780	0.465	0.676	<u>0.825</u>	**0.846**	0.698	0.728	0.733	0.424	0.612	<u>0.775</u>	**0.798**
CD	0.714	0.794	<u>0.855</u>	0.769	0.782	<u>0.855</u>	**0.864**	0.634	0.731	<u>0.809</u>	0.708	0.721	0.808	**0.818**

with 24GB video memory. Our methods are implemented in Python 3.8 with PyTorch 1.13 and C++. Source codes of all competitors are obtained from the respective authors in Python.

6.1 Experimental Setup

Datasets. Table 2 lists the statistics of the 5 real-world trajectory data on San Francisco City (SF) [29], Porto District (PT) [3], Beijing (BJ) [31], Xi'an (XA) and Chengdu (CD) [1]. Map-matching method in [41] is used. SF, PT, and BJ contain taxi trajectories; XA and CD contain trajectories by DiDi ride-sharing. There are millions of trajectories in 4 out of 5 datasets. Table 2 also provides the average number of segments, length, travel time of trajectories, and the number of segments and intersections.

Training, Validation and Test Data. We randomly split a trajectory dataset into training, validation, testing with ratio in 60%, 20%, and 20%. Training data is used to train models. Validation data is used to select model parameters. Test data serves as ground truth in the online stage to evaluate the predicted routes of testing RPQs.

Competitors. Following latest studies [17, 34], we compare with 6 competitors, including NMLR [17], CSSRNN [37], NASR [34], DeepMove [11], Short to get the shortest path, and Fast to get the fastest path of an RPQ. The codes of [12, 15] are unavailable. Studies [7, 24, 25] are subsumed by the competitors [17, 34].

Parameter Settings. In KSD model, we set candidate key segment pool size $k_c = 100$ as analyzed in Section 5.1. We set dimension $d_0 = 64$ in Eq. (6), $d_2 = 256$ in Eq. (7) and Eq. (9), and $d_3 = 64$ in Eq. (8). As for the hidden dimension of MLP, we set $d_1 = 2048$ in Eq. (7) and $d_4 = 512$ in Eq. (9). In KSD, the learning rate (lr) is 1e-3 and the batch size is 8,192. The number of time slots n_t is 48. The max length parameter $L = 300$ [17] and the number of training epochs 300 are applied to all methods. For competitors, we follow their suggested settings in their papers to tune optimal parameters and train all competitors until convergence. NMLR adopts Lipschitz embedding, 2-layer GCN, and 3-layer MLP with hidden dimension 256, and is trained using Adam [20] with lr 1e-3. CSSRNN uses LSTM with 512 hidden dimension and dropout rate 0.1, and is trained using RMSProp [32] with lr 1e-4 and decay rate 0.9. DeepMove inputs location and time embedding into GRU with 256 hidden size, and is trained by Adam with lr 1e-3, and L2 penalty with 1e-5 is applied as suggested. NASR feeds location and time embedding into LSTM with 256 hidden size, and the number of multi-head self attention layers is 3 with 6 heads, and it is trained using Adam with lr 1e-4.

Evaluation Metrics. (i) *Effectiveness metrics.* Given an RPQ q, let $\mathcal{R}$ and $\mathcal{R}^*$ be the predicted route by a method and the ground-truth route, respectively. We use 5 popular metrics, Precision (*pre*), Recall (*rec*), F1-score (*F1*), Jaccard (*jac*), and reachability, with formula shown below, all of which are the higher the better [17, 34]. Segment length $l(e)$ is used as weights. In particular, given an RPQ, recall (*rec*) evaluates the proportion of overlapping length of the predicted route $\mathcal{R}$ over the ground truth $\mathcal{R}^*$. Reachability is the ratio of testing RPQs with predicted routes arriving at their destinations. The metrics below are averaged on all testing RPQs.

$$pre(\mathcal{R}^*, \mathcal{R}) = \frac{\sum_{e\in(\mathcal{R}\cap\mathcal{R}^*)} l(e)}{\sum_{e\in\mathcal{R}} l(e)} \qquad rec(\mathcal{R}^*, \mathcal{R}) = \frac{\sum_{e\in(\mathcal{R}\cap\mathcal{R}^*)} l(e)}{\sum_{e\in\mathcal{R}^*} l(e)}$$

$$F1(\mathcal{R}^*, \mathcal{R}) = \frac{2pre(\mathcal{R}^*, \mathcal{R})rec(\mathcal{R}^*, \mathcal{R})}{pre(\mathcal{R}^*, \mathcal{R}) + rec(\mathcal{R}^*, \mathcal{R})} \qquad jac(\mathcal{R}^*, \mathcal{R}) = \frac{\sum_{e\in(\mathcal{R}\cap\mathcal{R}^*)} l(e)}{\sum_{e\in(\mathcal{R}\cup\mathcal{R}^*)} l(e)}$$

(ii) *Efficiency metrics.* We evaluate each method by the number of online RPQs processed per second (QPS for short), total training time, and training time per epoch.

6.2 Effectiveness Evaluation

Overall Effectiveness. Table 3 reports the precision, recall, F1, and Jaccard scores of all methods on all datasets. An overall observation is that our method DRPK consistently achieves the highest performance on all datasets under all evaluation metrics, outperforming existing methods often by a substantial margin. For instance, on SF dataset, DRPK achieves precision 0.658, while the precision of the best competitor NMLR is 0.600, indicating 5.8% absolute improvement, which is significant in terms of effectiveness. The recall of DRPK on SF is 0.650, improving 5.4% over NMLR with recall 0.596. On a large dataset BJ, DRPK has the highest Jaccard score, 0.798, 2.3% higher than the best Jaccard 0.775 from NMLR. The evaluation scores of almost all methods on XA and CD are relatively high. Nevertheless, DRPK still outperforms existing methods on XA and CD. The results in Table 3 demonstrate the effectiveness of DRPK powered by the DA indicator σ, DRP procedure, and KSD model. Besides, the recall of Short on SF is 0.491, meaning that on average the overlapping length of the shortest paths over the corresponding real trajectories is only 49.1%, which demonstrates that real trajectories do not always follow shortest paths.

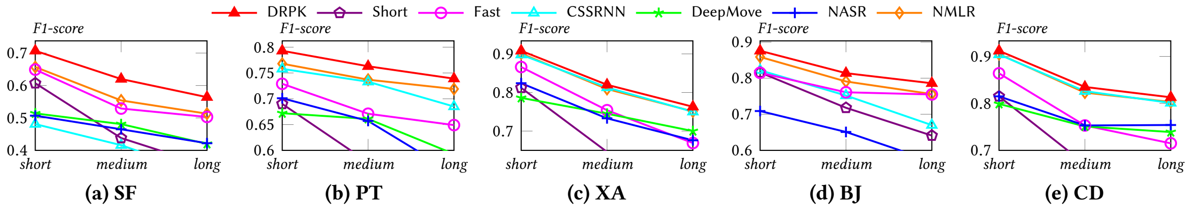


Figure 5: F1-score on short, medium, and long RPQs (low F1 data points are truncated for clarity)

Table 4: Average SD Distance of Testing RPQs in meters

	SF	PT	XA	BJ	CD
All	2534.78	3539.12	2987.01	2358.83	2696.22
NC-RPQs	2656.16	3795.06	3354.10	2802.87	3234.85
C-RPQs	2019.03	3372.10	2892.37	1825.37	2591.17

Table 5: F1 on hard NC-RPQs with Not Covered SD Pairs

Data	Short	Fast	CSSRNN	DeepMove	NASR	NMLR	DRPK
SF	0.477	0.561	0.390	0.453	0.462	<u>0.571</u>	**0.624**
PT	0.509	0.629	0.621	0.522	0.549	<u>0.671</u>	**0.692**
XA	0.558	0.666	0.721	0.620	0.630	<u>0.727</u>	**0.747**
BJ	0.678	0.716	0.664	0.307	0.579	<u>0.745</u>	**0.776**
CD	0.565	0.637	0.718	0.583	0.643	<u>0.727</u>	**0.747**

Table 6: F1 on C-RPQs with Covered SD Pairs

Data	Short	Fast	CSSRNN	DeepMove	NASR	NMLR	DRPK
SF	0.609	0.662	0.627	0.650	0.544	<u>0.704</u>	**0.777**
PT	0.596	0.723	<u>0.804</u>	0.734	0.714	0.793	**0.819**
XA	0.708	0.798	<u>0.854</u>	0.780	0.782	0.852	**0.859**
BJ	0.867	0.883	0.920	0.656	0.794	<u>0.921</u>	**0.930**
CD	0.744	0.826	<u>0.883</u>	0.807	0.810	0.881	**0.888**

Table 7: Reachability (%)

Data	CSSRNN	DeepMove	NASR	NMLR	DRPK
SF	64.00	73.76	57.90	<u>92.72</u>	**99.85**
PT	94.91	81.40	72.67	<u>98.37</u>	**99.65**
XA	99.66	98.62	89.19	<u>99.87</u>	**100.0**
BJ	92.85	48.07	79.16	<u>98.84</u>	**99.95**
CD	99.72	97.70	86.84	<u>99.88</u>	**100.0**

Table 8: Online QPS (# of RPQs processed per second)

	Shortest Path Processing				Learning-based Route Planning				
Data	Short	$\text{Short}_{\text{BCH}}$	Fast	Fast_{BCH}	CSSRNN	DeepMove	NASR	NMLR	DRPK
SF	916.6	8787.7	776.2	15236.1	23.8	11.7	2.08	1750.6	3569.9
PT	296.6	19207.5	191.1	30809.1	97.2	16.9	2.56	1266.9	2768.6
XA	2033.4	22541.1	1748.3	32886.8	198.3	20.6	9.92	3433.1	6070.2
BJ	1371.9	22737.3	1498.2	59725.7	104.7	3.44	2.24	2266.5	4085.1
CD	1773.5	15590.9	1906.6	24408.1	304.8	18.5	11.4	3622.2	5618.8

Impact of SD Distance. We then categorize testing RPQs into three groups based on their SD Euclidean distance: *short* (less than 2km), *medium* (2km to 4km) and *long* (more than 4km). Figure 5 displays the F1-scores of these groups (low F1 data points are truncated for clarity). The first important observation is that all methods have *better performance on RPQs with shorter distances*, which validates the rationale in DRPK to break an RPQ into shorter RPQs and motivates the design of KSD. In Figure 5, DRPK consistently outperforms competitors for short, medium and long RPQs, validating the effectiveness of DRPK regardless of SD distance.

Hard RPQs with Not Covered SD Pairs. In real world, future RPQs may have their source and destination segments covered or not covered by training data. Similar to [17], given all testing RPQs $q = \langle s, d, t_q \rangle$ with SD pairs $\langle e_s, e_d \rangle$, we separate into two categories: NC-RPQs with $\langle e_s, e_d \rangle$ not covered by the SD pairs of any training RPQs and C-RPQs with $\langle e_s, e_d \rangle$ covered. Intuitively, NC-RPQs are hard to handle. Table 4 reports the average SD Euclidean distance of all testing RPQs, testing NC-RPQs, and C-RPQs, respectively. Observe that NC-RPQs are usually with longer distances, making them even harder to be accurately predicted as shown in Figure 5. Table 5 reports the F1 scores of all methods on hard NC-RPQs. First, the performance gap between DRPK and the competitors on the hard NC-RPQs maintains, or enlarges especially on XA, BJ, and CD. For instance, in Table 5, on XA, DRPK has F1 0.747, 2% higher than NMLR with 0.727, while the overall F1 of DRPK and NMLR in Table 3 are 0.836 and 0.827. Table 5 illustrates the power of DRPK to handle RPQs with unseen SD pairs. Second, NC-RPQs are usually with longer distance (Table 4), so the F1 of Short and Fast on NC-RPQs also drop in Table 5, compared with Table 3. We report the F1 on C-RPQs in Table 6, where DRPK is the best.

Reachability. As showed in [17], some methods have a significant portion of predicted routes failing to reach destinations. We also evaluate the reachability of predicted routes in Table 7. DRPK always has the highest close-to-1 reachability on all datasets, due to the destination-driven procedure and the adoption of key segments as hubs, while existing methods suffer from low reachability. For instance, NMLR has reachability 92.72% on SF, probably due to insufficient historical trajectories to train its model. CSSRNN also has low reachability on SF, PT, and BJ. The reachability of Short and Fast is always 1, but they have inferior effectiveness in Table 3.

6.3 Efficiency Evaluation

Online QPS. During online phase, we evaluate the number of RPQs processed per second (QPS). In Table 8, there are 2 classes of methods: learning-based methods and shortest path processing methods that include Short and Fast adopt vanilla Dijkstra and $\text{Short}_{\text{BCH}}$ and Fast_{BCH} adopt well-established Bidirectional Dijkstra with Contraction Hierarchies [10]. Observe that, compared with learning-based competitors, DRPK has higher QPS. Meanwhile, $\text{Short}_{\text{BCH}}$ and Fast_{BCH} with dedicated shortest path techniques have unrivaled efficiency. Considering the effectiveness results reported ahead, we conclude that DRPK consistently achieves highest route planning accuracy, and is more efficient than learning-based competitors.

Offline Training Efficiency. Table 9 reports the total training time in hours. DRPK is much faster to train than the competitors, by up to orders of magnitude. For instance, on PT, DRPK takes 0.65 hour to train, 10.7× faster than NMLR that requires 7.59 hours, while the other competitors cost more than 24 hours. We then break

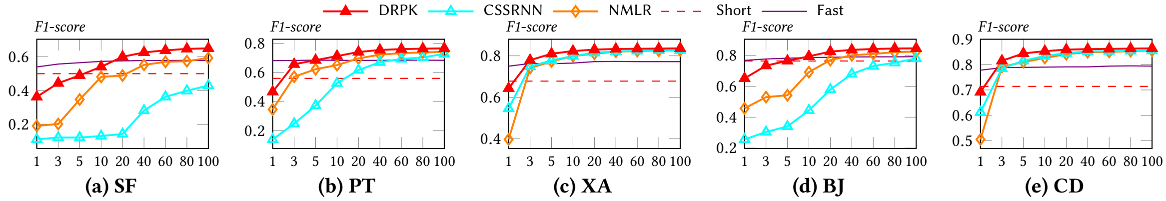


Figure 6: F1-score When Varying Training Data Size from 1% to 100%

Table 9: Offline Training Time (hours)

Data	CSSRNN	DeepMove	NASR	NMLR	DRPK
SF	5.03	2.29	6.34	3.25	**0.16**
PT	33.43	26.39	81.05	7.59	**0.65**
XA	9.29	20.42	20.59	4.62	**1.08**
BJ	54.34	99.55	97.06	8.74	**1.34**
CD	13.79	48.55	31.29	6.64	**2.17**

Table 10: DRPK Training Time Breakdown (seconds)

Component	SF	PT	XA	BJ	CD
DA indicator σ	9.18	106.5	32.8	45.5	63.3
Traffic Popularity	0.36	1.99	1.94	2.45	2.73
KSD Training	562.7	2324.5	3892.9	4833.8	7814.6

Table 11: Training Time per Epoch (seconds)

Data	CSSRNN	DeepMove	NASR	NMLR	DRPK
SF	60.35	161.53	292.75	233.62	**4.47**
PT	401.17	1895.09	4732.52	546.39	**14.33**
XA	278.78	1497.62	1681.08	332.53	**28.94**
BJ	652.14	7057.28	5797.68	628.89	**32.89**
CD	413.98	3760.87	1878.92	478.31	**71.31**

Table 12: F1-score of Ablation Study

Variant	SF	PT	XA	BJ	CD
DRPK	**0.650**	**0.765**	**0.836**	**0.846**	**0.864**
DRPK-Top1	0.624	0.705	0.789	0.837	0.826
DRP	0.634	0.737	0.827	0.840	0.855
DRP-TP	0.607	0.730	0.826	0.834	0.853

Table 13: Recency Evaluation on PT

	Short	Fast	CSSRNN	DeepMove	NASR	NMLR	DRPK
Precision	0.597	0.711	0.689	0.612	0.729	0.756	**0.781**
Recall	0.536	0.667	0.711	0.636	0.620	0.732	**0.750**
F1	0.556	0.679	0.685	0.609	0.631	0.735	**0.757**
Jaccard	0.467	0.604	0.624	0.540	0.566	0.673	**0.697**

down the training time of DRPK in Table 10, including the time of constructing DA indicator σ at Lines 1-5 of Algorithm 2 (Section 4.1), traffic popularity at Lines 6-10 of Algorithm 2 (Section 4.2), and KSD training time (Algorithm 4). First, the build of DA indicator σ and traffic popularity is highly efficient. For example, on CD with millions of trajectories, DA indicator construction only takes 63.3 seconds, and traffic popularity is built in 2.73 seconds. The build of σ on PT takes 106.5 seconds since the average number of segments per trajectory in PT is 49.44 (Table 2), but it is still fast. Second, training KSD is the main overhead in DRPK. Nevertheless, as reported in Table 9, DRPK needs shortest total training time. We further evaluate the training time per epoch of KSD in DRPK and the baselines in Table 11, where our method costs tens of seconds per epoch, while the competitors require hundreds/thousands of seconds per epoch. For instance, on BJ, on average DRPK requires 32.89*s* to train an epoch in KSD, while competitor NMLR needs 628.89*s*. The reason is that KSD only involves elementary learning components (FC and MLP) that are fast to train in Section 5.2.

6.4 Model Analysis

Ablation Study. We ablate the techniques in DRPK with F1 results in Table 12. The F1 of DRPK is at the first row. In the second row, DRPK-Top1 is DRPK without KSD but with blind choice of top-1 candidate in C_{sd} as the key segment e_{key} (Section 5.1). DRPK-Top1 has lower F1 than DRPK, proving that KSD (Section 5.2) is effective to detect high-quality key segments, and improves the performance of DRPK. The F1 of sub-procedure DRP is in the third row. First, DRP achieves comparable F1 compared with the best baseline NMLR in Table 3. Compared with NMLR, DRP has lower F1 on PT, same F1 on XA and CD, and higher F1 on SF and BJ. This validates the effectiveness of the DA indicator in Section 4.1 to preserve the associations between segments from historical trajectories. Further, DRPK is better than DRP on all datasets (*e.g.*, 0.765 v.s. 0.737 on PT) in Table 12, showing the importance of KSD to assist DRPK for the best performance in Table 3. In the last row of Table 12, DRP-TP is DRP without the traffic popularity **P** in Section 4.2, which has lower F1 than DRP, indicating that the traffic popularity at the departure time t_q in $q = \langle s, d, t_q \rangle$ is also helpful.

Robustness v.s. Amount of Training Data. We evaluate the robustness of DRPK when varying the amount of training data to be 1%, 3%, 5%, 10% 20%, 40%, 60%, 80% and 100% of the total training data, and report F1 scores in Figure 6, in which the two strong competitors CSSRNN and NMLR, as well as Short and Fast, are also included for comparison. Observe that, as the amount of training data reduces, all learning-based models have degraded F1 scores, but the performance gap of DRPK over NMLR and CSSRNN maintains or enlarges from 100% to 1%. Under most settings, DRPK is also better than Short and Fast. DRPK requires about minimum of 5%, 3%, 3%, 5%, 3% (resp. 15%, 5%, 3%, 10%, 3%) of training data on SF, PT, XA, BJ, and CD respectively, to be better than Short (resp. Fast), which is interesting to help decide when to switch between DRPK and shortest/fastest paths.

Recency Evaluation. We split all trajectories in PT based on the *chronological order* into training, validation, and testing with ratio 6:2:2 to test the effect of recency, with results in Table 13. DRPK achieves higher performance than existing methods on all metrics. Compared with the results on PT in Table 3, all methods have slightly lower scores under the recency setting. We further split the training data above into 4 splits by chronological order, use the *i*-th split to train DRPK-*i*, and use the same testing data above to evaluate the performance in Table 14. A slight performance increase is observed from DRPK-1 to DRPK-4 that is more recent to testing.

Table 14: 4-Split Recency on PT

	DRPK-1	DRPK-2	DRPK-3	DRPK-4
Precision	0.741	0.742	0.746	0.748
Recall	0.720	0.721	0.724	0.726
F1	0.721	0.722	0.726	0.727
Jaccard	0.658	0.659	0.664	0.666

Table 15: Train on BJ data and test on T-Drive data

	Short	Fast	CSSRNN	DeepMove	NASR	NMLR	DRPK
Precision	0.740	0.806	0.757	0.421	0.688	0.820	**0.841**
Recall	0.605	0.783	0.785	0.552	0.658	0.802	**0.828**
F1	0.666	0.789	0.754	0.430	0.651	0.802	**0.828**
Jaccard	0.499	0.736	0.712	0.395	0.595	0.756	**0.783**

Generalization to Unseen Data. In addition to the experiments on NC-RPQs (Table 5) and recency evaluation (Tables 13, 14), we develop another experiment on generalization. Specifically, the BJ data is from [31]. We find another trajectory dataset T-Drive on the same city, but from a *different data source* [43], which can be regarded as unseen data. For all learning-based methods, we use their models trained on BJ data to test on T-Drive with 208,951 testing RPQs, with results in Table 15, where DRPK still achieves the highest accuracy under all metrics.

6.5 Extension to Multiple Key Segments

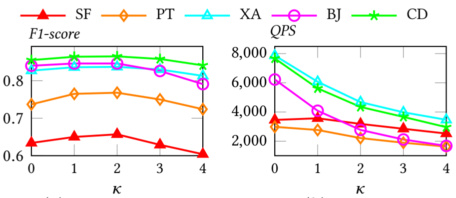
In Algorithm 1, DRPK uses KSD only once to break an RPQ q to RPQs q_1 and q_2. A natural question is: what about applying KSD multiple times to split q to multiple RPQs? Here we extend DRPK with multiple key segments and evaluate the performance.

Let a parameter κ specify the number of key segments to be used. For an RPQ q, let Q be the list of RPQs split from q by key segments. Initially, $Q = \{q\}$. In every iteration of a while loop, among all RPQs in Q, DRPK pops from Q the RPQ q' with the largest SD Euclidean distance, applies KSD to detect a key segment of q', and splits q' into two RPQs q'_1 and q'_2 that are then inserted into Q. In every iteration, KSD is applied once and the size of Q increases by 1. The while loop terminates when $|Q| = \kappa + 1$. Lastly, DRPK applies the DRP procedure to get the routes of all RPQs in Q and concatenate these routes together as the final route of the input RPQ q.

We vary κ in $\{0, 1, 2, 3, 4\}$. Figure 7a reports the F1 scores and Figure 7b displays the QPS. In Figure 7a, F1 scores increase when κ is from 0 to 2, and then drop as $\kappa = 3, 4$. The reasons are as follows. In real world, a route may not necessarily contain a lot of key segments as hubs, especially for short routes. If a segment is wrongly predicted as a key segment, this could bring divergence to the route planning process, and the excessive usage of key segments (*e.g.*, $\kappa \geq 3$) will accumulate this impact, offsetting the benefit brought by key segments. In Figure 7b, the QPS decreases as KSD is invoked more times. Thus, we set $\kappa = 1$ in DRPK as default to strike the balance between route quality and online efficiency. It is also an option to set $\kappa = 2$, especially on SF and PT, if the application is insensitive to QPS but requires high F1 scores.

7 RELATED WORK

We review other related work here, except [11, 17, 34, 37] reviewed in Section 2.2. There are studies formulating route planning as path finding or graph search problems [7, 9, 19, 26, 36]. An A* method



(a) F1-Scores (b) QPS

Figure 7: DRPK with Multiple Key Segments per RPQ

in [19] finds the fastest paths with speed patterns. Luo *et al.* [26] design a path finding query to retrieve the most frequent paths from historical trajectories. Wei *et al.* develop a framework to construct popular routes from uncertain trajectories [36]. Chen *et al.* [7] find most popular routes by Markov Chain and breath-first search. A personalized method considers travel costs and driver preferences via a series of filters in [9]. With given road costs, a work [27] finds diverse top-k paths. Note that the studies [9, 19, 26] mainly focus on efficient graph search with heuristic cost functions, and some studies consider different settings with given travel costs [19, 27].

Another trend is to adopt either conventional machine learning techniques [6, 15, 40] or deep learning models [11, 12, 17, 34, 37], to capture the patterns in trajectories for route planning. A spatio-temporal hidden Markov model [40] is used to model correlations among different traffic time series to infer travel costs. In [15], Guo *et al.* design a trajectory-based clustering technique and a modified Dijkstra algorithm to accommodate multiple preferences for route planning. Chen *et al.* [6] adopt Markov chain to combine the ranking and transition probabilities of places to recommend trajectories. Recent deep learning methods commonly use sequential/graph neural networks for trajectory modeling [11, 12, 17, 22, 24, 25, 34, 37], *e.g.*, RNNs also used in [12, 24, 25]. Methods in [24, 25] are subsumed by NMLR and NASR, while the code of [12] is not available.

There are orthogonal studies, *e.g.*, travel time estimation [42], trajectory similarity [35], route recovery [4, 38], and destination prediction [39, 44]. They, together with this study, demonstrate the value to leverage historical trajectories for spatial data management.

8 CONCLUSION

We present DRPK, a novel method for route planning using historical trajectories. The main designs in DRPK include the DA indicator σ, the destination-driven module DRP, and the key segment detection model KSD. We develop a set of thoughtful key segment concepts that holistically consider σ, trajectories, and road networks. Then KSD is formulated as a classification model that is efficient for offline training and online inference. Extensive experiments demonstrate the superiority of DRPK. We plan to study route planning with changes, *e.g.*, road closures and new connections. We will also investigate peak-hour route planning with heavy traffic.

ACKNOWLEDGMENTS

This work is supported by Hong Kong RGC ECS No. 25201221, and NSFC No. 62202404. This work is also supported by a collaboration grant from Tencent Technology (Shenzhen) Co., Ltd (P0039546). This work is supported by Singapore MOE Tier 1 Seed Funding (RS05/21) and Tier 2 Grant (MOE-T2EP20122-0003). This work is partially supported by NSFC No. 62002303, 61772492, 62072428.

REFERENCES

[1] 2022. Gaya. https://outreach.didichuxing.com/research/opendata/.

[2] 2022. OpenStreetMap. https://www.openstreetmap.org/.

[3] 2022. Porto Dataset. https://www.kaggle.com/c/pkdd-15-predict-taxi-service-trajectory-i/data.

[4] Prithu Banerjee, Sayan Ranu, and Sriram Raghavan. 2014. Inferring Uncertain Trajectories from Partial Observations. In *ICDM*. 30–39.

[5] Jean Bourgain. 1985. On Lipschitz embedding of finite metric spaces in Hilbert space. *Israel Journal of Mathematics* 52, 1 (1985), 46–52.

[6] Dawei Chen, Cheng Soon Ong, and Lexing Xie. 2016. Learning Points and Routes to Recommend Trajectories. In *CIKM*. 2227–2232.

[7] Zaiben Chen, Heng Tao Shen, and Xiaofang Zhou. 2011. Discovering popular routes from trajectories. In *ICDE*. 900–911.

[8] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In *EMNLP*. 1724–1734.

[9] Jian Dai, Bin Yang, Chenjuan Guo, and Zhiming Ding. 2015. Personalized route recommendation using big trajectory data. In *ICDE*. 543–554.

[10] Daniel Delling, Peter Sanders, Dominik Schultes, and Dorothea Wagner. 2009. Engineering Route Planning Algorithms. In *Algorithmics of Large and Complex Networks - Design, Analysis, and Simulation*, Vol. 5515. 117–139.

[11] 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.

[12] Tao-Yang Fu and Wang-Chien Lee. 2021. ProgRPGAN: Progressive GAN for Route Planning. In *KDD*. 393–403.

[13] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Deep Sparse Rectifier Neural Networks. In *AISTATS*. 315–323.

[14] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. In *KDD*. 855–864.

[15] Chenjuan Guo, Bin Yang, Jilin Hu, and Christian S. Jensen. 2018. Learning to Route with Sparse Trajectory Sets. In *ICDE*. 1073–1084.

[16] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Comput.* 9, 8 (1997), 1735–1780.

[17] Jayant Jain, Vrittika Bagadia, Sahil Manchanda, and Sayan Ranu. 2021. NeuroMLR: Robust & Reliable Route Recommendation on Road Networks. In *NeurIPS*. 22070–22082.

[18] Manas Joshi, Arshdeep Singh, Sayan Ranu, Amitabha Bagchi, Priyank Karia, and Puneet Kala. 2021. Batching and Matching for Food Delivery in Dynamic Road Networks. In *ICDE*. 2099–2104.

[19] Evangelos Kanoulas, Yang Du, Tian Xia, and Donghui Zhang. 2006. Finding Fastest Paths on A Road Network with Speed Patterns. In *ICDE*. 10.

[20] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *ICLR*.

[21] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.

[22] 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.

[23] Hai Lan, Jiong Xie, Zhifeng Bao, Feifei Li, Wei Tian, Fang Wang, Sheng Wang, and Ailin Zhang. 2022. VRE: A Versatile, Robust, and Economical Trajectory Data System. *PVLDB* 15, 12 (2022), 3398–3410.

[24] Xiucheng Li, Gao Cong, and Yun Cheng. 2020. Spatial Transition Learning on Road Networks with Deep Probabilistic Models. In *ICDE*. 349–360.

[25] 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.

[26] 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.

[27] Zihan Luo, Lei Li, Mengxuan Zhang, Wen Hua, Yehong Xu, and Xiaofang Zhou. 2022. Diversified Top-k Route Planning in Road Network. *PVLDB* 15, 11 (2022), 3199–3212.

[28] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk: online learning of social representations. In *KDD*. 701–710.

[29] Michal Piorkowski, Natasa Sarafijanovic-Djukic, and Matthias Grossglauser. 2009. Dataset of mobility traces of taxi cabs in San Francisco. Retrieved from https://crawdad.org/epfl/mobility/20090224.

[30] Daniele Quercia, Rossano Schifanella, and Luca Maria Aiello. 2014. The shortest path to happiness: recommending beautiful, quiet, and happy routes in the city. In *HT*. 116–125.

[31] Zeyuan Shang, Guoliang Li, and Zhifeng Bao. 2018. DITA: Distributed In-Memory Trajectory Analytics. In *SIGMOD*. 725–740.

[32] T Tieleman and G Hinton. 2012. Divide the gradient by a running average of its recent magnitude. *COURSERA Neural Netw. Mach. Learn.* 4, 2 (2012), 26–31.

[33] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *ICLR*.

[34] 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 *KDD*. 539–547.

[35] Zheng Wang, Cheng Long, Gao Cong, and Yiding Liu. 2020. Efficient and Effective Similar Subtrajectory Search with Deep Reinforcement Learning. *PVLDB* 13, 11 (2020), 2312–2325.

[36] Ling-Yin Wei, Yu Zheng, and Wen-Chih Peng. 2012. Constructing popular routes from uncertain trajectories. In *KDD*. 195–203.

[37] Hao Wu, Ziyang Chen, Weiwei Sun, Baihua Zheng, and Wei Wang. 2017. Modeling Trajectories with Recurrent Neural Networks. In *IJCAI*. 3083–3090.

[38] 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 *KDD*. 1915–1924.

[39] Andy Yuan Xue, Rui Zhang, Yu Zheng, Xing Xie, Jin Huang, and Zhenghua Xu. 2013. Destination prediction by sub-trajectory synthesis and privacy protection against such prediction. In *ICDE*. 254–265.

[40] Bin Yang, Chenjuan Guo, and Christian S. Jensen. 2013. Travel Cost Inference from Sparse, Spatio-Temporally Correlated Time Series Using Markov Models. *PVLDB* 6, 9 (2013), 769–780.

[41] Can Yang and Gyözö Gidófalvi. 2018. Fast map matching, an algorithm integrating hidden Markov model with precomputation. *Int. J. Geogr. Inf. Sci.* 32, 3 (2018), 547–570.

[42] Haitao Yuan, Guoliang Li, Zhifeng Bao, and Ling Feng. 2020. Effective Travel Time Estimation: When Historical Trajectories over Road Networks Matter. In *SIGMOD*. 2135–2149.

[43] 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 *ACM SIGSPATIAL*. 99–108.

[44] Jing Zhao, Jiajie Xu, Rui Zhou, Pengpeng Zhao, Chengfei Liu, and Feng Zhu. 2018. On Prediction of User Destination by Sub-Trajectory Understanding: A Deep Learning based Approach. In *CIKM*. 1413–1422.