DRNCS: Dual-Level Route Generation Model Based on Node Contraction and Shortcuts

Zhuoran Li[1], Yucen Gao[1], Yu Yin[2], Xinle Li[2], Hui Gao[2], Xiaofeng Gao[1]🖂, and Guihai Chen[1]

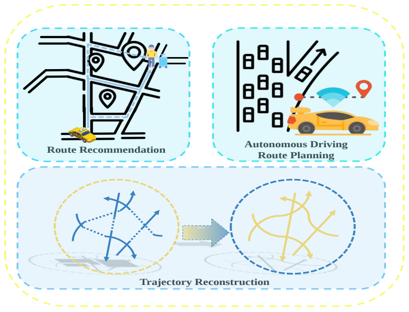
[1] Shanghai Key Laboratory of Scalable Computing and Systems,
School of Computer Science, Shanghai Jiao Tong University, China
{airplane, guo_ke}@sjtu.edu.cn, {gao-xf, gchen}@cs.sjtu.edu.cn
[2] Didi Global Inc., China
{yinyu, lixinle, deangaohui}@didiglobal.com

Abstract. Route prediction is a fundamental task in mapping services, with critical applications in trajectory reconstruction and route recommendation. However, existing algorithms face efficiency challenges, particularly for long routes. The primary challenge is to enhance computational efficiency while maintaining prediction accuracy, especially in large-scale, real-time scenarios. To address this challenge, we propose an accelerated route generation model **DRNCS** based on node contraction and shortcut acceleration, utilizing a dual-level model specifically designed for path prediction. By effectively employing Shortcut-Edge Differential Contraction, we manage to contract nodes in a way that avoids the common problem of out-degree explosion, which is typically encountered in conventional node contraction methods. This approach allows us to transform the original graph into a much sparser representation, preserving the structural integrity while significantly reducing the complexity of the graph. We then compute and store the most likely shortcuts using two methods: merging historical routes and applying a probability-based bidirectional Dijkstra's algorithm on historical paths. Ultimately, the final prediction results are generated by predicting on the sparse graph and invoking the stored shortcut data, which is produced by the model trained on the original graph. Through in-depth experiments on real-world datasets, we establish that our model imparts significant improvement in generation speed while maintaining query accuracy over state-of-the-art approaches and demonstrates advantages in reachability. This provides robust support for the practical application of our algorithm in real-world route generation scenarios.

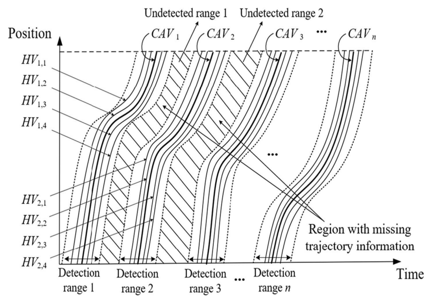
Keywords: Route Recommendation · Path Optimization.

This work was supported by the National Key R&D Program of China [2024YFF0617700], the National Natural Science Foundation of China [U23A20309, 62272302, 62372296], the CCF-DiDi GAIA Collaborative Research Funds for Young Scholars [202404]. Xiaofeng Gao is the corresponding author

1 Introduction



(a) Multiple applications



(b) Trajectory Data Lost

Fig. 1: Application Scenarios of route Recommendation

Route generation technology has significant potential in optimizing urban mobility [5], aiding applications such as helping taxi drivers select faster routes for quicker pickups and improving customer satisfaction [9]. It also enables autonomous vehicles to choose less congested paths, reducing travel time during peak hours and alleviating overall traffic congestion. Additionally, as shown in Figure 1b [14], real-world trajectory data is often incomplete or discontinuous. In such cases, route generation can reconstruct trajectories and forecast potential routes of moving objects, providing valuable insights into future traffic conditions and helping urban planners optimize infrastructure and traffic management [**?**].

In real-world scenarios, as illustrated in Figure 2, the straightforward application of shortest path algorithm to infer the most likely route, intuitively assuming that the shortest path is always the best option, can lead to significant errors, particularly as trajectory length increases and the scale of the road network expands. In this context, considerable scholarly attention has been devoted to the optimization of route generation technologies in recent years. Notably, the NEUROMLR [6] framework has reported marked enhancements in both precision and recall rates. Despite these advancements, prevailing algorithms continue to grapple with substantial challenges pertaining to the time required for route generation, especially in the context of longer path problems, where pronounced search delays are frequently observed. This challenge is particularly evident within the NEUROMLR-D variant, wherein the average prediction time for a single path exceeds three seconds when processing extensive road networks and extended routes. Such performance is insufficient to satisfy the requirements of practical applications.

In light of the current state of research, the following pivotal question emerges: **How can we significantly enhance the efficiency of the search process while preserving the accuracy of route predictions to fulfill practical application requirements?**

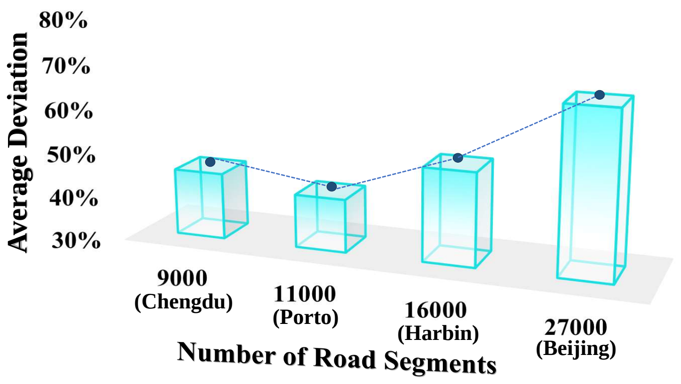


Fig. 2: Loss of Shortest Path

To address this critical issue, we recognize that the extended time required for predicting long paths primarily stems from the massive number of nodes needing processing. Inspired by the CH algorithm [3] that accelerates shortest path computation through node contraction and shortcut addition, we propose **DRNCS** – a **D**ual-Level **R**oute Generation framework based on **N**ode **C**ontraction and **S**hortcuts. Our dual-level acceleration mechanism operates through two complementary tiers: **Architectural-Level Acceleration:** We design a hierarchical learning system where (1) a *contracted-graph model* predicts key nodes on a simplified topology to establish sparse routing skeletons, while (2) an *original-graph model* generates probabilistic shortcuts between these nodes using transition probabilities learned from historical trajectories. **Inference-Level Enhancement:** During path generation, we implement: (1) dynamic shortcut injection from precomputed high-probability connections to bypass redundant node computations, and (2) accuracy preservation through hybrid verification combining model predictions with bidirectional Dijkstra search on transition-probability-weighted graphs. This hierarchical design achieves acceleration through graph contraction while preserving precision via shortcut verification, effectively balancing efficiency and accuracy.

We have carried out the following work in this paper: We propose a node contraction method, Shortcut-Edge Differential Contraction, which uses a top-down edge difference as the criterion to transform the original graph into a sparse representation, while concurrently processing the original trajectory data to construct sparse graph trajectory datasets. Utilizing the original graph, we conduct model training to generate the necessary data for shortcut creation which involves merging historical trajectory information (i.e., selecting a representative historical route from numerous historical trajectories) and calculating the most likely shortcuts using bidirectional Dijkstra's algorithm on the graph weighted by transition probabilities. Based on the sparse graph trajectory datasets mentioned above, we first trained the model on the sparse graph, then made predictions on it, and finally obtained the final prediction result by combining the predicted paths with the shortcut dataset. Finally, we conduct extensive experiments using real-world city dataset to validate our approach. our contributions within this paper are as follows:

– **Maintained Prediction Accuracy:** We employed appropriate embedding techniques and probabilistic prediction models, enabling us to sustain high levels of prediction accuracy and reachability.
– **Accelerated the Prediction Process:** We proposed Shortcut-Edge Differential Contraction to generate a sparse graph, then applied historical path merging and bidirectional Dijkstra's algorithm on graph weighted by transition probabilities based on model training results to obtain shortcut dataset, accelerating subsequent query processes.
– **Refined Dual-Level Model Design:** Our model is trained on both the original and sparse graphs, allowing for more effective acquisition of shortcut and final prediction data.
– **Experimental Validation:** Extensive experiments conducted on real-world city dataset validate the effectiveness of our model and a substantial enhancement in prediction efficiency.

2 Related Work

2.1 Path Prediction Algorithm

Shortest Path-Based Route Prediction The shortest path is the simplest approach for route prediction, widely used in existing studies [10, 16, 11]. [10] proposes the spMM model, leveraging geometric and topological road network features, while [16] incorporates traffic signal delays and left turns into a heuristic cost function. Building on this, [11] constructs a candidate graph favoring straighter paths. However, these methodologies do not adequately account for driver preferences in route selection. In reality, drivers consider various factors beyond the length and straightness of the route. Consequently, the most likely route frequently diverges from the shortest path, as shown in the Figure 2. This suggests that a singular focus on distance is insufficient for accurately predicting the most likely route.

Route Prediction based on RNN To address the challenges associated with route prediction using the shortest path, several existing methods have been proposed that utilize RNNs based on historical trajectories: In the realm of route prediction, several notable models have emerged, each with distinct principles and limitations. **DEEPST** [8] uses variational autoencoders to capture trajectory and traffic relationships but struggles with poor reachability for unseen points. **CSSRNN** [15] leverages RNNs and road network constraints but suffers from long query times, low reachability, and data dependency. **NEUROMLR** [6] combines Lipschitz embeddings with GCNs for accuracy, with two variants: NEUROMLR-D (Dijkstra's Algorithm) and NEUROMLR-G (Greedy Algorithm). NEUROMLR-D enhances shortest-path calculations but has high query latency, while NEUROMLR-G improves speed and reachability but remains inefficient for long routes, requiring further optimization.

Route Prediction based on Transformer While autoregressive models like RNNs have been the cornerstone of route prediction, recent research has increasingly adopted Transformer architectures for their ability to model complex dependencies. [1] employs a Transformer-based framework to capture spatiotemporal correlations in road networks, while [7] introduces BERT-Trip, a BERT-inspired model which is a self-supervised contrastive learning framework for route recommendation that addresses the challenges of learning effective trip representations without labeled data and eliminating the need for manually designed negative samples. Despite their effectiveness, the substantial parameter size of Transformer-based models presents challenges, particularly in inference speed, which constrains their applicability in real-time scenarios.

Table 1: Limitations of Existing Route Prediction Models

Model	**Limitations**
DEEPST	Inability to predict points not in original data; suboptimal reachability.
CSSRNN	Poor reachability; extended query time; reliance on high-quality training data.
NEUROMLR-D	Prolonged query time; less suitable for real-time applications.
NEUROMLR-G	Significant delays for longer routes; need for further optimization.
BERT-Trip	High training costs; Prolonged query time.

In summary, while each method presents unique advantages in predictive capabilities and model design, it is evident that current path prediction methods face significant challenges related to slow query speeds. To mitigate this issue, we investigate various potential approaches aimed at effectively accelerating queries while preserving an acceptable level of accuracy. One particularly promising avenue is inspired by the **Contraction Hierarchies (CH)** algorithm, which enhances the efficiency of shortest path queries through the contraction of nodes and the incorporation of shortcuts.

2.2 Node Contraction Method

Contraction Hierarchies (CH) Contraction Hierarchies (CH) [3] enhance the efficiency of shortest path queries through a preprocessing phase that simplifies the graph structure. This is achieved by contracting less significant nodes while preserving essential connectivity through the introduction of shortcut edges, which maintain the correctness of shortest path distances. Once the preprocessing is complete, shortest path queries are executed using a bidirectional variant of Dijkstra's algorithm [2], which simultaneously explores the search space from both the source and target nodes. This approach effectively reduces the number of processed nodes, thereby significantly improving query performance.

The classical CH algorithm cannot be directly applied to our context of predicting the most likely route. We need to address the following issues:

1. **What criteria should be used for node contraction?** In our application scenario, it is crucial to minimize the increase in out-degree as much as possible, as this will affect the speed at which our model predicts potential nodes. Therefore, We propose Shortcut-Edge Differential Contraction, a top-down approach that uses edge difference as the criterion for node contraction. This approach ensures that we can contract more nodes without causing an explosion in out-degree.
2. **How can we generate shortcuts that are more aligned with the most likely route prediction scenario?** Directly using the shortcuts derived from classical algorithms is not feasible. Consequently, we employ bidirectional Dijkstra's algorithm on a probabilistic graph generated from historical data to create shortcuts that are more representative of the most likely route. This enables acceleration in our predictions.

3 Problem Formulation and Transformation

We begin with the definitions of **Road Network**, **Route** and **Query**. Subsequently, we formally present the research problem and provide its transformation. For the sake of clarity and ease of reference, the notation employed in this paper is presented in Table 2.

Definition 1 (Road Network) A road network is represented as a directed graph $G(V, E)$, in which V and E represent the vertices (crossroads) and edges (road segments) respectively.
Definition 2 (Route) A route $r = [e_i]_{i=1}^{n}$ is a sequence of adjacent road segments, where $e_i \in E$. In practical applications, a route, $R(s, d) = \{v_1, \ldots, v_k\}$ represents a simple path from the source node $s = v_1$ to the destination $d = v_k$ in the road network G, with no cycles. Similarly, it can be expressed as a sequence of edges $R(s, d) = \{e_1, \ldots, e_{k-1}\}$, where $e_i = (v_i, v_{i+1})$.
Definition 3 (Query) A query is represented by a tuple $q = (s, d, t)$, where s and d are the source and destination nodes within the set of nodes V, respectively. The variable t indicates the time when the trajectory takes place.
Problem Statement Given a road network $G(V, E)$ and a historical trajectory dataset $D = (T(m))_{m=1}^{M}$, alongside a query $q = (s, d, t)$, the objective is to infer the most likely route $R^*(s, d)$ based on the traffic patterns contained in D. Formally, the most likely route is defined as:

$$R^*(s, d) = \arg \max_{R(s,d) \in G} \Pr(R(s, d) \mid q) \tag{1}$$

Table 2: Notation and Description Table

Notation	**Description**
$G(V, E)$	A directed graph consisting of vertices V and edges E.
$\|V\|, \|E\|$	Number of nodes and edges.
G_{neg_log}	The graph G weighted by the negative logarithm of probabilities.
R, R^*	True routes and Predicted routes.
$SC^{(1)}, SC^{(2)}$	Shortcuts filtered through historical trajectories and Shortcuts generated based on probabilistic graphs.
$\|R\|, \|R^*\|$	Number of edges in the true route and in the predicted route, respectively.
$\|R^*_{d=d^*}\|$	Count of predicted paths matching the original path's endpoint.
$q = (s, d, t)$	A query is represented by a tuple ,involving start node s, destination node d, and time t.
$Pr(R\|q)$	The probability of selecting route R given a specific query q.
P_S	The set of historical paths within the historical trajectory that share the same start-end pair.
$P(e)$	Probability of selecting edge. e.
UnConf	The unnormalized confidence score for predictions.
$Q((\text{curr}, \text{nbr})\|\text{curr}, d; \Theta)$	Transition probability from the current node to a neighboring node, given the destination and parameterized by Θ.
N2V(node)	The embedding of a node using the node2vec algorithm.
W	Weight matrix for a layer in the neural network.
T_{total}	Total time taken to generate all predicted routes.

Problem Transformation The problem of predicting the most likely route can be framed as a path search problem on the graph. Mathematically, the probability of a route R can be formulated as the product of the probabilities of its individual edges. Formally, we have:

$$\Pr(R|q) = \prod_{i=1}^{|R|} \Pr(R.e_i | R.e_0 \to R.e_{i-1}, s, d, t) \tag{2}$$

where $\Pr(R.e_i|R.e_0 \to R.e_{i-1}, s, d, t)$ denotes the probability that route R traverses edge $R.e_i$ given the path taken thus far and the query parameters $q = (s, d, t)$. Past studies have shown that human mobility patterns conform to the Markovian assumption [13], which states that the future state depends only on the present state and not on the sequence of events that preceded it. Consequently, the above equation reduces to:

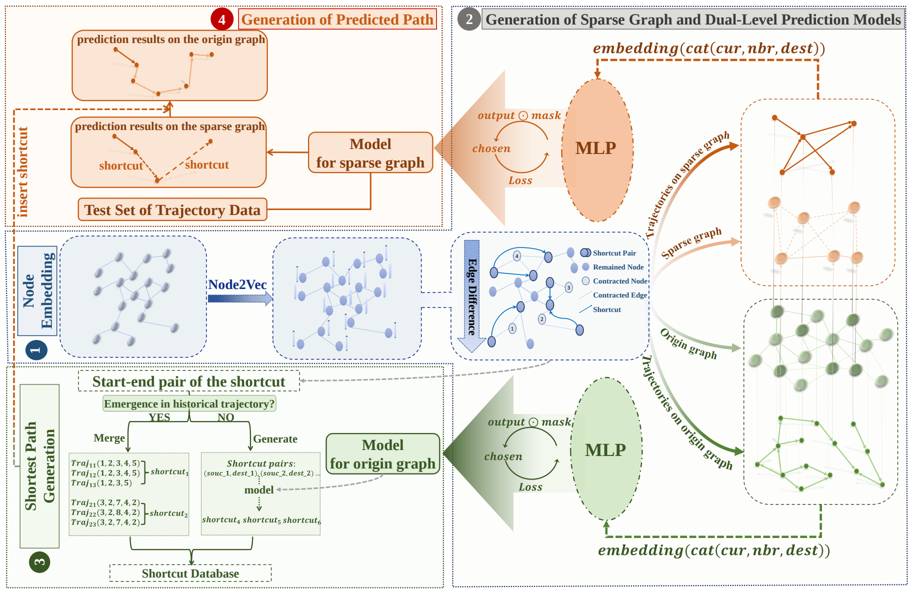

4 Generation of Predicted Path
prediction results on the origin graph
insert shortcut
prediction results on the sparse graph
shortcut
shortcut
Model for sparse graph
Test Set of Trajectory Data
2 Generation of Sparse Graph and Dual-Level Prediction Models
embedding(cat(cur, nbr, dest))
output ⊙ mask
chosen
Loss
MLP
Trajectories on sparse graph
Sparse graph
1 Node Embedding
Node2Vec
Edge Difference
Shortcut Pair
Remained Node
Contracted Node
Contracted Edge
Shortcut
Origin graph
Trajectories on origin graph
3 Shortest Path Generation
Start-end pair of the shortcut
Emergence in historical trajectory?
YES
NO
Merge
Generate
Traj11(1, 2, 3, 4, 5)
Traj12(1, 2, 3, 4, 5)
Traj13(1, 2, 3, 5)
shortcut1
Traj21(3, 2, 7, 4, 2)
Traj22(3, 2, 8, 4, 2)
Traj23(3, 2, 7, 4, 2)
shortcut2
Shortcut pairs: (souc_1, dest_1), (souc_2, dest_2) ...
model
shortcut4 shortcut5 shortcut6
Model for origin graph
output ⊙ mask
chosen
Loss
MLP
embedding(cat(cur, nbr, dest))
Shortcut Database

Fig. 3: Framework of DRNCS: **DRNCS** first generates node embeddings using node2vec, then contracts nodes based on edge difference to create a sparse graph and shortcut pairs. The model is trained separately on both the original and sparse graphs, producing a dual-level model. During the shortcut database generation, two approaches are used: merging historical paths and generating shortcuts based on model results from the original graph. Finally, in the inference phase, predictions are made on the sparse graph, conbining the stored shortcut data to derive the final results.

$$\Pr(R|q) = \prod_{i=1}^{|R|} \Pr(R.e_i|v_i, d, t) \tag{3}$$

With these simplifications, Eq (2) reduces to:

$$R^*(s,d) = \arg\max_{\forall R \in G} \prod_{i=1}^{|R|} \Pr(R.e_i|v_i, d, t) = \arg\min_{\forall R \in G} \sum_{i=1}^{|R|} -\log(\Pr(R.e_i|v_i, d, t)) \tag{4}$$

This transformation indicates that the problem of selecting the most probable route has been converted into a problem of searching for the shortest path on a weighted graph, where the weights correspond to the negative logarithm of the probabilities. We denote this weighted graph as G_{neg_log}.

4 Modeling

Our model incorporates several key components: generating node embeddings, creating sparse graph, training models with original path information on the original graph, and training models using sparsified trajectory data on the sparse graph, generating shortcuts with two distinct approaches, executing the inference phase. The framework of our model is illustrated in Figure 3.

4.1 Node Embedding Generation

The node2vec algorithm [4] effectively captures the structural features of nodes in a graph using a flexible random walk strategy. For our study, which aims to predict the most likely route, we require node embeddings that account for complex factors beyond simple metrics like distance and geographic coordinates. Thus, we selected node2vec as our embedding method.

4.2 Sparse Graph Modeling

To address the query speed issue of route generation, we aim to minimize the increase in the maximum out-degree of nodes during the graph contraction process. So we propose **Shortcut-Edge Differential Contraction**, where we select **edge difference**—defined as the difference between the number of shortcuts added during node contraction and the sum of the node's original in-degree and out-degree—as our criterion for node contraction. Furthermore, we employ a **top-down** approach to control the scale of out-degree increase as much as possible. The node contraction process is presented in Algorithm 1.

Algorithm 1: Shortcut-Edge Differential Contraction

```
   Input: Graph $G = (V, E)$, Contraction ratio $r$
   Output: Shortcut pairs $shortcut\_pair$ , Sparse Gragh $G^*$
1  copy $G$ to $G^*$
2  while (number of contracted nodes $<$ total nodes $\times r$) do
3      Select the node with the minimum edge difference in $G^*$;
4      Contract the selected node;
5      for each predecessor $p$ of the contracted node do
6          for each successor $s$ of the contracted node do
7              if not edge $(p, s)$ exists in $G^*$ then
8                  Add edge $(p, s)$ in $G^*$;
9                  Store $(p, s)$ in $shortcut\_pair$;
10 return $shortcut\_pair, G^*$ ;
```

The algorithm in Algorithm 1 initializes a new graph G^* as a copy of G (line 2) and iteratively contracts nodes until $r \times |V|$ nodes are removed (line 3). In each iteration, the node with the smallest *edge difference* is selected to minimize connectivity disruption (lines 4-6). Once contracted (lines 7-9), shortcut edges are added between its predecessor p and successor s if no direct edge (p, s) exists, storing them in *shortcut_pair*. The algorithm terminates when the contraction ratio is met, returning G^* and the shortcut pairs for model training (line 10).

4.3 Model Training

After completing the node embedding, we train on the original trajectories from the original graph and the processed sparse trajectories from the sparse graph, obtaining the trained models at **two levels**. The trained model from the sparse graph will be used to infer phase, while the trained model from the original graph will be utilized for shortcut generation. The specific details of the model training are as follows: We obtained our embedding results by concatenating the embeddings generated by node2vec, where the 'neighbors' (nbr) include both the true neighboring nodes of each node and the virtual neighboring nodes, represented by -1, which are added through padding to ensure that each node has the same number of neighbors. After obtaining the final embeddings, we pass these embeddings through a multi-layer perceptron (MLP) to convert the vector into a scalar unnormalized confidence value, defined as follows:

$$embedding = cat(N2V(curr), N2V(nbr)N2V(dest)) \tag{5}$$

$$\text{UnConf} = f(nbr, curr, dest) = \text{MLP}(embedding), \quad nbr \in N(curr) \tag{6}$$

The transition probability $Q((\text{curr}, v)|\text{curr}, d; \Theta)$ can be computed as follows:

$$Q((curr, nbr)|curr, dest; \Theta) = \frac{\exp(f(nbr, curr, dest))}{\sum_{nbr' \in N(curr)} \exp(f(nbr', curr, dest))} \tag{7}$$

This equation normalizes the unnormalized confidence by the sum of the exponential values of the confidence for all neighbors $nbr' \in N(curr)$.

Next, we generate a mask based on the actual neighbors of the nodes to filter the unnormalized confidence. We apply the mask to set the confidence values to zero for neighbors represented by -1 (indicating the absence of such neighbors):

$$\text{UnConf}^* = \text{UnConf} \odot \text{mask} \tag{8}$$

We then select the candidate with the highest confidence, simplified as:

$$\text{chosen} = \arg\max_i \left(\text{UnConf}_i^*\right), \quad \text{for } i \in \{1, 2, \ldots, \text{max\_nbrs}\} \tag{9}$$

Finally, the model parameters Θ are optimized through *cross-entropy loss* over trajectories in $\mathbf{D}$, defined as:

$$\text{Loss}(\Theta) = -\frac{1}{|D|} \sum_{R \in D} \sum_{j=1}^{|R|} \log Q(R.e_j | R.nbr_j, R.dest; \Theta) \tag{10}$$

4.4 Shortcut Generation

To generate shortcut data within a short timeframe, we developed our shortcut database based on whether the required start-end shortcut pairs have appeared in historical trajectory data. We employed two approaches for this purpose:

$SC^{(1)}$ **generation** For the start-end pairs that have been observed in the historical data, we utilized a merging approach. Within the historical trajectories, for paths sharing the same start and end points, we computed the precision of each path relative to the others and calculated the average precision as our selection criterion. The path with the highest average precision was selected as the final shortcut data $SC^{(1)}$ for that specific start-end pair.

$$SC^{(1)} = \arg\max_{R_i \in P_S} \frac{1}{N} \sum_{j=1}^{N} \text{Precision}(R_i, R_j), \quad \text{where } R_i, R_j \in P_S \tag{11}$$

$SC^{(2)}$ **generation** For those start-end pairs of shortcuts that do not appear in the historical trajectory, firstly, for a given shortcut's starting point S and endpoint D, we consider all nodes in the graph as potential source nodes, with D designated as the destination. We then utilize the model trained on the original graph to generate the transition probabilities $P(e)$ for each edge e:

$$P(e) = f(nbr, curr, dest), \quad curr \in G \tag{12}$$

where f represents the function derived from our trained model.

Next, we transform these transition probabilities into negative logarithmic weights for incorporation into the graph:

$$W(e) = -\log(P(e)) \tag{13}$$

Following this, we apply a bidirectional Dijkstra's algorithm to identify the most likely route between S and D:

$$SC^{(2)} = \text{BidirectionalDijkstra}(S, D, W) \tag{14}$$

This route $SC^{(2)}$ is then stored in the shortcut dataset. The time complexity of Dijkstra's [12] is $O((|E|+|V|)\log|V|)$, where $|E|$ and $|V|$ represent the number of edges and vertices, respectively.

While this time complexity may be impractical for real-time route recommendation systems, it is exceptionally well-suited for our scenario for the following reasons: First, Dijkstra's algorithm is only used during the preprocessing phase to compute the shortcuts, which is a low-frequency operation. Second, the shortcuts we generate are relatively short compared to the routes that need to be predicted in real-world applications, which reduces the number of nodes and edges that need to be explored during the search process.

4.5 Inference Phase

For a given query $q = (S, D)$, we employ a greedy search algorithm, which relies exclusively on transition probabilities. The prediction of a transition probability $P(e = (curr, nbr)|curr, dest)$ is achieved through a forward pass within the architectural framework of the model. This search is performed on the sparse graph, where we predict the most probable transitions between nodes based on the learned probabilities. To further refine the predictions, we incorporate previously obtained shortcut datasets—those generated in Section 4.4. These shortcuts provide additional context or connections, which are then inserted into the predicted results to enhance the accuracy and robustness of the final prediction.

5 Experiments

5.1 Dataset

The data used in our experiments, presented in Table 3 [1] [2], includes road network and trajectory data from five cities: Chengdu (CD), Porto (PT), Harbin (HRB), Beijing (BJ), and Shenzhen with its surrounding areas (SZ). These datasets, rigorously cleaned and preprocessed, cover diverse road network scales and configurations, closely reflecting real-world urban scenarios. This ensures our model's applicability to large-scale practical settings.

Table 3: Details of datasets

City	**CD**	**PT**	**HRB**	**BJ**	**SZ**
Number of nodes	3,973	5,330	6,598	31,199	79,348
Number of edges	9,255	11,491	16,292	72,156	171,313
Number of trajectories	3,600,503	1,426,312	1,133,548	1,382,948	1,007,209
Average number of edges/trip	22.93	51.07	56.81	36.08	37.02

5.2 Metric

Since the actual length of an edge is not considered as a criterion during the route generation process, and in order to comprehensively test the efficiency and accuracy of our model's predictions, the metrics evaluated in this experiment can be simplified as follows:

$$\text{Precision} = \frac{|R^* \cap R|}{|R^*|}, \text{Recall} = \frac{|R^* \cap R|}{|R|}, \text{Reachability} = \frac{|R^*_{d=d^*}|}{|R^*|}$$

$$\text{Query Time} = \frac{T_{\text{total}}}{|R^*|}, \text{Generated Paths Rate} = \frac{|R^*|}{T_{\text{total}}}$$

[1] https://drive.google.com/file/d/1bICE26ndR2C29jkfG2qQqVkmpirK25Eu/view
[2] https://github.com/lehaifeng/T-GCN/tree/master/data

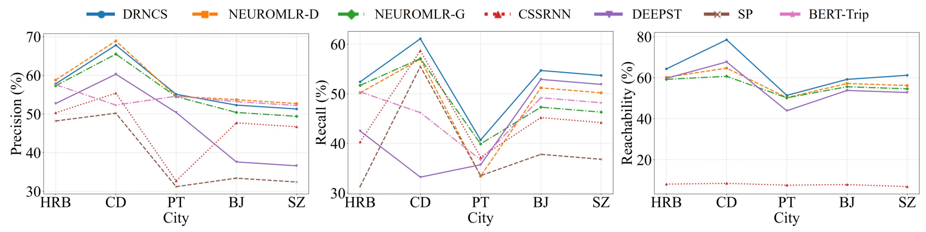


Fig. 4: Model Prediction Accuracy Comparison

5.3 Baseline Methods

We compare the performance of our model with the following baseline models:

- DEEPST (Deep Spatio-Temporal Model) [8]
- CSSRNN (Convolutional Spatiotemporal Sequence Prediction with RNN) [15]
- NEUROMLR-D (NEUROMLR using Dijkstra's Algorithm) [6]
- NEUROMLR-G (NEUROMLR using Greedy Algorithm) [6]
- SP (Shortest Path) [2]
- BERT-Trip [7]

5.4 Implementation Details

The models in this paper are trained with Python 3.6 and PyTorch 2.4.1 on a machine with a 1.70 GHz Intel(R) Core(TM) i5-1240P processor and 16 GB RAM. For the comparative experiments, we use 200 epochs and a batch size of 512. The code is available at https://github.com/iairplane/DRNCS.git.

5.5 Comparison with Baseline Methods

Precision and Recall As shown in Figure 4, the results indicate that, under the same number of training iterations, our model outperforms the NEUROMLR-G, DEEPST, CSSRNN and BERT-Trip methods in terms of accuracy, while maintaining a level of accuracy comparable to NEUROMLR-D. This suggests that our model construction and shortcut generation have achieved desired effect.

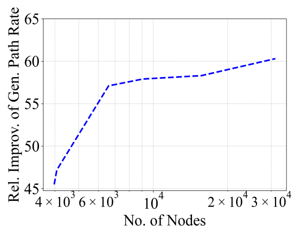
Query Time and Generated Paths Rate Delving deeper into the query time metrics, our model demonstrates exceptional enhancements across different datasets, as shown in Table 4. In the Chengdu dataset, the query speed improved by over 45.5%, by over 40.0% in the Porto dataset, by over 57.1% in Harbin, and by over 47.8% in Shenzhen. Notably, a 57.5% improvement was observed in the Beijing dataset. These results collectively illustrate the substantial optimization of query time realized by our model, aligning perfectly with our initial objective of leveraging shortcuts to expedite the computation of the most likely route.

Table 4: Model Performance Metrics

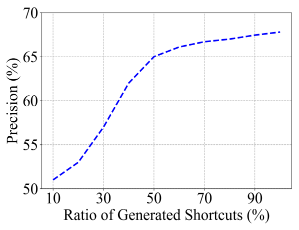
Model	Query Time (ms/trip)					Generated Paths Rate (trips/s)				
	HRB	CD	PT	BJ	SZ	HRB	CD	PT	BJ	SZ
DRNCS	**0.3**	**0.6**	**0.3**	**0.7**	**0.8**	**3356**	**1678**	**3345**	**1432**	**1249**
NEUROMLR-D	243	201	221	314	378	4.11	4.98	4.52	3.18	2.64
NEUROMLR-G	0.7	1.1	0.6	1.1	1.3	1428	902	1667	909	845
CSSRNN	0.9	1.2	1.1	1.4	1.5	1112	773	972	714	678
BERT-Trip	217	236	219	421	489	4.61	4.23	4.56	2.42	2.04

5.6 Analysis of Ablation Study Results

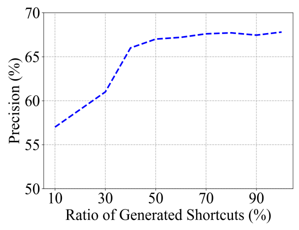
The core component of our model is the shortcut database, primarily consisting of $SC^{(1)}$ and $SC^{(2)}$. Therefore, in the ablation study, we validate the effectiveness of the overall and partial shortcuts in enhancing efficiency while preserving accuracy. Additionally, as selecting appropriate criteria and ratios for node contraction is crucial in model construction, we conduct multiple ablation studies to determine the optimal parameters for our model. The experiment validating the effectiveness of the shortcut in improving query efficiency was conducted on road network datasets of varying scales, while the other experiments were conducted using the Chengdu dataset.



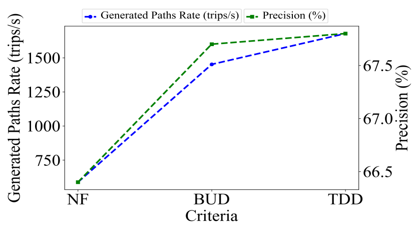
(a) efficiency improvement



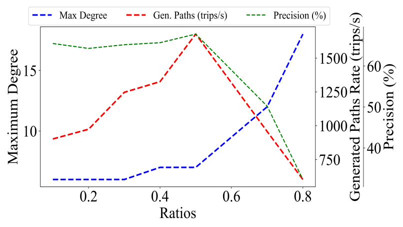
(b) vary shortcut ratios



(c) vary $SC^{(2)}$ ratios



(d) vary node contraction criteria



(e) vary node contraction ratios

Fig. 5: Results of the Ablation Study

Effect of Shortcut As discussed in the previous section, incorporating shortcuts allows us to maintain accuracy while significantly speeding up inference. To further validate our model, we conducted two ablation experiments. First, we tested road networks of varying sizes, comparing the inference speed of DRNCS with and without shortcut insertion, using the Relative Improvement of Generated Paths Rate to show how speed differences increase with network size. The results in Figure 5a confirm the effectiveness of shortcut insertion. Second, we replaced some shortcuts in our database with Dijkstra-generated shortest paths and compared model prediction accuracy at different proportions. The results, presented in Figure 5b, validate the effectiveness of the shortcuts.

Effect of $SC^{(2)}$ Component of Shortcut To further validate the effect of shortcuts generated by bidirectional Dijkstra's algorithm on a graph weighted by transition probabilities, we systematically replaced a subset of these shortcuts with Dijkstra-generated shortest paths at varying proportions, and compared the resulting differences in precision. The results, presented in Figure 5c, demonstrate the effectiveness of the shortcuts in enhancing model performance.

Criteria for Node Contraction We employed three criteria for potential node contraction: node occurrence frequency in historical paths **(NF)**, bottom-up edge difference **(BUD)**, and top-down edge difference **(TDD)**. Each criterion was applied to reduce 30% of the nodes across the entire graph, with the comparison results shown in Figure 5d. The results indicate that selecting the top-down edge difference as the criterion for node contraction effectively maximizes control over the increase in node out-degree, thereby enhancing prediction accuracy and improving search speed. Additionally, the top-down node contraction method demonstrates a notably faster contraction speed.

Ratios for Node Contraction In order to achieve a balance between accuracy and query speed, we tested the precision and query rate of our model under different contraction ratios. The comparison results are presented in Figure 5e. Thus, to simultaneously maintain a high prediction accuracy and a faster query rate, we ultimately selected a contraction ratio of 0.5.

6 Conclusion

Route generation plays a critical role in numerous downstream applications. In practical scenarios, when addressing the problem of recommending the most likely route, it is not only imperative to ensure high prediction accuracy, but also to significantly improve the inference speed. This latter aspect remains a key limitation of existing route generation models. To mitigate this issue, we propose **DRNCS**, an accelerated dual-level route generation model based on node contraction and shortcuts. Specifically, we transformed the original graph into a sparse graph through Shortcut-Edge Differential Contraction. By training the model on both the original trajectories from the original graph and the sparse trajectories from the sparse graph, we successfully generated a dual-level trained model. Furthermore, we employed a shortcut generation approach that consists of two distinct methods: first, merging historical paths, and second, generating shortcuts based on the trained model from the original graph. This comprehensive methodology enabled us to create a more accurate shortcut database in a significantly reduced timeframe. Ultimately, the final prediction results are derived by performing predictions on the sparse graph while invoking the stored shortcut data. Additionally, testing on road network datasets of varying scales demonstrated that our approach achieved over 40% improvement in query speed, while effectively maintaining satisfactory accuracy and reachability. Notably, this optimization in query speed becomes even more pronounced as the scale of the road network increases, thereby highlighting the practical applicability and advantages of our model in real-world scenarios.

References

1. Bhumika, Das, D.: Marrs: A framework for multi-objective risk-aware route recommendation using multitask-transformer. Proceedings of the 16th ACM Conference on Recommender Systems (2022), https://api.semanticscholar.org/CorpusID:252216597
2. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik **1**, 269–271 (1959)
3. Geisberger, R., Sanders, P., Schultes, D., Delling, D.: Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In: McGeoch, C.C. (ed.) Experimental Algorithms. pp. 319–333. Springer Berlin Heidelberg (2008)
4. Grover, A., Leskovec, J.: node2vec: Scalable feature learning for networks. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2016)
5. Hsieh, H.P., Lin, F.: Recommending taxi routes with an advance reservation – a multi-criteria route planner. International Journal of Urban Sciences **26**, 162 – 183 (2021), https://api.semanticscholar.org/CorpusID:233810149
6. Jain, J.L., Bagadia, V., Manchanda, S., Ranu, S.: Neuromlr: Robust & reliable route recommendation on road networks. In: Neural Information Processing Systems (2021)
7. Kuo, A.T., Chen, H., Ku, W.S.: Bert-trip: Effective and scalable trip representation using attentive contrast learning. 2023 IEEE 39th International Conference on Data Engineering (ICDE) pp. 612–623 (2023), https://api.semanticscholar.org/CorpusID:260171278
8. Li, X., Cong, G., Cheng, Y.: Spatial transition learning on road networks with deep probabilistic models. IEEE 36th International Conference on Data Engineering (ICDE) pp. 349–360 (2020)
9. Li, X., Cong, G., Sun, A., Cheng, Y.: Learning travel time distributions with deep generative model. The World Wide Web Conference (2019)
10. Rahmani, M., Koutsopoulos, H.N.: Path inference of low-frequency gps probes for urban networks. International IEEE Conference on Intelligent Transportation Systems pp. 1698–1701 (2012)
11. Rahmani, M., Koutsopoulos, H.N.: Path inference from sparse floating car data for urban networks. Transportation Research Part C-emerging Technologies **30**, 41–54 (2013)

12. Tarjan, R.E.: Data structures and network algorithms. In: CBMS-NSF Regional Conference Series in Applied Mathematics (1983)
13. Wang, J., Wu, N., Zhao, W.X., Peng, F., Lin, X.: Empowering a* search algorithms with neural networks for personalized route recommendation. Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (2019)
14. Wang, Y., Wei, L., Chen, P.: Trajectory reconstruction for freeway traffic mixed with human-driven vehicles and connected and automated vehicles. Transportation Research Part C: Emerging Technologies **111**, 135–155 (2020)
15. qing Wu, H., Chen, Z., Sun, W., Zheng, B., Wang, W.: Modeling trajectories with recurrent neural networks. In: International Joint Conference on Artificial Intelligence (2017)
16. Zheng, Y., Quddus, M.A.: Weight-based shortest-path aided map-matching algorithm for low-frequency positioning data (2011)