Path Representation Learning in Road Networks

Yang, Sean Bin

DOI (link to publication from Publisher):
10.54337/aau510734142

Publication date:
2022

Document Version
Publisher's PDF, also known as Version of record

PATH REPRESENTATION LEARNING IN ROAD NETWORKS

BY
SEAN BIN YANG

DISSERTATION SUBMITTED 2022

AALBORG UNIVERSITY
DENMARK

Path Representation Learning in Road Networks

Ph.D. Dissertation

Sean Bin Yang

Dissertation submitted October 21, 2022

Dissertation submitted: October 21, 2022

PhD supervisor: Prof. Bin Yang
Aalborg University

PhD Co-supervisor: Assoc. Prof. Jilin Hu
Aalborg University

PhD committee: Associate Professor Alvaro Torralba (chairman)
Aalborg University, Denmark

Professor Gao Cong
Nanyang Technological University, Singapore

Professor Baihua Zheng
Singapore Management University, Singapore

PhD Series: Technical Faculty of IT and Design, Aalborg University

Department: Department of Computer Science

ISSN (online): 2446-1628
ISBN (online): 978-87-7573-796-3

Published by:
Aalborg University Press
Kroghstræde 3
DK – 9220 Aalborg Ø
Phone: +45 99407140
aauf@forlag.aau.dk
forlag.aau.dk

Printed in Denmark by Stibo Complete, 2022

Abstract

With the continued digitization of transportation systems, there is an increasing demand for path-based smart city applications which require appropriate path representations. To support path-based applications, different techniques have been proposed to learn path representations. However, existing studies suffer from the following limitations: (1) Supervised methods learn a task-specific path representation and require a large amount of labeled training data, which works well on the labeled task, but generalizes poorly on other tasks; (2) Although graph representation learning based methods learn a task-unspecific path representation, they cannot capture sequential dependencies and fail to introduce the temporal information into the learned path representations; (3) Existing works mainly focus on accuracy improvement, ignoring the mode scalability and size, which plays a critical role in resource-constrained environments.

In this thesis, we investigate the task-unspecific path representation learning approaches that are able to generalize well for different downstream tasks. More specifically, we focus on the following specific works. (1) Context-aware path ranking in road networks. (2) Unsupervised path representation learning. (3) Weakly-supervised contrastive curriculum path representation learning. (3) Lightweight and scalable path representation learning.

First, we study the path ranking framework *PathRank*. This framework learns task-specific path representations, which are used to rank candidate paths. In particular, we propose a training data enrichment strategy to enhance the learning process. Subsequently, we propose an end-to-end context-aware multi-task framework to enable the *PathRank*. We conduct extensive experiments on one real-world dataset to verify the effectiveness of the *PathRank*.

Second, we study the unsupervised path representation learning framework Path InfoMax (PIM) by maximizing the mutual information. PIM takes as input a path and output task-unspecific path representations. In particular, we first propose a curriculum negative sampling strategy to enhance the *PIM* training. Then, we propose a path-path discriminator and path-node discriminator to jointly learn task-unspecific path representations by capturing the global and local information of the path. Finally, we conduct extensive

experiments on two downstream tasks under two real-world datasets. The results illustrate our *PIM* is more effective than other baseline methods. In addition, the pre-trained *PIM* can enhance supervised learning methods.

Third, we study a weakly-supervised contrastive curriculum temporal path representation learning framework by leveraging the information from weak labels and considering both spatial and temporal correlations. This framework takes as input a temporal path (a path with a departure time) and returns task-unspecific temporal path representations. We first introduce the weak label to capture the temporal variation of traffic. Then, we study the weakly-supervised contrastive learning method to enable the temporal path encoder training. Subsequently, we combine weakly-supervised contrastive learning with a two-stage curriculum learning strategy to improve the performance of weakly-supervised contrastive learning. Finally, we conduct extensive experiments on three downstream tasks under three real-world datasets. The results show the effectiveness of our proposals.

Finally, we study the lightweight and scalable path representation framework *LightPath*. This framework aims at learning task-unspecific path representations by reducing resource consumption and enabling model scalability with respect to path length. In particular, we first propose a sparse auto-encoder that guarantees *LightPath* with good scalability of path length. Then, we propose cross-network and cross-view relational reasoning to train sparse path encoders jointly. Subsequently, we propose global-local knowledge distillation to reduce the model size and improve the performance of the learned path representations. Finally, extensive experiments are conducted, and the results demonstrate the efficiency and scalability of the *LightPath*.

Resumé

I takt med digitaliseringen af transportsystemer stiger efterspørgslen efter sti-baserede smart-city applikationer, hvilket skaber et behov for hensigtsmæssige repræsentationer af stier i vejnetværk. Forskellige teknikker er de senere år blevet udviklet til at lære sti-repræsentationer med henblik på at understøtte sti-baserede applikationer. Men eksisterende teknikker har flere begrænsninger: (1) Superviserede teknikker er i stand til at lære opgavespecifikke sti-repræsentationer, men de kræver store mængder af træningsdata; og mens de er velfungerende til de valgte opgaver, så generaliserer de dårligt til andre opgaver. (2) Selvom grafbaserede teknikker er i stand til at lære opgave-uspecifikke repræsentationer, så kan de hverken modellere sekventielle afhængigheder eller knytte tidsinformation til lærte sti-repræsentationer. (3) Eksisterende teknikker fokuserer desuden på forbedring af nøjagtigheden på bekostning af skalerbarhed og modelstørrelse, som er vigtige aspekter i situationer med begrænsede ressourcer.

Afhandlingen præsenterer løsninger til læring af opgave-uspecifikke sti-repræsentationer, der understøtter tiltænkte applikationer. Mere specifikt omhandler afhandlingen følgende: (1) Kontekstafhængig rangering af stier. (2) Ikke-superviseret læring af stier. (3) Svagt superviseret såkaldt contrastive curriculum læring af stier. (4) Resursebesparende og skalerbar læring af stier.

Først præsenter afhandlingen PathRank, der muliggør rangering af stier ved først at lære opgavespecifikke sti-repræsentationer, som derefter kan anvendes til at rangere stier. Indledningsvist præsenteres teknikker til berigelse af træningsdata med henblik på forbedret læring. Herefter præsenteres et komplet kontekstafhængigt rammeværk, der realiserer PathRank. Endelig præsentes omfattende eksperimentelle studier af PathRank baseret på et virkeligt datasæt.

For det andet præsenterer afhandlingen PathInfoMax (PIM), der muliggør ikke-overvåget læring af opgaveuspecifikke sti-repræsentationer baseret på maksimering af gensidig information. Afhandlingen præsenterer en såkaldt curriculum negative sampling-teknik for at forbedre PIM's læring. Derefter præsenter den sti-sti og en sti-knude diskriminatorer med henblik på fælles læring af opgaveuspecifikke sti-repræsentationer baseret på globale og lokale

informationer om stier. Endelig præsenteres eksperimentelle studier af to opgaver baseret på to virkelige datasæt. Studierne viser, at PIM er bedre end andre løsninger og at PIM med prætræning kan forbedre svagt superviserede læringsmetoder.

For det tredje præsenterer afhandlingen et rammeværk, der tager stier med afgangstider som input og lærer opgaveuspecifikke repræsentationer af disse ved hjælp af svagt overvåget kontrastiv curriculum læring, som inddrager bl.a. spatiale og tidsmæssige korrelationer. Afhandlingen præsenterer eksperimentelle studier af tre opgaver baseret på tre virkelige datasæt. Studierne viser, at rammeværket fungerer hensigtsmæssig.

For det fjerde præsenterer afhandlingen et rammeværk, LightPath, der har til formål at muliggøre resursebesparende og skalerbar læring af opgaveuspecifikke stier. LightPath omfatter en encoder, der sikrer god skalerbarhed i forhold til stilængden. Desuden anvendes såkaldt relational læring samt teknikker til modelreduktion med henblik på at gøre LightPath resursebesparende. Afhandlingen præsenterer eksperimentelle studier af resurseforbrug og effektivitet, som viser at LightPath opnår sine designmål.

Acknowledgements

First, I would like to express my sincere gratitude to my supervisor Prof. Bin Yang, for giving me an opportunity to be one of his Ph.D. students. I would like to thank him for the continuous support of my Ph.D. study and research and for his patience, motivation, enthusiasm, and immense knowledge. I am very grateful for his guidance, which helped me in all the time of research and writing this thesis. His constructive suggestions on academic research made me feel more and more confident about my research. I could not have imagined a better supervisor for my Ph.D. study.

Second, I would like to thank Associate Prof. Jilin Hu, who helped me greatly during my Ph.D. study. Without his professional and construction suggestions, I cannot imagine that I could have these achievements. We have scheduled meetings every week regarding my research work, and I am often inspired by our discussions. It is my honor to work with him. I learned a lot from him regarding study and research. In addition, I would like to thank Prof. Christian S. Jensen and Associate Prof. Chenjuan Guo for their insightful comments when helping me revise my papers and for teaching me how to write a good research paper.

Third, I would like to thank Associate Prof. Jian Tang, who hosted me as a visiting student at Mila-Quebec AI Institute while studying abroad. I appreciate his support and constructive suggestions for my research work.

Fourth, I would like to thank all my colleagues at Daisy, who provided a warm environment when I studied here. I also want to thank Independent Research Fund Denmark, the VILLUM FONDEN, and the Innovation Fund Denmark center, DIREC, for funding my position and providing an opportunity for a six-month extension.

Finally, I would like to thank my family and friends, especially my parents and wife, for their encouragement and support. They are always there to help and support me whenever I need help. In addition, I also want to thank my new baby, who is the best gift to me during my Ph.D. study. Thanks to everyone for not making me feel alone.

Sean Bin Yang
Aalborg University, October 21, 2022

Contents

Thesis Details

Thesis Title:	Path Representation Learning in Road Networks
PhD Student:	Sean Bin Yang Aalborg University
PhD Supervisor:	Prof. Bin Yang Aalborg University
PhD Co-supervisor:	Assoc. Prof. Jilin Hu Aalborg University

The main body of the thesis consists of the following papers.

(A) **Sean Bin Yang**, Bin Yang "*Learning to Rank Paths in Spatial Networks*," in 2020 IEEE 36th International Conference on Data Engineering (ICDE). IEEE, pp. 2006–2009, 2020.

(B) **Sean Bin Yang**, Chenjuan Guo, Bin Yang "*Context-Aware Path Ranking in Road Networks*," IEEE Transactions on Knowledge and Data Engineering (TKDE), vol. 34, no. 7, pp. 3153-3168, 2022.

(C) **Sean Bin Yang**, Chenjuan Guo, Jilin Hu, Jian Tang, Bin Yang "*Unsupervised Path Representation Learning with Curriculum Negative Sampling*," in International Joint Conferences on Artificial Intelligence(IJCAI), pp. 3286-3292, 2021.

(D) **Sean Bin Yang**, Chenjuan Guo, Jilin Hu, Bin Yang, Jian Tang, Christian S. Jensen "*Weakly-supervised Temporal Path Representation Learning with Contrastive Curriculum Learning*," in 2022 IEEE 38th International Conference on Data Engineering (ICDE). IEEE, pp. 2873–2885, 2022.

(E) **Sean Bin Yang**, Jilin Hu, Chenjuan Guo, Bin Yang, Christian S. Jensen "*LightPath: Lightweight and Scalable Path Representation Learning*," in submission, 2022.

This thesis has been submitted for assessment in partial fulfillment of the Ph.D. degree. The thesis is based on the submitted or published scientific papers listed above. Parts of the content of the papers in the main body

of the thesis are used directly or indirectly in the extended summary part of the thesis. As part of the assessment, co-author statements have been made available to the assessment committee and are also available at the Faculty. The permission for using the published and accepted articles in the thesis have been obtained from the corresponding publishers with the condition that they are cited and copyrights are placed prominently in the references. In reference to IEEE copyrighted material, which is used with permission in this thesis, the IEEE does not endorse any of Aalborg University's products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to `http://www.ieee.org/publications_standards/publications/rights/rights_link.html` to learn how to obtain a License from RightsLink.

Part I

Thesis Summary

Chapter 1

Introduction

1 Background and Motivation

With the acceleration of smart city developments, especially for the continued digitalization of transportation systems, a large number of trajectory data has been generated, promoting an increasing range of path-based intelligent transportation applications, e.g., travel cost estimation [22, 25, 30, 59], routing [36, 39, 40, 74], path recommendation [14, 15, 24], and traffic analysis [22, 25, 30, 59]. Path representation learning (PRL) is the basis to support these applications under different traffic scenarios, where PRL aims to learn the representation vector for each path. Figure 1.1 shows an example, we aim at learning a function $f(\cdot) : \mathbb{R}^{N \times D} \rightarrow \mathbb{R}^d$, where N is the number of edges in a path, D is the dimension of edge representation vector. This function embeds each path in a road network into a d dimensional representation vector (i.e., Z_{P_1} and Z_{P_2}).

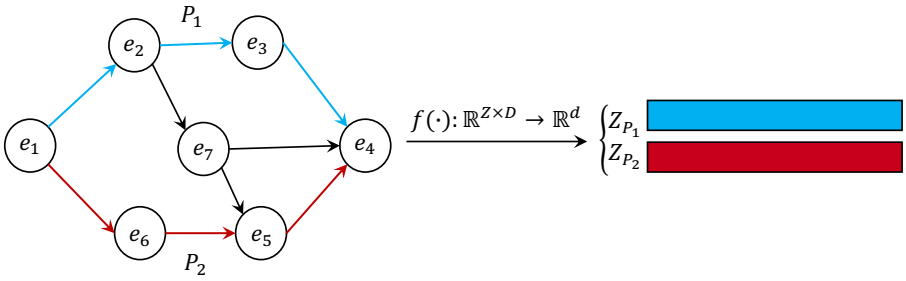


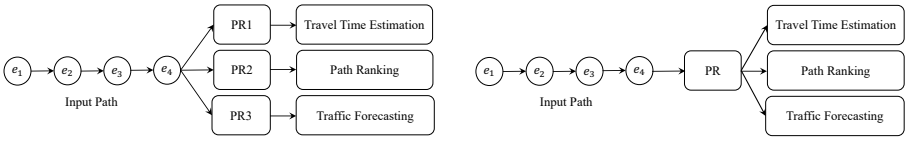
Figure 1.1: The road network consists of multiple paths, for example $P_1 = \langle e_1, e_2, e_3, e_4 \rangle$ and $P_2 = \langle e_1, e_6, e_5, e_4 \rangle$.

1.1 Task-Specific PRL

Supervised Learning

Some existing works explore the path representation learning from the task-specific labels perspective through a supervised manner [10, 17, 33, 54, 70] and return the task-specific path representations (PRs). It is worth noting that, task-specific PRs are path representation learned from task labeled data. However, (1) Task-specific PRs work well on the task labeled data but generalize poorly on another tasks (e.g., PR1 works well on travel time estimation task but returns poorly results on path ranking and traffic forecasting, respectively. The same is true for PR2 and PR3. Refer to Figure 1.2a.). (2) Learning task-specific PRs require a larger amount of labeled data, where high-quality data annotations are very expensive and time-consuming, and there are many unlabeled data available. These limitations call for a task-unspecific PRL.

Example 1.1 (Task-specific vs. Task-unspecific PRL)
Figure 1.2 compares the task-specific PRL with task-unspecific PRL. Given an input path $p = \langle e_1, e_2, e_3, e_4 \rangle$ and three downstream tasks, e.g., travel time estimation, path ranking, and traffic forecasting. Task-specific PRL aims to learn three independent PRs (i.e., PR1, PR2, and PR3) for three different tasks with availability of a larger amount of labeled data, which is shown in Figure 1.2a. In contrast, as shown in Figure 1.2b, task-unspecific PR is to learn generic PR that can work well for different tasks. To this end, we aim to learn a task-unspecific path representation PR for a given path p by using larger amount of unlabeled data, which can be applied in different downstream tasks, e.g., travel time estimation, path ranking, and traffic forecasting, respectively.



(a) Task-specific PRL **(b)** Task-unspecific PRL

Figure 1.2: Task-specific vs. Task-unspecific.

1.2 Task-unspecific PRL

Unsupervised Learning

Some other existing works investigate path representation learning from an unsupervised learning perspective based on graph representation learning [12, 46, 47], where we first learn a task-unspecific graph/node representation vector, and then we achieve the task-unspecific path representation by aggregating node representation vector in a path. In particular, the goal of unsupervised graph representation learning is to learn task-unspecific graph/node representations by directly using raw unlabeled data, where these representations can work well for different tasks, e.g., link prediction and node classification. However, although we can achieve the PR through aggregation based on learned node representations, it ignores the correlations between each node in a path. This limitation calls for a novel algorithm that is designed for learning a task-unspecific path representation in an unsupervised manner.

Example 1.2 (Graph Representation Learning based PRL)
Figure 1.3 illustrates an example of graph representation learning based PRL. Given an input road network graph, graph representation learning methods aim to learn a mapping function $g(\cdot)$ that takes as input a road network graph and returns node representation. In practice, this mapping function $g(\cdot)$ can be *Node2vec*$(\cdot)$ [13], *deepwalk*$(\cdot)$ [42], etc.. Then, we use $g(\cdot)$ to achieve each node representation vector in a road network graph. As shown in Figure 1.3, the colored □ denotes the node representations. Finally, we achieve the path representations (i.e., Z_{P_1}, and Z_{P_2}) by aggregating each node representation vectors in a path.

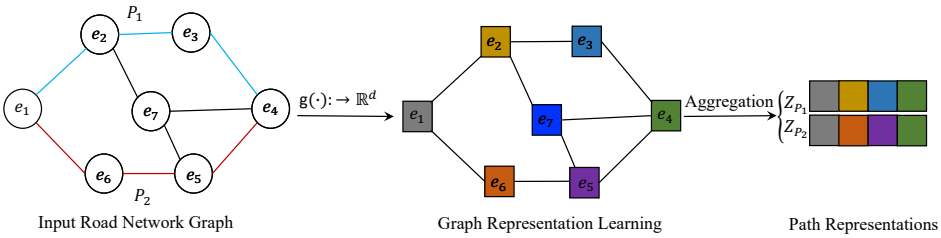


Figure 1.3: Graph Representation Learning based PRL.

Weakly Supervised Learning

To the best of our knowledge, transportation systems is dynamic system that varies with time, i.e., the temporal aspect plays an essential role in

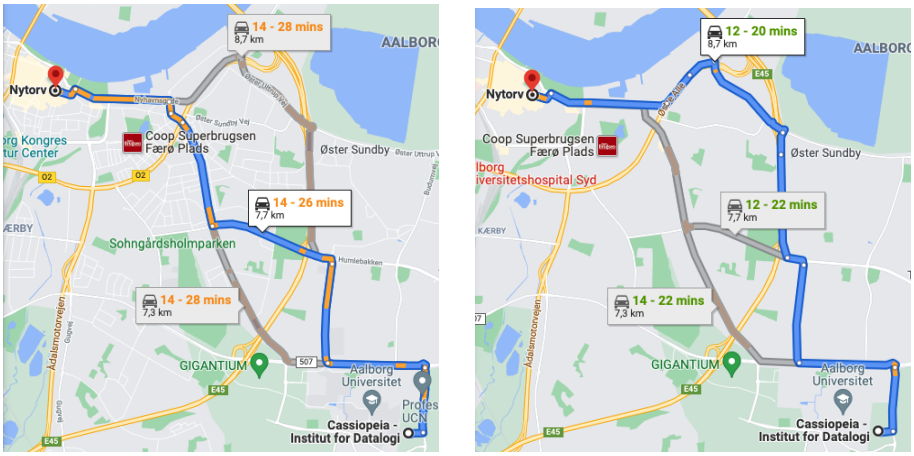
transportation system applications. A native way to learn task-unspecific temporal path representations (TPRs) is first to learn task-unspecific PRs through unsupervised learning and then concatenate the temporal embedding with learned PRs. However, this operation is not effective as it cannot fully explore the temporal information. It is non-trivial to learn temporal path representations (TPRs) through unsupervised learning. To ensure the task-unspecific TPRs that are able to conduct on different downstream applications, we aim to learn TPRs based on weakly-supervised contrastive learning by introducing weak labels, where weak labels are very cheap and easy to achieve. In particular, these weak labels are relevant to different tasks, e.g., peak vs. off-peak periods.

Example 1.3 (Travel Time Estimation and Weak Label)
Consider the travel-time estimation example from Google Maps in Figure 1.4. Travel from *"Cassiopeia"* to *"Nytorv"* takes longer at 8:00 a.m. than at 10:00 a.m., due to the traffic congestion during morning peak hours. Further, it can be seen that the path recommendation rankings are also different. It recommends avoiding the highway at 8:00 a.m. due to the heavy congestion there, while recommends the highway again at 10:00 a.m., when the traffic is clear [63]. Thus, the temporal aspect plays an essential role in smart-city path-based applications. Learning task-unspecific path representation without considering temporal information led to poor performance. In particular, we introduce the weak label to learn TPRs. Take Figure 1.4a as example, we define temporal path as (P_i, dt_i), where P_i denotes the path and dt_i denotes the departure time. In this case, the representation of $(P_i, 8:00 a.m.)$ and $(P_i, 10:00 a.m.)$ should be different since they depart from the peak and off-peak hour, respectively.

Lightweight and Scalability PRL

Since path is represented as a sequence of edges, path representation learning often employs models that are good at capturing sequential relationships, such as Transformer [49]. However, a Transformer-based method [3] employs a self-attention mechanism, where one edge attends to all other edges in a path in each attention, resulting in quadratic complexity, $\mathcal{O}\left(N^2\right)$ of path length N, where the path length is the number of edges in a path. As a result, models exhibits poor scalability with respect to path length.

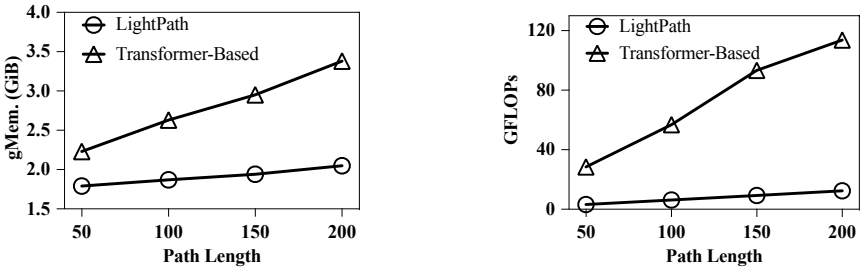
Example 1.4 (Scalability w.r.t. Path Length)
Figure 1.5 gives an example of the scalability w.r.t. path length N, covering both memory consumption and computational cost, in terms of GPU



(a) 8:00 a.m. **(b)** 10:00 a.m.

Figure 1.4: Travel Time Estimation. Travel times of paths from *Cassiopeia* to *Nytorv* at (a) 8:00 a.m. and (b) 10:00 a.m [63]. © 2022 IEEE

memory (gMem.) and Giga floating point operations per second (GFLOPs), respectively. We observe when the path length N increases from 50 to 200, the Transformer-based method performs poorly.



(a) GPU Memory **(b)** GFLOPs

Figure 1.5: Scalability w.r.t. Path Length [65].

In addition, existing path representation learning works mainly focus on performance improvement by introducing more depth networks. Moreover, models with large amounts of parameters also suffer from high storage and computational costs. Such costs are unattractive, particularly in resource-limited environments. These limitations require a lightweight and scalability framework to learn task-unspecific path representations.

Example 1.5 (Model Size with different Encoder Layers)
Table 1.1 shows the numbers of parameters of Transformer-based path encoders when varying the number of layers among 12, 24, 48, and 96 while fixing the number of heads at 8 per layer and the feature dimension of the encoder at 512. We can observe that the model parameters grow dramatically when the number of encoder layers increases, preventing the models from being deployed in resource-constrained environments.

Table 1.1: Model Parameter Size with Varying Encoder Layers [65]

Encoder Layers L	12	24	48	96
Parameters (Millions)	29.85	55.07	105.51	206.40

In a brief, efficient and accurate path representations enable drivers better schedule their routes, thus improving traffic conditions and reducing carbon emissions to meet the requirements of carbon neutrality [16]. Therefore, path representation learning plays an essential role in Intelligent Transportation Systems (ITS) as it enables efficient traffic system planning and management. As mentioned above, there are several challenges to solving the problems mentioned above. First, *how to learn context-aware task-specific model for path ranking?* Second, *how to mine the unlabelled data to learn task-unspecific path representations through unsupervised learning?* Third, *how to enhance the task-unspecific path representations learning by leveraging the information from the weak labels?* Fourth, *how to design a lightweight and scalability framework to learn task-unspecific path representations?* However, there is still no systematic and comprehensive study on path representation learning to solve these challenges, which is shown in Figure 1.6. To bridge the gap, we propose and study the following problems.

1. **Context aware path ranking in road networks.** Following the existing supervised learning methods, we study the context-aware path ranking framework, which learns a task-specific path representation. In particular, we first study the powerful training data enhancement methods that generate compact and diversified training sample sets based on historical trajectories, enhancing efficient and powerful feature learning by providing more effective training data. Then, we present the multi-task learning based *PathRank* strategy to study the task-specific path representation by considering different relevant contexts information [64, 66].

2. **Unsupervised path representation learning with curriculum negative sampling.** We study a new unsupervised path representation learning

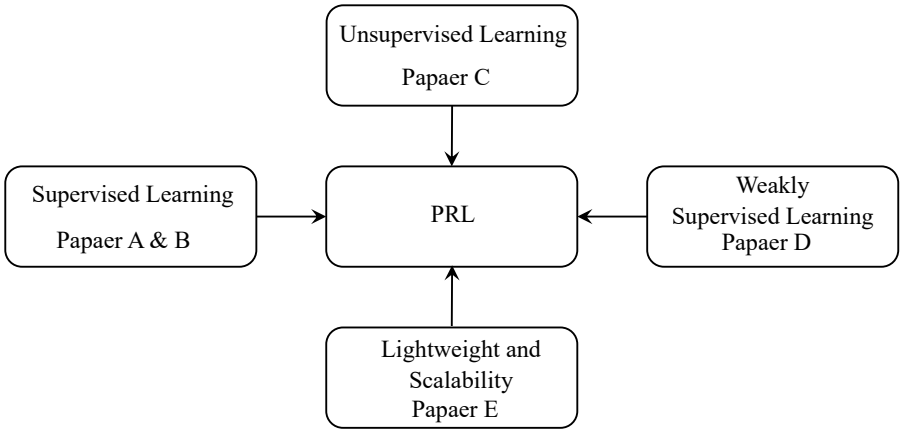


Figure 1.6: Path Representation Learning.

framework Path InfoMax(PIM), which returns task-unspecific path representations based on mutual information maximization and a curriculum negative sampling strategy [62]. To enhance the *PIM* training, we first construct negative paths for each input path based on the fundamental principles (i.e., generating paths from easy to hard order) of curriculum learning, where we first generate the most accessible negative paths that are entirely different from the input path (i.e., no overlap exists between input and these negative paths). Then, we gradually increase the generation difficulty that makes the input path with overlap with negative paths. Subsequently, we further investigate the path-path discriminator and path-node discriminator to train the path encoder by mutual information maximization, which finally returns the task-unspecific path representations [62].

3. **Weakly-supervised temporal path representation learning with contrastive curriculum learning.** We propose a novel path representation learning methods based on weakly supervised contrastive curriculum learning (WSCCL) [63]. It returns a task-unspecific temporal path representation by simultaneously taking into account spatial and temporal correlations. In particular, to enable temporal path representation learning, we first introduce a weak label to capture the temporal variation of the traffic dynamics, which is cheap and relevant to different downstream applications [22, 25, 39, 59, 63]. Then, we study the weakly-supervised contrastive learning method to train a temporal path encoder which returns a task-unspecific temporal path representation [63]. To further enrich the weakly-supervised contrastive learning, we next combine curriculum learning strategies with weakly-supervised learning, leading

to the improvement of the generalization capacity and convergence rate for TPR learning [63].

4. **LightPath: lightweight and scalable path representation.** We further propose and study a lightweight and scalable path representation learning, termed *LightPath*, which aims to reduce resource consumption and achieve scalability without affecting accuracy, thus enabling broader applicability [65]. We first propose a sparse path auto-encoder that ensures the *LightPath* with good scalability in terms of path length [65] Next, we propose a relational reasoning framework to enable faster training of sparse path encoder [65] We further design a global-local knowledge distillation to reduce the size and improve the performance of sparse path encoder [65].

2 Thesis Structure

The thesis utilizes machine learning methods to learn task-unspecific path representations, the goal being of providing a foundation for systematic and comprehensive path representation study that is able to contribute to enabling different smart-city transportation system applications, e.g., travel time estimation, eco-routing, path recommendation, and traffic analysis. As already mentioned before, we aim at addressing three challenges in this thesis and the overall outline is given in Figure 1.7. First, we study the context-aware path ranking framework based on training data enrichment strategy and multi-task learning by following existing supervised methods. However, this model returns task-specific path representations. Then, to learn task-specific path representations, we propose a novel unsupervised path representation framework Path InfoMax *PIM* by mutual information maximization and curriculum negative sampling, which provide a theoretic foundation for task-unspecific PRL and thus solve the first challenge. Subsequently, considering the dynamic properties of real traffic conditions, i.e., temporal variation, we further investigate a task-unspecific temporal path representation learning by leveraging the information of weak label (e.g., departure time.) and introducing a weakly-supervised contrastive learning principle. To improve the temporal path representation learning, we also study the curriculum learning-based weakly-supervised contrastive learning methods, resulting in the improvement of the generalization capacity and convergence rate for TPR learning. This method solves the second challenge. Finally, we explore a lightweight and scalable path representation learning framework, aiming at reducing resource consumption and achieving scalability without affecting accuracy. We propose a sparse path auto-encoder to guarantee the framework with good scalability in terms of path length. Next, we further propose a

relational reasoning framework to train sparse path auto-encoder faster and more robustly. Finally, we study the global-local knowledge distillation to reduce mode size and improve the performance of the sparse path encoder. This framework solves the last challenge.

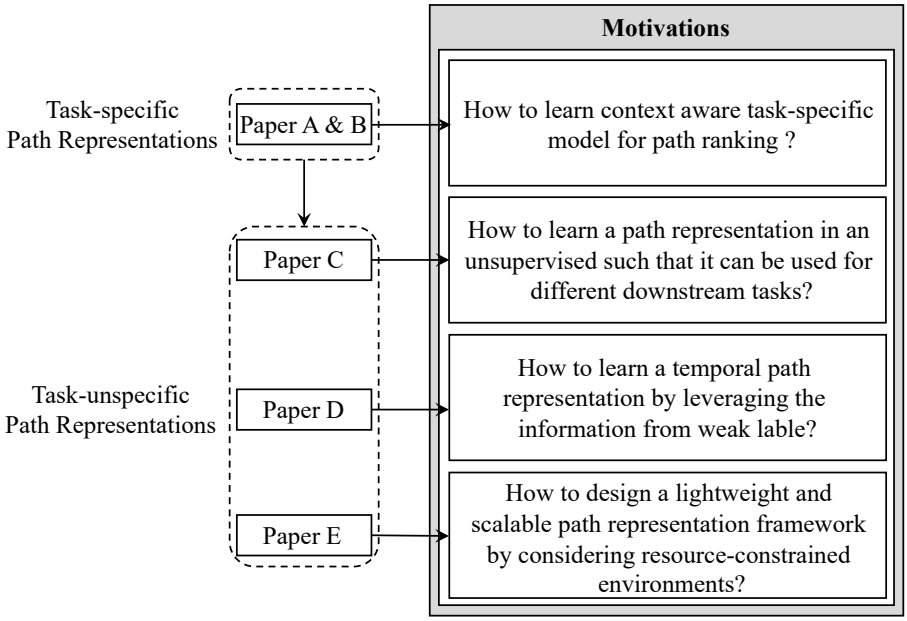


Figure 1.7: Thesis Structure. Paper A and Paper B study the context-aware path ranking by training data enrichment and multi-task learning, which returns a task-specific path representation. To learn task-unspecific PRs, Paper C proposes a unsupervised path representation learning methods. Further, to learn powerful temporal path representations, Paper D study the weakly-supervised contrastive learning for TPRs. Finally, we investigate the lightweight and scalability path representation learning by considering resource-constrained environment. The recommended order to go through the thesis is Paper A&B then Paper C then Paper D then Paper E.

Chapter 2

Context-Aware Path Ranking in Road Networks

This chapter provides a brief conclusion of Paper A [66] and Paper B [64]. Paper A is an original *PathRank* framework, which is a data-driven end-to-end framework and was published as a poster paper in 2020 [66]. In contrast, Paper B extends the original idea by proposing a novel training data enrichment strategy and using multi-task learning to further improve the ranking performance, where termed as a basic framework. Based on this, we further propose advanced framework that is able to incorporate contexts, e.g., departure time and driver information. Then we conduct some extra experiments to verify the effectiveness of the proposed architectures, which include the effects of different training data enriching strategies, effects of different node embedding learning strategies, effects of multi-task learning, effects of contexts, comparisons with baseline ranking strategies, comparisons with driver-specific ranking strategies, effects on training data sizes, and online efficiency. These changes improve the performance of Paper A. In addition, we also reorganized the Paper structure based on these new changes. Thus, Paper A is a special case of Paper B. This chapter reuses content from paper A and paper B when that is considered the most effective.

1 Problem Motivation and Statement

Vehicular transportation plays an essential role in our daily lives. Recently, with the acceleration of digitization of transportation systems, there are many vehicle trajectory data generated, which offer a solid data assurance to improve the performance of various transportation services, especially for vehicle routing. Generally, we use an algorithm like Dijkstra to search for a single

optimal path with the least travel cost, e.g., the fastest and shortest routes, when given a source and destination. However, there is one routing service study [6] pointed out that local drivers generally select the paths that are not the fastest or shortest, indicating traditional routing algorithms do not work well in some real-world routing cases. To this end, some works, e.g., top-k shortest path routing [68] and skyline routing [60], are investigated to return multiple paths that drivers can select.

Under this scenario, candidate path ranking becomes critical to guarantee high-quality routing. Nonetheless, existing studies often rank these paths based on simple heuristics, w.r.t. their travel times. In addition, when drivers choose their paths, they may not always select the fastest paths. Moreover, existing studies offer all drivers with the same ranking, which assume all drivers with the same driver preference.

To address the aforementioned challenges, in Paper A [66], we study the data-driven framework *PathRank*, which returns ranked candidate paths by considering local driver used paths based on their historical trajectories. In contrast, in Paper B [64], we propose a data-driven, context-aware ranking framework *PathRank* to rank paths in road networks [64]. In particular, *PathRank* formulates candidate path ranking as a regression problem, which predicts the ranking score for each candidate path with respect to the local driver's trajectories. In addition, we can accommodate different contextual information into the *PathRank*, which enables the flexible framework. More specifically, we first propose a training data-enriching strategy that returns a compact and diversified training path set, which includes different travel costs that drivers may consider [64]. Then, we propose an end-to-end multi-task framework to effectively train the *PathRank*, which simultaneously takes road network topology and spatial properties into account. Next, we introduce the context representation into the *PathRank* to enable context-aware ranking.

The contributions of Paper B are summarised as following [64]:

- First, we propose a method to generate a compact and diversified set of training paths that enables effective and efficient learning [64].
- Second, we propose a multitask learning framework to enable spatial network embedding that captures not only topological information but also spatial properties [64].
- Third, we integrate contextual information embedding into the framework to enable context-aware ranking [64].
- Fourth, we conduct extensive experiments using a large real-world trajectory set to offer insight into the design properties of the proposed framework and to demonstrate that the framework is effective [64].

2 Preliminaries

We first give the preliminaries and problem definition. The following definitions in this section are reproduced from [64, 66].

2.1 Basic Concepts

A *road network* is represented as a weighted, directed graph $G = (\mathbb{V}, \mathbb{E}, D, T, F)$, where $\mathbb{V}$ denotes vertex set and $\mathbb{E}$ denotes edge set. In particular, edge set $\mathbb{E} \subset \mathbb{V} \times \mathbb{V}$ denotes road segments. D, T, and F are the edge distance, travel time, and fuel consumption, respectively [64].

A *path* $P = (v_1, v_2, v_3, \ldots, v_X)$ consists of a sequence of X vertices where $X > 1$ and there should be an edge between two neighboring vertices [64].

A *trajectory* $\boldsymbol{T} = (p_1, p_2, p_3, \ldots, p_Y)$ consists of a sequence of GPS records and each record $p_i = (location, time)$ denotes the vehicle location at a specific timestamp [64].

Path Similarities: Many similarity functions [9, 14, 37, 61] can be employed to compute the similarity between different paths. Here, we conduct the weighted JaccardSimilarity [14, 61](see Equation 2.1) to calculate the ranking score for training paths.

$$sim\,(P_1, P_2) = \frac{\sum_{e \in P_1 \cap P_2} G.D(e)}{\sum_{e \in P_1 \cup P_2} G.D(e)} \tag{2.1}$$

Ranking scores: Given a trajectory path P and another path P' that are under the same source $P.s$ and destination $P.d$. We treat the similarity between the trajectory path P and another path P' as the ranking score for P', where we take the trajectory path P as ground truth path. Thus, the higher the ranking score is, the higher ranking P' achieves [64].

2.2 Problem Definition

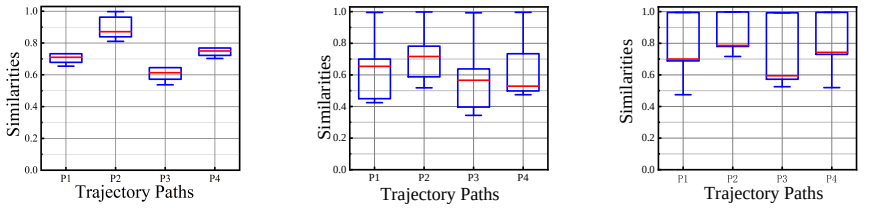
Given a set of N candidate paths $\mathbb{P}$ that connect the same source and destination and optional contexts such as a departure time and a driver identifier, we aim at (1) estimating a ranking score $sim(P, P'_i)$ for each candidate path $P'_i \in \mathbb{P}$; and (2) providing a ranked list of the candidate paths $\langle P'_1, P'_2, \ldots, P'_N \rangle$, such that $sim(P, P'_i) \geq sim(P, P'_j)$ when $1 \leq i < j \leq N$ [64].

3 Training Data Enrichment

Training data enrichment strategy aims to construct a set of paths $\mathcal{PS}$, which consist of the other paths that the driver takes into account during the routine

schedule. This $\mathcal{PS}$ is called as *competitive path set*.

Generally, the competitive path set is generated by including all paths from source s to destination d [64]. However, this is impractical in the form of real-world applications since the $\mathcal{PS}$ may contain a larger number of paths in terms of city-level road networks [64]. To this end, we aim to build a competitive path set module to generate an identity competitive path set that contains a small number of paths, e.g., less than 20 paths [64]. In particular, we should select these paths carefully instead of randomly choosing them.



(a) Top-9 Shortest Paths **(b)** Diversified Top-9 Shortest Paths **(c)** Multi-cost Diversified Paths

Figure 2.1: Similarity Spreads of Different Strategies [64].© 2022 IEEE TKDE

The easiest way to construct competitive path set PS is to use the traditional top-k shortest path algorithm, e.g., Yen's algorithm [68], to generate top-k shortest paths from s to d to build competitive path set $\mathcal{PS}$. However, the shortcoming is that the generated paths are very similar. For example, we randomly select four different sources and destinations based on the trajectory paths [64]. Then, we conduct Yen's algorithm to generate top-9 shortest paths [64]. Next, the similarity between the competitive path with trajectory paths is computed. Figure 2.1a illustrates the box plots for the trajectory path similarity. It can be seen that the similarity spread in a small range for four sources and destinations, which will degrade the training performance since the limited estimation range of the training instances.

To this end, we propose the other strategy based on the diversified top-k shortest paths [37], which generates top-k shortest paths set DkPS that the paths are dissimilar with each other in terms of threshold δ. Figure 2.1b shows the box plots of the similarity for the same four sources and destinations by using the diversified top-9 shortest paths when threshold $\delta = 0.8$ [64]. The similarities spread larger ranges than Figure 2.1a [64].

In addition, existing works on personalized routing [6, 14, 61] indicates that drivers may take different travel costs, e.g., distance, travel time, and fuel consumption, into account when scheduling their routine. This encourages us to consider different travel costs when constructing the competitive path set. To address this, we study a simple but effective framework, where we run diversified top-k shortest paths x times where each time we consider a specific travel cost [64]. Then, we construct our competitive path set $\mathcal{PS}$

based on these diversified paths generated based on multiple travel costs. For example, we consider three travel costs (i.e., distances, travel times, and fuel consumption) to generate diversified top-3 shortest, fastest, and most fuel-efficient paths, which makes our $\mathcal{PS}$ also contain 9 paths [64]. Figure 2.1c illustrates the similarities of four same sources and destinations under a multi-costs diversified paths scenario. We observe that the similarities spread larger ranges, and the ranges are closer to 1 [64].

4 Ranking Framework

We study the end-to-end, data-driven framework to predict the similarity score for paths [64]. We first study the basic framework, including a spatial network embedding and a recurrent neural network [64]. Next, we study the advanced framework by introducing contextual embedding, e.g., departure time and driver identifiers [64].

4.1 Basic Framework

Considering that *PathRank* takes as input a path, i.e., competitive path P'_i, and corresponding label ranking score, i.e., similarity, sim_i [64]. To address the ranking score prediction issues, we first convert each vertex in the input path to a feature vector [64]. Then, we use a recurrent neural network to embed a sequence of feature vectors to achieve the path representations. Next, these path representations are used to estimate the ranking score sim_i with respect to the ground truth ranking score, which enables our basic framework, as shown in Figure 2.2.

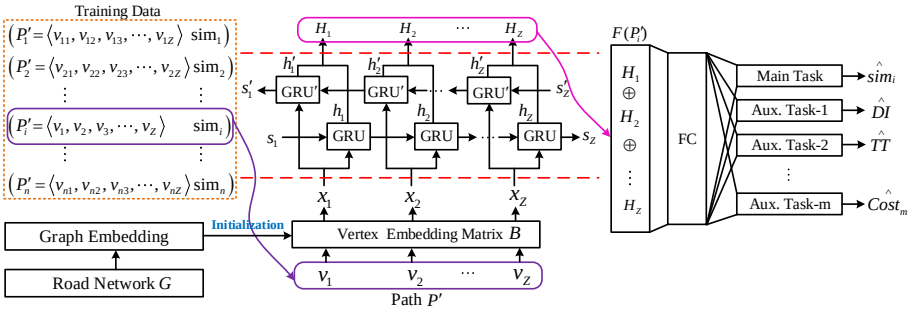


Figure 2.2: Basic Framework of *PathRank* [64].© 2022 IEEE TKDE

Vertex Embedding

Given a competitive path $P'_i = \langle v_1, v_2, \ldots, v_Z \rangle$, we employ the vertex embedding to convert a node's one-hot vector q_i into a new feature vector $x_i = Bq_i \in \mathbb{R}^M$ by defining an embedding matrix $B \in \mathbb{R}^{M \times N}$ [64]. Thus, we achieve a sequence of feature $\langle x_1, x_2, \ldots, x_Z \rangle$. Moreover, to capture the topology of the road network graph, we further introduce the graph embedding into the framework. We first achieved the node embedding through existing graph embedding approaches, e.g., DeepWalk or node2vec, representing each node in a graph into a low-dimensional feature vector. Then, we initialize the embedding matrix B in the embedding module in *PathRank* based on this node embedding achieved from graph embedding approaches [64]. This enables *PathRank* to update the embedding matrix B during the training resulting in the learned model being able to capture the graph topology and estimate the ranking score simultaneously [64]. To further enhance the embedding matrix B learning, we propose a multi-task learning framework, where we set the similarity estimation as the main task and reconstruct the multiple travel costs of the competitive path as the auxiliary tasks, which ensures the learned embedding matrix B also considers spatial properties of the road network graph [64]. In contrast, Paper A considers the ranking score estimation task.

RNN

After representing a path P'_i into a sequence of feature vectors $\langle x_1, x_2, \ldots, x_Z \rangle$, we apply a bidirectional gated recurrent neural network (BD-GRU) [4] to embed the sequence of feature vectors into a path representation by considering the sequence dependencies in both the direction and the opposite direction of the travel flow on path P'_i [64].

In particular, we first embed the path travel flow from left to right, where a GRU unit embed sequential correlations by maintaining a hidden state $h_j \in \mathbb{R}^Q$ at position j and $h_j = GRU(x_j, h_{j-1})$, where x_j represents the feature vector of an input at position j and h_{j-1} indicates the hidden state at position $j-1$ [64]. The specific definition of GRU is given as follows:

$$\mathbf{r}_j = \sigma\left(\mathbf{W}_r x_j + \mathbf{U}_r \mathbf{h}_{j-1}\right) \tag{2.2}$$

$$\mathbf{z}_j = \sigma\left(\mathbf{W}_z x_j + \mathbf{U}_z \mathbf{h}_{j-1}\right) \tag{2.3}$$

$$\tilde{\mathbf{h}}_j = \phi\left(\mathbf{W}_h x_j + \mathbf{U}_h\left(\mathbf{r}_j \odot \mathbf{h}_{j-1}\right)\right) \tag{2.4}$$

$$\mathbf{h}_j = \mathbf{z}_j \odot \mathbf{h}_j + \left(1 - \mathbf{z}_j\right) \odot \tilde{\mathbf{h}}_j \tag{2.5}$$

where σ is the logistic function, $\odot$ denotes the Hadamard product, and ϕ is the hyperbolic tangent function. x_j and $\mathbf{h}_j$ are the feature vector and hidden state at position j, respectively. $\mathbf{W}_r$, $\mathbf{W}_z$, $\mathbf{W}_h$, $\mathbf{U}_r$, $\mathbf{U}_z$ and $\mathbf{U}_h$ are parameters to be learned [64].

Then, we further embed the path travel flow from right to left, where we employ another GRU to achieve hidden state $\mathbf{h}'_j = GRU'(x_j, \mathbf{h}'_{j+1})$ [64]. Here, the GRU takes as input a feature vector at position j and the hidden state at the position [64]. To this end, we represent the hidden state H_j at position j as $H_j = \mathbf{h}_j \oplus \mathbf{h}'_j$, where $\oplus$ is the concatenation operation [64]. We achieve the path representations by stacking all hidden state output by BD-GRU, i.e., $F(P'_i) = \langle H_1 \oplus H_2 \oplus \ldots \oplus H_Z \rangle$.

Fully Connected Layer

We define weight vector $W_{FC} \in \mathbb{R}^{|F(P'_i)| \times X}$ to transfer each competitive path P'_i's to a vector of X values with respect to the estimated similarity score, travel time ,distance and fuel consumption [64].

Loss Function

To enable our *PathRank* training, we formulate the objective function for the multi-task learning in Equation 2.6.

$$\begin{aligned}\mathcal{L}(\mathbf{W}) = &\frac{1}{|n|}[(1-\alpha) \cdot \sum_{i=1}^{n} \left(\hat{sim}_i - sim_i\right)^2 + \\ &\alpha \cdot \sum_{i=1}^{n} \sum_{k=1}^{m} \left(\hat{y}_i^{(k)} - y_i^{(k)}\right)^2] + \lambda \|\mathbf{W}\|_2^2\end{aligned} \tag{2.6}$$

The first term of the loss function measures the discrepancy between the estimated similarity $\hat{sim}_i$ and the ground truth similarity sim_i [64]. The second term of the loss function represents auxiliary tasks that consider the discrepancies between the actual spatial properties vs. the estimated spatial properties [64].

4.2 Advanced Framework

Ranking candidate paths should also consider context-dependent. For example, drivers may choose different paths as the traffic condition varies with the time. To capture such information, we further propose an advanced framework by leveraging the contextual information of departure time t and a driver ID k [64]. Figure 2.3 illustrate the advanced framework.

To enable the *PathRank* model to capture the context information, we first need to embed each context into a meaningful feature space. As for departure time, we first construct the temporal graph, which is shown in Figure 2.4, where we split a day into five intervals–a morning peak interval, an afternoon peak interval, and three off-peak intervals [64]. If two nodes have a similar traffic situation, then we connect these two nodes. Next, we conduct graph

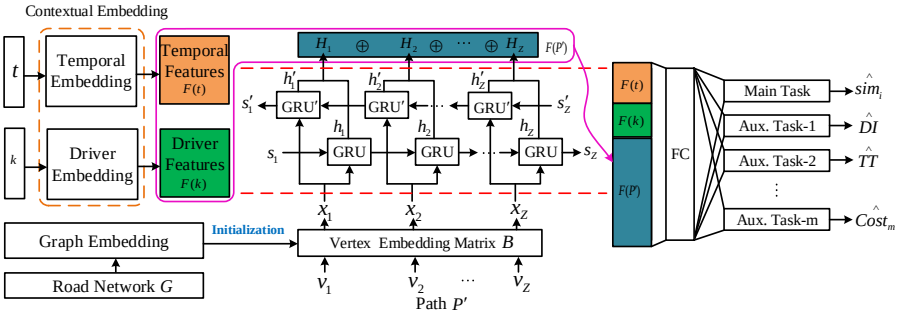


Figure 2.3: Advanced *PathRank* Overview [64].© 2022 IEEE TKDE

embedding to achieve the node feature vectors for a temporal graph. Given a departure time t, we first match the node $node(t)$ into the specific time interval and then achieve embedding vector for this node $F(t) = GraphEmbed(node(t))$ [64].

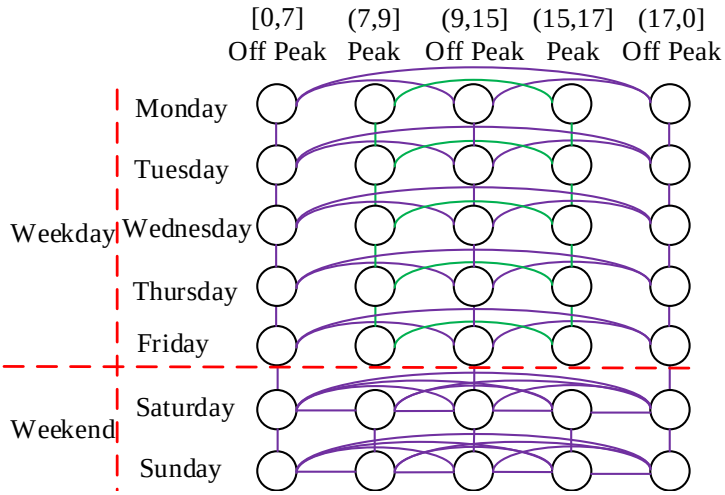


Figure 2.4: Temporal Graph [64].© 2022 IEEE TKDE

In addition, we employ one-hot embedding to convert driver ID k into the corresponding feature vector $F(k)$ [64]. In particular, we can achieve the final feature vector for the competitive path P'_i is $F(t) \oplus F(k) \oplus F(P'_i)$ of the driver k at departure time t. Then, we follow the basic framework and estimate the ranking score and spatial properties using the same loss function defined in Equation 2.6 [64].

5 Experiments

5.1 Experiments Setup

Road Network and Trajectory

We use the Danish dataset, where we achieve the corresponding road network from OpenStreetMap. This road network contains 667,950 vertices and 818,020 edges [64]. We utilize the trajectory data recorded on this road network, and we partition the GPS records into 22,612 trajectories, which represent the different trips. Then, we use the map matching method [38] to obtain the corresponding trajectory path.

Travel Costs and Ground Truth Data

Here we consider three different travel costs, i.e., travel distance (DI), travel time (TT), and fuel consumption (FC) [64].

We use 70% trajectory set for training, 10% for validation, and 20% for testing. More specifically, we first achieve the corresponding source s, destination d, and the trajectory path P_T for each trajectory path T [64]. Then, we construct seven different competitive path set based on the source-destination pairs (s, d) [64].

1. Top-k shortest paths (*TkDI*) [64];
2. Top-k fastest paths (*TkTT*) [64];
3. Top-k most fuel efficient paths (*TkFC*) [64];
4. Diversified top-k shortest paths (*D-TkDI*) [64];
5. Diversified top-k fastest paths (*D-TkTT*) [64];
6. Diversified top-k most fuel efficient paths (*D-TkFC*) [64];
7. Diversified, multi-cost top-k paths (*D-TkM*) [64].

Then, the weighted Jaccard similarity $sim(P, P_T)$ is used to compute the ground truth ranking score for each trajectory path P.

PathRank Frameworks

We consider 5 different *PathRank* variation in our implementation: (1) *PR-B*: we first consider the implementation in Paper A. We random initialized the vertex embedding matrix B and set the $\alpha = 0$ in Equation 2.6, which means we just consider the ranking score estimation task. (2) *PR-A1*: we use the

node matrix from graph embedding methods to initialize the embedding matrix B, and we do not update it during the training procedure. In addition, here, we also consider the main task, i.e., $\alpha = 0$. (3) *PR-A2*: This strategy is almost the same as the *PR-A1*, but we update the embedding matrix during the training. (4) *PR-A2-Mx*: The difference between *PR-A2* and this strategy is that we use multi-task learning to capture spatial properties. In addition, we utilize *PR-A2-Mx* to represent a *PathRank* model, where x means how many auxiliary tasks are considered in an objective function. (5) *PRC*: we introduce the contextual information into *PR-A2-Mx*.

Evaluation metrics

To evaluate the performance of the *PathRank* framework, we consider two different types of metrics. We use Mean Absolute Error (MAE) and Mean Absolute Relative Error (MARE) to measure the ranking score estimation accuracy with respect to the ground truth ranking scores [64]. We further apply the Kendall rank correlation coefficient (τ) and Spearman's rank correlation efficient (ρ) to measure the ranking performance [64].

Baselines

We consider six regression models as the baselines to verify the effectiveness of the *PathRank* [64].

1. Linear Regression (LR) [45];
2. Lasso Regression [52];
3. Support Vector Regression (SVR) [2];
4. Decision Tree Regression (DT) [27];
5. Decision Tree Regression with Adaboost(DTA) [31];
6. Long Short-Term Memory (LSTM) [21], we use the LSTM to replace the GRU.

5.2 Experiments Results

We first evaluate the effect of the training data generation strategy on *PR-A2*, where we update the embedding matrix B during the training procedure. Table 2.1 illustrates the results. We observe that diversified, multi-cost top-k paths obtain the best results compared to when using top-k paths or diversified top-k paths. In addition, we also found that the larger embedding feature size results in better results.

Table 2.1: Training Data Generation Strategies, *PR-A2* [64].© 2022 IEEE TKDE

Strategies	M	MAE	MARE	τ	ρ
TkDI	64	0.1163	0.1868	0.6835	0.7256
	128	0.1130	0.1814	0.7082	0.7481
TkTT	64	0.1218	0.1956	0.6858	0.7282
	128	0.1161	0.1864	0.7026	0.7446
TkFC	64	0.1216	0.1952	0.6911	0.7321
	128	0.1082	0.1737	0.7070	0.7477
D-TkDI	64	0.0940	0.1509	0.7144	0.7532
	128	0.0855	0.1373	0.7339	0.7731
D-TkTT	64	0.1010	0.1622	0.7283	0.7693
	128	0.0997	0.1600	0.7169	0.7596
D-TkFC	64	0.0938	0.1506	0.7318	0.7743
	128	0.0809	0.1299	0.7386	0.7811
D-TkM	64	0.0966	0.1551	0.7393	0.7771
	128	**0.0725**	**0.1164**	**0.7528**	**0.7905**

Next, we study the performance of different vertex embedding. For graph embedding, we employ node2vec to embed nodes in a graph by considering unweighted and weighted graphs. Table 2.2 shows the results. It can be seen that *PR-B* achieves the worst estimation accuracy since it randomly initializes the embedding matrix B. We further observe that *PR-A1* is better than *PR-B*, which means that the ranking performance can be improved by considering the road network topology. We also find that with or without considering edge weight does not improve the estimation and ranking performance. Moreover, *PR-A2* achieves the best results, indicating updating the embedding matrix B is able to improve the performance.

Table 2.2: Effects of Vertex Embedding Strategies [64].© 2022 IEEE TKDE

	Embedding	MAE	MARE	τ	ρ
PR-B	—	0.1159	0.1816	0.7233	0.7611
PR-A1	unweighted	0.0878	0.1410	0.7453	0.7852
	weighted	0.0792	0.1271	0.7478	0.7876
PR-A2	unweighted	0.0734	0.1178	**0.7640**	**0.8012**
	weighted	**0.0725**	**0.1164**	0.7528	0.7905

We also study the effects of multi-task learning. In Table 2.3, *PR-A2-M1* represents we just consider only one auxiliary task, i.e., travel distances. Under this scenario, our *PathRank* estimates both the ranking score and travel distance of the competitive paths. When $\alpha = 0$, we ignore the auxiliary task, which

changes the *PR-A2-Mx* into *PR-A2*. When $\alpha > 0$, we consider the auxiliary task, which improves the ranking performance. In particular, *PR-A2-M3* gives the best results when $\alpha = 0.6$ with respect to both τ and ρ, which indicates the estimated ranking score is the most consistent with the ground truth ranking. In addition, auxiliary task, i.e., distance, travel time, and fuel consumption, is able to improve the ranking performance.

Table 2.3: Effects of α, *PR-A2-Mx* [64].© 2022 IEEE TKDE

	α	**MAE**	**MARE**	τ	ρ
PR-A2	0	0.0725	0.1164	0.7528	0.7905
PR-A2-M1	0.2	0.0756	0.1214	0.7713	0.8057
	0.4	0.0704	0.1129	0.7765	0.8110
	0.6	0.0693	0.1113	0.7783	0.8141
	0.8	0.0680	**0.1029**	0.7712	0.8057
PR-A2-M2	0.2	**0.0653**	0.1048	0.7727	0.8089
	0.4	0.0701	0.1125	0.7869	0.8235
	0.6	0.0777	0.1247	0.7752	0.8100
	0.8	0.0807	0.1296	0.7616	0.7973
PR-A2-M3	0.2	0.0724	0.1162	0.7732	0.8092
	0.4	0.0740	0.1188	0.7711	0.8090
	0.6	0.0662	0.1063	**0.7923**	**0.8261**
	0.8	0.0695	0.1116	0.7842	0.8177

We further compare the *PathRank* with other regression baselines. Here we conduct the compassion based on two different categories of features: (1) Basic feature (BF), where we use the travel distance, travel time, and fuel consumption to construct a feature vector for each path. (2) Advanced features (AF), where we use the node embedding matrix from node2vec to construct features for each path. Table 2.4 gives the comparison results. *PRC* achieves the best results.

Table 2.4: Comparison with Regression Baselines [64].© 2022 IEEE TKDE

	Method	MAE	MARE	τ	ρ
BF	**LR**	0.2640	0.4012	0.6879	0.7150
	Lasso	0.2876	0.4371	0.6245	0.6678
	SVR	0.2390	0.3632	0.6543	0.6683
	DT	0.2516	0.3824	0.6530	0.6777
	DTA	0.2686	0.4082	0.6784	0.7135
AF	**LR**	0.3430	0.5213	0.0864	0.0854
	Lasso	0.2955	0.4484	0.6260	0.6686
	SVR	0.3369	0.5120	0.0857	0.0846
	DT	0.4141	0.6284	0.0450	0.0693
	DTA	0.4301	0.6527	0.0812	0.0395
Deep Learning	**LSTM**	0.2682	0.4076	0.4569	0.4619
	PRC	**0.0611**	**0.0929**	**0.8178**	**0.8454**

Chapter 3

Unsupervised Path Representation Learning with Curriculum Negative Sampling

This chapter offers a brief summary of Paper C [62] and does not offer additional contributions. This chapter reuses content from the paper C when that is considered most effective.

1 Problem Motivation and Statement

Path Representation (PR) plays an essential role in various transportation applications, such as, travel cost prediction [25, 39], path recommendation [14, 66], traffic analysis [5, 23], and routing [15, 40]. The goal of path representation learning (*PRL*) is to learn path representations, which are distinguishable for different paths in a transportation network and hence benefit various downstream transportation applications. Existing works learn PRs based on supervised learning, which suffers from two shortcomings. First, supervised learning needs many labeled data for effective training. Second, the supervised learning method returns a task-specific PR, which achieves good performance on tasks with the label. In contrast, these learned path representations cannot achieve better performance on other tasks. These two limitations constrain the supervised learning widely used in generic path representation learning scenarios. In addition, although several studies investigate graph representation learning, these studies cannot capture the sequence dependencies in a path.

To address the above-mentioned limitations, in Paper C [62], we propose a novel unsupervised path representation learning framework Path InfoMax (*PIM*), which consists of a curriculum negative sampling strategy and a task-unspecific path representation learning approach [62]. In particular, we present a negative sample construction strategy based on the key principle of curriculum learning, where we first generate the most accessible negative paths that are entirely different from the input path (i.e., no overlap exists between input and these negative paths). Then, we gradually increase the generation difficulty that makes the input path with overlap with negative paths. Next, we jointly maximize the mutual information of path-path discriminator and path-node discriminator to learn task-unspecific PRs. The contributions of the paper C are summaried as follows:

- We study a curriculum negative sampling strategy to enhance the path representation learning [62].
- We propose the path-path and path-node discriminators to learn path representations jointly from a global and a local view [62].
- We conduct extensive experiments on two datasets with two downstream tasks to demonstrate the effectiveness of *PIM* [62].

2 Preliminaries

We first give the preliminaries and problem definition. The following definitions in this section are reproduced from [62].

2.1 Basic Concepts

Road Network Graph. We define road network graph as a directed graph $G = (\mathbb{V}, \mathbb{E})$, where $\mathbb{V}$ represents the node set and $\mathbb{E}$ denotes the edge set and we define $|\mathbb{V}| = N$ and $|\mathbb{E}| = M$. Each node $V_i \in \mathbb{V}$ is assigned with a node representation vector $v_i \in \mathbb{R}^D$ [62].

Path. A path $P = \langle V_1, V_2, \ldots, V_Z \rangle$ consists of a sequence of nodes, where Z represents the path length and $P.s = V_1$ and $P.d = V_Z$ denote the source and destination of path P, respectively. The connection between each pair of neighboring nodes (V_k, V_{k+1}) is defined as an edge in $\mathbb{E}$, $1 \leq k < Z$ [62]. In particular, we concatenate the node representation vectors in a path to achieve $IV(P) \in \mathbb{R}^{Z \times D}$. We define $IV(P_i)$ as the initial view of path P_i.

2.2 Problem Definition

Given a set of path $\mathbb{P}$ in road network graph G, the goal of *Path Representation Learning (PRL)* is to learn a task-unspecific path representation vector $p_i \in \mathbb{R}^{D'}$ for each path $P_i \in \mathbb{P}$ [62]. In reality, *PRL* learns a path encoder PE_ψ that takes the initial view $IV(P_i)$ of path P_i as input, i.e., the node features of the nodes in path P_i, and returns the corresponding path representation p_i [62].

$$PE_\psi : \mathbb{R}^{Z \times D} \rightarrow \mathbb{R}^{D'}, \tag{3.1}$$

where ψ is the path encoder parameters, which is able to learn through training, Z is the length of path P_i, and $D' \ll Z \times D$ represents the dimension of the learned path representation vector p_i [62].

3 Path InfoMax

Figure 3.1 illustrates the outline of the Path InfoMax (PIM). The design of *PIM* contains a path encoder, path-path discriminator, and path-node discriminator. In particular, *PIM* utilize contrastive learning to train the path encoder to generate PRs without requiring task-specific labels [62].

The trained path encoder takes as input the initial view of an input path and returns corresponding path representations [62]. The path encoder is trained by maximizing the mutual information of the path-path discriminator and path-node discriminator. Then, we construct negative paths for each input path based on curriculum learning principles. Subsequently, the path-path discriminator and path-node discriminator guide the path encoder to generate path representations from the global and local view of an input path. Efficient learn path representations. Finally, we give the objective function of *PIM*.

3.1 Path Encoder

We know that a path contains a sequence of nodes. We utilize the path encoder PE_ψ(e.g., recurrent neural network [4, 21] or a Transformer [49]) that can embed the sequential data into a corresponding PRs, where ψ is the parameters of the path encoder and we aim to learn it during training. In this paper, we use LSTM as our path encoder.

3.2 Curriculum Negative Sampling

Inspired by the principle of curriculum learning [1], we present a curriculum negative sampling strategy to construct negative paths. Here, we first construct the negative paths that are totally different from an input path, e.g., no overlap between the negative paths and the input path [62]. This enables

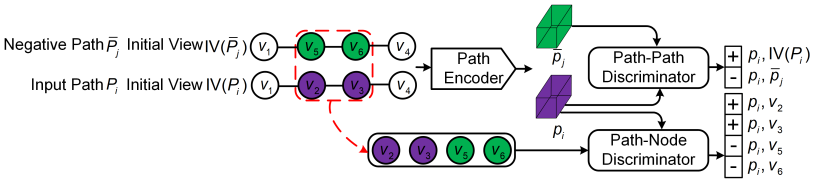


Figure 3.1: *PIM* Overview. The Path Encoder takes as input the initial view $IV(P_i)$ of input path P_i and the initial view $IV(\bar{P}_j)$ of negative path $\bar{P}_j$, and returns their representations p_i and $\bar{p}_j$, respectively. The Path-Path Discriminator takes as input a pair of path representations and decides whether they are from the same path. A positive pair, e.g., is$(p_i, IV(P_i))$, refers to two different representation views of the same input path P_i. A negative pair, e.g., $(p_i, \bar{p}_j)$, refers to the path representations of an input path vs. its negative path. The Path-Node Discriminator takes as input a (input path representation, node feature vector) pair and decides whether the node is from the input path. A positive pair, e.g., (p_i, v_2), represents the path representation of P_i and a node feature vector of node v_2 that only appears in P_i. A negative pair, e.g., (p_i, v_5), represents the path representation of the input path and a node feature vector of node v_5 that only appears in the negative path [62]. © 2021 IJCAI

the path encoder to produce distinguishable path representations more easily. Then, we gradually construct negative paths that have higher similarity with the input path, e.g., with the same source and destination and with increasingly overlapping nodes [62]. To this end, it can be difficult to train a path encoder that returns distinguishable path representations of the input path and negative paths [62]. Figure 3.2 gives three negative paths $\bar{P}_1$, $\bar{P}_2$, and $\bar{P}_3$ with increasingly difficulties for input path P_1, along with the underlying road network graph [62].

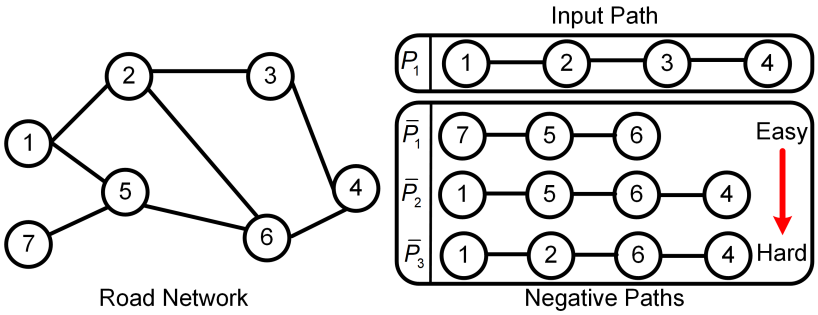


Figure 3.2: Curriculum Negative Sampling [62]. © 2021 IJCAI

3.3 Global Mutual Information Maximization

We investigate a path-path discriminator that trains the path encoder from a global view of the path representations, which aims at the learned path repre-

sentations that are distinguishable from the negative path representations.

To enable the *path-path discriminator* $D^{PP}_{\omega_1}$ training, we first generate negative and positive pairs. As for negative pair, $\langle (p_i, \bar{p}_j), - \rangle$, p_i and $\bar{p}_j$ represent the PRs of input path P_i and a negative path $\bar{P}_j$, respectively, which are both returned by the path encoder PE_ψ [62]. In contrast, as for positive pair, $\langle (p_i, IV(P_i)), + \rangle$, p_i still represent the path representations of input path P_i output by the path encoder and $IV(P_i)$ denotes the initial view of path P_i [62]. Here, p_i and $IV(P_i)$ are two different views, i.e., a view from the path encoder vs. a view from the node features, of the same input path P_i [62]. Figure 3.1 gives an example of a negative sample construction.

Next, the mutual information maximization is used to train the path-path discriminator $D^{PP}_{\omega_1}$, which enables it to conduct a binary classification of the negative vs. positive pairs [62]. In particular, we try to maximize the estimated mutual information (MI) between the positive and negative pairs [62].

$$\underset{\psi, \omega_1}{\operatorname{argmax}} \sum_{P_i \in \mathbb{P}} I_{\psi, \omega_1}(p_i, \mathbb{NP}_i),$$

where $I_{\psi,\omega_1}(\cdot, \cdot)$ is the MI estimator modeled by the path-path discriminator $D^{PP}_{\omega_1}$ that is parameterized by parameters ω_1 and the path encoder PE_ψ that is parameterized by parameters ψ [62]. Path P_i is an input path from $\mathbb{P}$, and p_i is its path representation returned by the path encoder [62]. $\mathbb{NP}_i$ includes the negative paths of P_i [62]. Motivated by [20, 50], we employ a noise-contrastive type objective with a standard binary cross-entropy loss on the positive pairs and the negative pairs, as shown in Equation 3.2 [62].

$$\begin{aligned} \mathcal{I}_{\psi,\omega_1}(p_i, \mathbb{NP}_i) := & \frac{1}{1 + |\mathbb{NP}_i|} (\mathbb{E}_{\mathbb{P}} \left[\log D^{PP}_{\omega_1}(p_i, IV(P_i)) \right] + \\ & \sum_{\bar{P}_j \in \mathbb{NP}_i} \mathbb{E}_{\mathbb{NP}_i} \left[\log \left(1 - D^{PP}_{\omega_1} \left(p_i, \bar{p}_j \right) \right) \right]) \end{aligned} \tag{3.2}$$

3.4 Local Mutual Information Maximization

We then study the path-node path discriminator that trains the path encoder from a local view perspective, which tries to distinguish the learned path representation from the node feature vectors of the nodes from input vs. negative paths [62].

We further choose the nodes only appear in the input path but not appear in negative paths as a positive node set $\mathbb{X}_i$ [62]. In contrast, we select the nodes only appear in negative paths but not appear in input path P_i as a negative node set $\mathbb{Y}_i$ [62]. We then train a *path-node discriminator* $D^{PN}_{\omega_2}$ based on negative and positive node sets. We let $\langle (p_i, v_j), - \rangle$ as a negative pair, where p_i denotes the PR of P_i, v_j represents the node representation of a negative node V_j [62]. Similarly, We treat $\langle (p_i, v_k), + \rangle$ as positive set, v_k is the node

representation of a positive node $V_k \in \mathbb{X}_i$. Figure 3.1 gives an example of a path-node discriminator.

We also conduct mutual information maximization to train the path-node discriminator $D^{PN}_{\omega_2}$ [62]. Specifically, we have

$$\underset{\psi,\omega_2}{\operatorname{argmax}} \sum_{P_i \in \mathbb{P}} I_{\psi,\omega_2}(p_i, \mathbb{X}_i \cup \mathbb{Y}_i),$$

where I_{ψ,ω_2} denotes the MI estimator for path-node discriminator $D^{PN}_{\omega_2}$ and ω_2 is the parameters for the path-node discriminator. In addition, ψ denotes the parameters of the path encoder PE_ψ. Similar with the path-path discriminator, we also utilize a noise-contrastive with a BCE loss to calculate $I_{\psi,\omega_2}(p_i, \mathbb{X} \cup \mathbb{Y})$.

$$\begin{aligned} I_{\psi,\omega_2}(p_i, \mathbb{X}_i \cup \mathbb{Y}_i) := & \frac{1}{|\mathbb{X}_i \cup \mathbb{Y}_i|}\Big(\sum_{v_k \in \mathbb{X}_i} \mathbb{E}_{\mathbb{X}_i}\left[\log \mathcal{D}^{PN}_{\omega_2}(p_i, v_k)\right] + \\ & \sum_{v_j \in \mathbb{Y}_i} \mathbb{E}_{\mathbb{Y}_i}\left[\log\left(1 - D^{PN}_{\omega_2}\left(p_i, v_j\right)\right)\right]\Big) \end{aligned} \tag{3.3}$$

3.5 Maximization of *PIM*

We jointly maximize the global and local mutual information to train the final *PIM*, which is formulated as follows:

$$\underset{\psi,\omega_1,\omega_2}{\operatorname{argmax}} \sum_{P_i \in \mathbb{P}} \left(I_{\psi,\omega_1}(p_i, \mathbb{NP}_i) + I_{\psi,\omega_2}(p_i, \mathbb{X}_i \cup \mathbb{Y}_i)\right).$$

4 Experiments

4.1 Experimental Setup

Road Network and Paths

We first consider the road network of Aalborg, Denmark, which includes 8,893 nodes and 10,045 edges [62]. Then, we use the road network of Harbin, China to verify the effectiveness of *PIM*. This dataset contains 5,796 nodes and 8,498 edges. We achieve the road network of both datasets from OpenStreetMap. In particular, we utilize 52,494 paths in the Aalborg network and 37,079 paths in the Harbin network [62].

4.2 Downstream Task

We consider two different downstream tasks: path travel time estimation and path ranking [62]. We use the Mean Absolute Error (**MAE**), Mean Absolute

Relative Error (**MARE**), and Mean Absolute Percentage Error (**MAPE**) as the evaluation metric of the travel time estimation task [62]. We employ **MAE**, Kendall's rank correlation coefficient (denoted by τ), and Spearman's rank correlation coefficient (denoted by ρ) to evaluate the performance of the path ranking [62].

4.3 Baselines

We select seven baselines for the effectiveness study of *PIM*. We select **Node2vec** [12], Deep Graph InfoMax (**DGI**) [50], and Graphical Mutual Information Maximization (**GMI**) [41] as they are popular graph embedding methods. Next, we choose the most recent representation learning methods **Memory Bank (MB)** [58], **InfoGraph** [46] and **BERT** [8]. Finally, we also select *PathRank* as a supervised baseline.

4.4 Regression Model

We select Gaussian Process Regressor (GPR) to estimate the path travel time and path ranking score based on the learned path representation for all unsupervised learning methods.

4.5 Experimental Results

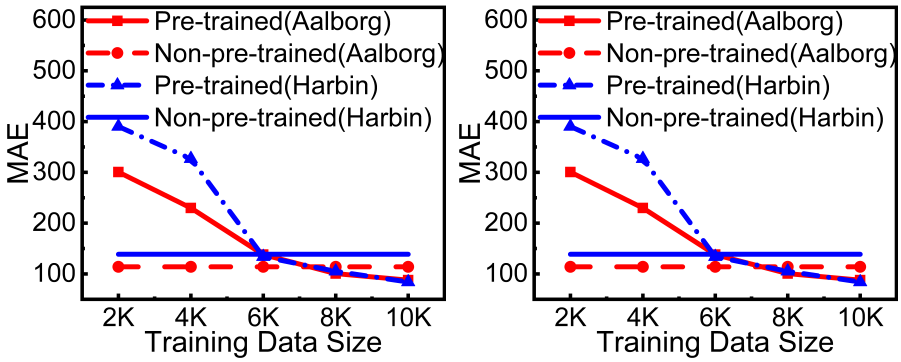
Overall accuracy on both downstream tasks

Table 3.1 gives the overall results for travel time and ranking score estimation. We can see that *PIM* achieve the best results on both tasks under two road network when compared to *Node2vec*, *DGI*, *GMI*, *MB*,*BERT*, and *InfoGraph*. This highlight the effectiveness of *PIM*.

Using *PIM* as a Pre-training Method

In this phase, we treat *PIM* as a pre-training approach for the supervised method *PathRank* [62]. *PathRank* uses a GRU that takes as input a sequence of the node feature vector in a path and estimates path travel time or ranking scores [62]. To enable *PIM* as a pre-training method for *PathRank*, we choose GRU as our path encoder [62]. In particular, we first train the path encoder in the form of *PIM*, i.e., in an unsupervised manner. Then, we initialize the parameters of the GRU path encoder in *PathRank* based on the learned parameters of the GRU path encoder in *PIM*. Next, the labeled training paths are used to fine-tune the model *PathRank*.

Figure 3.3 gives the performance comparison of travel time and ranking score estimation whether considering pre-training or not. When we do not



(a) Travel Time Estimation. **(b)** Path Ranking.

Figure 3.3: Effects of Pre-training [62]. © 2021 IJCAI

employ pre-training, the *PathRank* is trained by using 10K labeled training paths [62]. We can see that: (1) Pre-training makes the *PathRank* obtain the same accuracy using less labeled training paths [62]. (2) Pre-training makes the *PathRank* achieve higher accuracy when using the same number of labeled training paths.

Table 3.1: Overall Accuracy on Travel Time Estimation and Ranking Score Estimation [62]. © 2021 IJCAI

Method	Aalborg						Harbin					
	Travel Time Estimation			Path Ranking			Travel Time Estimation			Path Ranking		
	MAE	MARE	MAPE	MAE	τ	ρ	MAE	MARE	MAPE	MAE	τ	ρ
Node2vec	121.43	0.27	31.04	0.18	0.66	0.70	258.91	0.22	23.17	0.15	0.70	0.72
DGI	192.63	0.42	82.44	0.54	0.49	0.52	528.71	0.39	86.53	0.21	0.59	0.60
GMI	136.58	0.30	50.81	0.23	0.58	0.61	979.68	0.73	192.45	0.24	0.55	0.56
MB	243.97	0.53	84.17	0.35	0.34	0.38	533.41	0.40	86.01	0.27	0.31	0.34
BERT	254.17	0.54	61.61	0.36	0.38	0.39	514.95	0.57	49.80	0.28	0.45	0.46
InfoGraph	132.28	0.29	39.47	0.17	0.69	0.73	391.45	0.44	44.60	0.29	0.68	0.72
PIM	**76.10**	**0.16**	**17.28**	**0.12**	**0.72**	**0.76**	**125.76**	**0.14**	**13.73**	**0.11**	**0.75**	**0.79**

Chapter 4

Weakly-supervised Temporal Path Representation Learning with Contrastive Curriculum Learning

This chapter offers a brief summary of Paper D [63] and does not offer additional contributions. This chapter reuses content from paper D when that is considered most effective.

1 Problem Motivation and Statement

Temporal variation is critical in transportation applications since it represents the different types of traffic conditions, e.g., peaking hours vs. off-peaking hours. Thus, temporal path representation learning is able to facilitate various downstream task. However, it is non-trivial to learn task-unspecific temporal path representations (TPRs) through supervised or unsupervised learning. Supervised methods (Figure 4.1(a)) learn task-specific TPRs [32, 64], which generalize poorly on other tasks and need a larger amount of labeled training samples to train the model effectively. In contrast, unsupervised methods learn a task-unspecific path representation without using task-specific labels. However, existing unsupervised path representation learning methods mainly focus on unsupervised graph representation learning, where we achieve the node representation first and then aggregate node representations into a path representation. In addition, existing graph representation learning methods ignore the temporal aspect, resulting in the learned path representation also

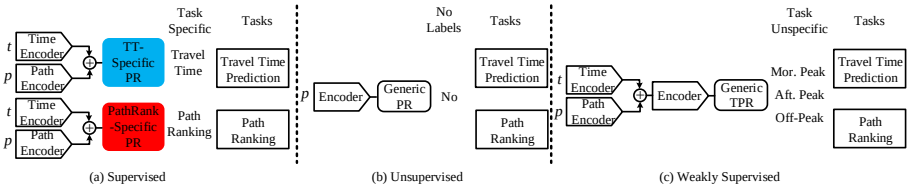


Figure 4.1: Supervised, Unsupervised, and Weakly-Supervised methods for learning Temporal Path Representations (TPR): (a) Supervised learning relies on task-specific labels to obtain task-specific path representations (PRs), and thus fails to generalize across tasks; (b) Unsupervised learning produces generic path representations for use in different tasks, but fails to capture temporal traffic aspects of paths; (c) Weakly supervised learning (**Ours**) uses weak labels to learn generic TPRs [63]. © 2022 IEEE ICDE

lacking temporal information. This will degrade the estimation performance of downstream applications.

To address the aforementioned challenge, in Paper D [63], we propose using the weakly-supervised contrastive (WSC) learning principle to enable the temporal path encoder training such that we can achieve task-unspecific temporal path representations. Then, we further introduce two-stage curriculum learning into a weakly-supervised contrastive learning framework to improve the temporal path encoder training [63]. The contributions of the paper D are summaried as follows:

- We formulate the temporal path representation learning problem [63].
- We propose a weakly-supervised, contrastive model (basic framework) to learn generic path representations that take temporal information into account [63].
- We integrate curriculum learning into the weakly-supervised contrastive model to further enhance the learned temporal path representations, yielding the advanced framework [63].
- We report on extensive experiments using three real-world data sets on three downstream tasks to assess in detail the effectiveness of the proposed framework [63].

2 Preliminaries

We first give the preliminaries and problem definition. The following definitions in this section are reproduced from [62].

2.1 Basic Concepts

Road Network Graph. We consider road network graph as a directed graph $G = (\mathbb{V}, \mathbb{E})$, where $\mathbb{V}$ represents a set of vertices v_i that denotes intersections and $\mathbb{E} \subset \mathbb{V} \times \mathbb{V}$ is a set of edges $e_i = (v_j, v_k)$ that denotes edges [63].

GPS Trajectory. A GPS trajectory of moving object is defined as $traj = \langle (l_i, t_i) \rangle_{i=1}^{|traj|}$, which includes a sequence of location l_i with specific timestamp t_i [63].

Path. A path is defined as a sequence of neighboring edges $\mathbf{p} = \langle e_i \rangle_{i=1}^{|\mathbf{p}|}$, where $e_i \in \mathbb{E}$ denotes the i-th edge in the path [63].

Temporal Path. A temporal path is defined as $tp = (\mathbf{p}, t)$, where $tp.\mathbf{p}$ denotes a path and $tp.\mathbf{t}$ represents a departure time [63].

2.2 Problem Definition

Given a set of temporal paths $\mathbb{TP} = \{tp_1, tp_2, \ldots, tp_n\}$ and each temporal path tp_i is assigned with a weak label y_i, temporal path representation learning (*TPRL*) aims to train the temporal path encoder and achieve the task-unspecific temporal path representation $TPRL(tp_i)$ for each temporal path $tp_i \in \mathbb{TP}$, which is given in Eq. 4.1 [63].

$$TPRL_\psi(tp_i) : R^{d_{tem}} \times \mathbb{R}^{M \times d} \rightarrow \mathbb{R}^{d_h}, \tag{4.1}$$

where ψ is the parameters of path encoder, M represents the total number of edges in the path, d, d_{tem}, and d_h denote the feature dimensions for an edge, a departure time embedding, and a resulted temporal path representation, respectively [63].

3 Temporal Path Encoder

Figure 4.2 shows the outline of the basic framework of WSC. We observe that the temporal path encoder includes a spatial embedding layer, a temporal embedding layer, and an LSTM layer [63].

3.1 Temporal Embedding

The temporal information is fed into the temporal embedding layer, which returns temporal feature representations. In this case, we first build the temporal graph $G' = (\mathbb{V}', \mathbb{E}')$. In temporal graph, each node $v' \in \mathbb{V}'$ represents a departure time slot and each edge $e' \in \mathbb{E}'$ represents a connection between two

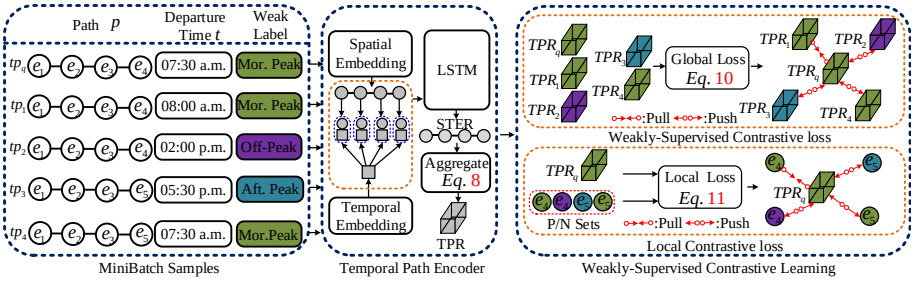


Figure 4.2: Illustration of Basic Framework. Given a set of temporal paths in a minibatch, an input temporal path is encoded into a feature map by the temporal path encoder. The global loss framework takes a (query temporal path tp_q, positive or negative temporal path) pair as input and pulls together the TPRs of the query path and positive path, while pushing away the TPRs of the query path and negative paths. The local loss takes as input a TPR of query path and a spatio-temporal edge representation (STER) and brings a TPR of a query path with the positive edge representations closer while pushing apart TPR of a query path with the negative edge representations [63]. © 2022 IEEE ICDE

time slots [63]. Then, we employ node2vec [12] to learn node representations in the temporal graph [63], which is formulated as:

$$t^{all} = \text{Node2Vec}^{tg}(t_g^{emb}), \tag{4.2}$$

where $t^{all} \in R^{d_{tem}}$ represent the finalized temporal feature vector. t_g^{emb} denotes initialized node feature vector.

3.2 Spatial Embedding

We consider four spatial edge categories, including *Road Type (RT)*, *Number of Lanes (NoL)*, *One Way (OW)*, and *Traffic Signals (TS)*. Then, we first define these spatial features as one-hot vectors, which is denoted as: $s_{RT}^{one} \in R^{n_{rt}}$, $s_{NoL}^{one} \in R^{n_l}$, $s_{OW}^{one} \in R^{n_o}$, $s_{TS}^{one} \in R^{n_{ts}}$, where n_{rt}, n_l, n_o, n_{ts} are the values for these different spatial features. Next, we achieve the final spatial representation by concatenating these four feature representations for edge s_i, which is given as follows:

$$s^{type} = [s_{RT}^{emb}, s_{NoL}^{emb}, s_{OW}^{emb}, s_{TS}^{emb}], \tag{4.3}$$

where $[\cdot,\cdot]$ represents concatenation operation.

In addition, to further capture the road network topology, we employ *node2vec* [12] to learn the representations for road network graph, which can be described as: $n_{vi}^{rn} = \text{Node2Vec}^{rn}(n_{vi}^{one})$, where vi^{one} is initialized node feature, e.g., one-hot vector, for node v_i. $n_{vi}^{rn} \in R^{\frac{d_{top}}{2}}$ denotes the feature vector returned by *node2vec*. We can define the final edge representation as:

$$s_{e_k}^{rn} = [n_{v_i}^{rn}, n_{v_j}^{rn}], \tag{4.4}$$

where v_i and v_j denote start and end nodes of edge e_k.

In this case, the final feature vector for the edge e_k can be formulated as:

$$s_{e_k}^{all} = [s_{e_k}^{rn}, s^{type}], \tag{4.5}$$

where $s_{e_k}^{all} \in R^d$ represents the final spatial feature vector for an edge e_k and $d = d_{rt} + d_l + d_o + d_{ts} + d_{top}$.

3.3 LSTM Encoder

Given a temporal path $tp = (\mathbf{p}, t)$, where $tp.\mathbf{p}$ denotes a path and $tp.\mathbf{t}$ denotes a departure time [63]. We get a sequence of Spatio-temporal edge representations $\langle \mathbf{x}_{e_1}, \mathbf{x}_{e_2}, \cdots \mathbf{x}_{e_\mathbf{p}} \rangle$, where $\mathbf{x}_{e_i} = [\mathbf{t}^{all}, \mathbf{s}_{e_i}^{all}]$, where $\mathbf{t}^{all}$ denotes the temporal feature vector for $tp.t$. Then, the LSTM encoder takes as input Spatio-temporal edge representations and returns temporal path representations through aggregate function.

$$\hat{\mathbf{p}} = \langle \mathbf{p}_{e_1}, \mathbf{p}_{e_2}, \cdots \mathbf{p}_e \rangle = \text{LSTM}(\langle \mathbf{x}_{e_1}, \mathbf{x}_{e_2}, \cdots \mathbf{x}_e \rangle), \tag{4.6}$$

$$\vec{h}_p = \frac{\sum_{i=1}^{n} \vec{p}_i}{|\hat{\mathbf{p}}|} \in \mathbb{R}^{d_h}, \tag{4.7}$$

where $\vec{p}_{e_j} \in R^{d_h}$ denotes edge e_j's spatio-temporal representations that captures the sequence dependencies. $\vec{h}_p$ denotes the temporal path representations. $\vec{p}_i$ denotes the latent feature vector of edge i in a path. $\hat{\mathbf{p}}$ denotes the average edge representations.

4 Weakly-Supervised Contrastive Learning

4.1 Generation of Positive and Negative Samples

To enable the weakly-supervised contrastive learning, we first generate the positive and negative samples. Given a set of temporal paths (TPs), positive TPs are defined as: (1) different temporal path representations of the same path; (2) TPs pass through the same path and have the same weak labels, e.g., peaking hours and off-peaking hours [63]. While we further define the negative TPs as (1) sample paths with different weak labels, (2) different paths with the same weak labels, (3) different paths with different weak labels. To this end, we are able to construct many positive and negative TPs for a query temporal path (TP) [63]. Figure 4.2 gives the MiniBatch example, where we consider $tp_q, tp_1, tp_2, tp_3,$ and tp_4 five TPs with different weak labels, i.e., morning peak (Mor. Peak), Afternoon peak (Aft. Peak), and Off-Peak [63]. Suppose tp_q is a query temporal path TP, tp_1 is the positive sample since these

two paths are the same and also have the same departure weak label (i.e., Mor. Peak) [63]. In contrast, $tp_2, tp_3,$ and tp_4 are negative samples according to the rules defined before.

4.2 Global Weakly-supervised Contrast Loss

Although we have witnessed the significant success of the self-supervised, the contrastive loss for self-supervised cannot consider the difference among negative samples. Inspired by *SupCon* [29], the good framework should have the ability to leverage the information between samples with same class and distinguish them from samples in other classes [63]. Toward this end, we study weakly-supervised contrastive learning, which uses positive and negative samples generated in Section 4.1. Thus, our global WSC can be defined as:

$$\mathcal{L}^{global} = \sum_{(\boldsymbol{tp}_i, \mathbf{y}_i) \in \mathbb{P}} \mathcal{L}^{global}_{(\boldsymbol{tp}_i, \mathbf{y}_i)} = \sum_{(\boldsymbol{tp}_i, \mathbf{y}_i) \in \mathbb{P}} \frac{1}{|\mathbb{S}_{\boldsymbol{tp}_i}|} \sum_{\boldsymbol{tp}_j \in \mathbb{S}_{\boldsymbol{tp}_i}} \log \frac{\exp\left(\text{sim}(TPR_i, TPR_j)\right)}{\sum_{\boldsymbol{tp}_k \in \mathbb{N}_{tp_i}} \exp\left(\text{sim}(TPR_i, TPR_k)\right)}, \tag{4.8}$$

where $\text{sim}(\cdot)$ represents cosine similarity function that measures the similarity between two TPRs; $\mathbb{P} = \{(tp_i, y_i)\}_{i=1}^{B}$ denotes temporal paths in the training batch and y_i denotes departure weak label for tp_i; $\mathbb{S}_{tp_i} = \{tp_j\}_{j=1}^{|\mathbb{S}_{tp_i}|}$ is the positive sample set for query tp_i, where $y_i = y_j$ and $tp_i.\mathbf{p} = tp_j.\mathbf{p}$; and $\mathbb{N}_{tp_i} = \mathbb{P} \setminus \{\boldsymbol{tp}_i \cup \mathbb{S}_{\boldsymbol{tp}_i}\}$ is the negative sample set for query tp_i [63].

4.3 Local WSC loss

Instead of conducting weakly-supervised learning between query TP and global positive and negative TPs, we further employ local WSC loss that captures a local difference between positive and negative edge samples. In particular, Local WSC aims to make TPRs close to a representation of positive edges and be distant from representations of negative edges. We randomly choose edges in positive temporal paths as positive edges (denoted by $\mathbb{PN}_i$) [63]. In contrast, we select edges showing in negative temporal paths as our negative edge set (denoted by $\mathbb{NN}_i$) [63]. Therefore, the objective of local WSC is defined as:

$$\mathcal{L}^{local} = \sum_{(\boldsymbol{tp}_i, \mathbf{y}_i) \in \mathbb{P}} \frac{1}{|\mathbb{PN}_i|} \log \frac{\sum_{(e_j, y_i) \in \mathbb{PN}_i} \exp\left(s(TPR_i, e_j)\right)}{\sum_{(e_k, y_j \neq y_i) \in \mathbb{NN}_i} \exp\left(s(TPR_i, e_k)\right)}, \tag{4.9}$$

where $\mathbb{PN}_i$ and $\mathbb{NN}_i$ represent the positive and negative edge sets, and y_i is the weak label for edge representation, which inherits from the corresponding temporal path [63].

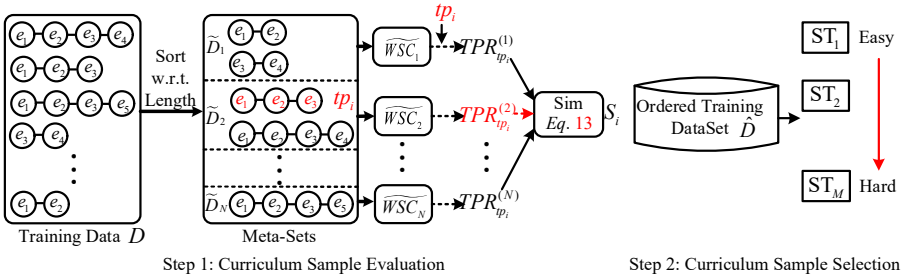


Figure 4.3: Illustration of Advanced Framework [63]. © 2022 IEEE ICDE

4.4 Objective for WSC

To enable temporal path encoder training, we jointly maximize the global and local WSC loss. Thus, we have:

$$\mathcal{L} = \arg\max_{\psi} \sum_{i \in I} \lambda \cdot \mathcal{L}_i^{global} + (1 - \lambda) \cdot \mathcal{L}_i^{local}, \tag{4.10}$$

where λ represents a balancing factor. I denotes the training batch sets [63].

5 Contrastive Curriculum Learning

5.1 Curriculum Samples Evaluation

As shown in Figure 4.3, to conduct curriculum learning on training temporal paths, we need first to evaluate the difficulty of each temporal path through model learning instead of heuristic ways. We first rank all temporal training sets $\mathbb{D}$ based on their length. Then, we partition $\mathbb{D}$ into N non-overlapping meta-sets, i.e., $\mathbb{D} = \{\widetilde{\mathbb{D}}_1, \widetilde{\mathbb{D}}_2, \cdots, \widetilde{\mathbb{D}}_N\}$, where $\widetilde{\mathbb{D}}_i$ denotes the i-th meta-set, $\widetilde{\mathbb{D}}_i \cap \widetilde{\mathbb{D}}_j = \emptyset, \forall i \neq j, i, j \in [1, N]$ [63]. Subsequently, we learn N unrelated WSC models in terms of the different meta-sets, i.e., $\widetilde{WSC}_i, i = [1, N]$. In particular, we only consider i-th meta-set, $\widetilde{\mathbb{D}}_i$ when training the $\widetilde{WSC}_i$ model. Finally, we achieve a number of N trained unrelated WSC models, which is treated as *Experts*, to measure the difficulty of each temporal path [63].

We then utilize the *Experts* to compute the difficulty scores for each training sample [63]. We feed a temporal path $\boldsymbol{tp}_i \in \widetilde{\mathbb{D}}_j$ into each *Expert*, $\widetilde{WSC}_j, j \in$

$[1, N]$, which results in a total of N TPRs, i.e., $TPR^{(1)}_{\boldsymbol{tp}_i}, TPR^{(2)}_{\boldsymbol{tp}_i}, \cdots, TPR^{(N)}_{\boldsymbol{tp}_i}$ [63]. Considering $\boldsymbol{tp}_i$ coming from $\widetilde{\mathbb{D}}_j$, we treat $TPR^{(j)}_{\boldsymbol{tp}_i}$ as the ground truth, and compute the similarity between $TPR^{(j)}_{\boldsymbol{tp}_i}$ and $TPR^{(k)}_{\boldsymbol{tp}_i}$, where $\forall k \neq j, k \in [1, N]$ [63]. After that, we add all corresponding similarity scores and define them as the difficulty score (denoted by S_i) for the temporal path tp_i, which is given in Eq. 4.11.

$$S_i = \sum_{k=1, k\neq j}^{N} Sim(WSC_j(tp_i), WSC_k(tp_i)), \tag{4.11}$$

where $Sim(\cdot, \cdot)$ represent the similarity function. Finally, we can achieve the difficulty score for each training paths, which makes $\mathbb{D} = \{(\boldsymbol{tp}_1, S_1), \cdots, (\boldsymbol{tp}_N, S_N)\}$, where $(\boldsymbol{tp}_i, S_i)$ is one element and S_i is the difficulty score for temporal path $\boldsymbol{tp}_i$ [63].

5.2 Curriculum Sample Selection

We further sort all temporal training paths based on their difficult scores and split the training path sets into M subsets with respect to different difficulty levels, which range from ST_1 (the easiest) to ST_M (the hardest). Then, the WSC is trained for one epoch in terms of each stage. After that, we train WSC on another stage ST_{M+1}, until WSC coverage, where the $ST_{M=1}$ contains the entire training paths.

6 Experiments

6.1 Experiments Setup

Data sets

We consider three datasets: Aalborg, Denmark; Harbin, China; and Chengdu, China. We obtain the corresponding road network from OpenStreetMap [63]. More specifically, the Aalborg data set contains 10,017 nodes and 11,597 edges, and we achieve 28,370 paths after map matching [63]. Harbin's data set includes 8,497 nodes and 14,497 edges. After map matching, we have 58,977 paths. Chengdu data set consists of 6,632 nodes and 17,038 edges [63]. We obtain 57,404 paths after map matching.

Downstream Task

We consider three downstream tasks, including path travel time estimation, path ranking, and path recommendation [63]. As for path travel time estimation, we use Mean Absolute Error (**MAE**), Mean Absolute Relative Error

(**MARE**), and Mean Absolute Percentage Error (**MAPE**) to evaluate the accuracy [63]. While for Path Ranking, we utilize **MAE**, the Kendall rank correlation coefficient (τ) [28], and the Spearman's rank correlation coefficient (ρ) [72] as the evaluation metrics [63]. Finally, we select **Accuracy** (Acc.) and **Hit Rate** (HR) to evaluate the recommendation effectiveness.

Baselines

We first consider three graph embedding methods **Node2vec** [12], Deep Graph InfoMax (**DGI**) [50], and Graphical Mutual Information Maximization (**GMI**) [41]. We then consider four unsupervised representation learning methods: **Memory Bank (MB)** [58], **InfoGraph** [46], **BERT** [8], and **PIM** [62] since they are representative methods in different domains. Next, we also consider five supervised methods: **DeepGTT** [33], **HMTRL** [35], **PathRank** [64], **GCN** [7], and **STGCN** [69].

Models for Downstream Tasks

Based on the learned path representations, we use ensemble model *Gradient Boosting Regressor* (GBR) to estimate path travel time and ranking score as they are regression problems [63]. We employ ensemble model *Gradient Boosting Classifier* (*GBC*) to predict the path recommendations since it is classification problem [63].

Weak Label

We select two different types of weak labels, consisting of peak/off-peak (POP) and traffic congestion indices (TCI). We treat POP as default weak labels [63].

6.2 Experimental Results

Overall accuracy on downstream task

Table 4.1, Table 4.2 and Table 4.3 illustrate the overall accuracy in terms of three downstream tasks. We observe that (1) *WSCCL* outperforms all other unsupervised baselines (i.e., graph embedding methods and representation learning methods) on the three downstream tasks for three road network data sets. (2) *WSCCL* also achieves the best results compared with supervised baselines on these three tasks since we use the small size of training labeled data to train these models.

Table 4.1: Overall Accuracy on Travel Time Estimation [63]. © 2022 IEEE ICDE

Methods	Aalborg		Harbin		Chengdu	
	MAE	**MAPE**	**MAE**	**MAPE**	**MAE**	**MAPE**
Node2vec	63.82	45.67	269.21	31.41	290.47	34.43
DGI	67.22	49.36	288.09	34.01	312.28	38.46
GMI	70.61	52.40	310.39	36.60	337.06	41.58
MB	57.32	39.37	315.25	35.28	333.73	42.45
BERT	71.96	45.42	217.96	24.52	303.00	36.77
InfoGraph	69.36	41.28	200.81	22.68	291.54	36.07
PIM	57.66	39.34	196.06	21.96	289.10	35.55
DeepGTT	44.78	26.53	214.95	22.76	305.08	35.47
HMTRL	40.59	21.81	228.58	23.60	360.08	37.33
PathRank	37.09	23.89	190.08	20.12	334.94	35.11
GCN	78.04	53.05	368.21	35.62	480.83	42.01
STGCN	58.57	38.97	284.12	23.48	406.09	33.58
WSCCL	**31.66**	**21.39**	**178.89**	**19.43**	**281.20**	**33.30**

Table 4.2: Overall Accuracy on Path Rank Estimation [63]. © 2022 IEEE ICDE

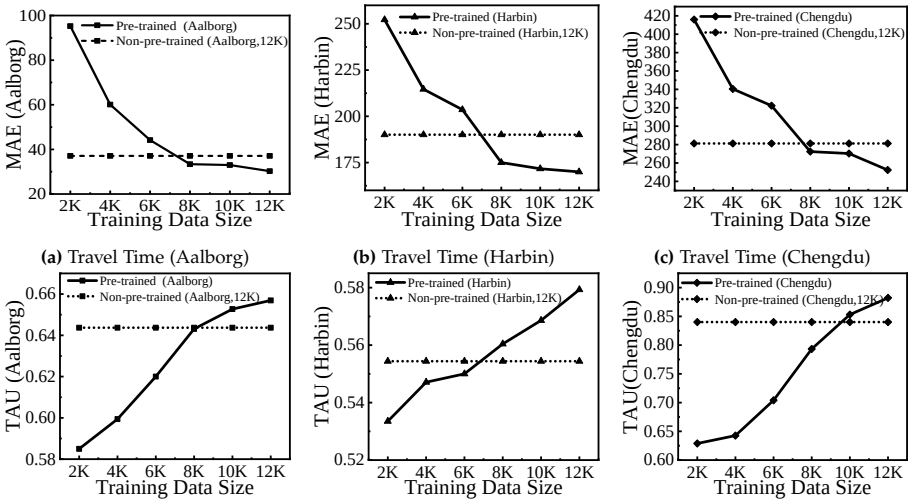
Methods	Aalborg		Harbin		Chengdu	
	MAE	τ	**MAE**	τ	**MAE**	τ
Node2vec	0.23	0.60	0.22	0.37	0.20	0.73
DGI	0.24	0.60	0.21	0.48	0.21	0.52
GMI	0.24	0.59	0.21	0.49	0.21	0.51
MB	0.23	0.62	0.22	0.44	0.20	0.71
BERT	0.26	0.49	0.22	0.46	0.22	0.55
InfoGraph	0.26	0.52	0.21	0.45	0.20	0.73
PIM	0.22	0.60	0.21	0.43	0.19	0.76
DeepGTT	0.39	0.12	0.29	0.04	0.23	0.20
HMTRL	0.17	0.65	0.22	0.51	0.16	0.77
PathRank	0.23	0.64	0.18	0.55	0.17	0.79
WSCCL	**0.15**	**0.68**	**0.14**	**0.68**	**0.13**	**0.84**

Table 4.3: Overall Performance on Path Recommendation [63]. © 2022 IEEE ICDE

Methods	Aalborg		Harbin		Chengdu	
	Acc.	**HR**	**Acc.**	**HR**	**Acc.**	**HR**
Node2vec	0.79	0.51	0.76	0.51	0.75	0.61
DGI	0.74	0.55	0.70	0.36	0.70	0.57
GMI	0.78	0.53	0.72	0.41	0.68	0.58
MB	0.67	0.48	0.61	0.69	0.73	0.69
BERT	0.60	0.43	0.64	0.53	0.66	0.61
InfoGraph	0.72	0.69	0.79	0.78	0.73	0.65
PIM	0.79	0.82	0.86	0.83	0.76	0.74
HMTRL	0.80	0.86	0.81	0.82	0.78	0.83
PathRank	0.77	0.71	0.79	0.74	0.77	0.73
WSCCL	**0.82**	**0.88**	**0.97**	**0.91**	**0.81**	**0.90**

Using *WSCCL* as a Pre-training Method

We further treat *WSCCL* as a pre-training approach for a supervised framework *PathRank*, where we first train *WSCCL* in a weakly-supervised manner, and then we initialize the parameters of the encoder in *PathRank* by using the temporal path encoder's parameters learned in *WSCCL* [63]. Figure 4.4 report the results. We observe that (1) The same performance is achieved by using less labeled data under the pre-training scenario. and (2) We can achieve higher performance when using pre-training in the format of same size data (e.g., 12K).



(a) Travel Time (Aalborg) **(b)** Travel Time (Harbin) **(c)** Travel Time (Chengdu)

(d) Path Ranking (Aalborg) **(e)** Path Ranking (Harbin) **(f)** Path Ranking (Chengdu)

Figure 4.4: Effects of Pre-training [63]. © 2022 IEEE ICDE

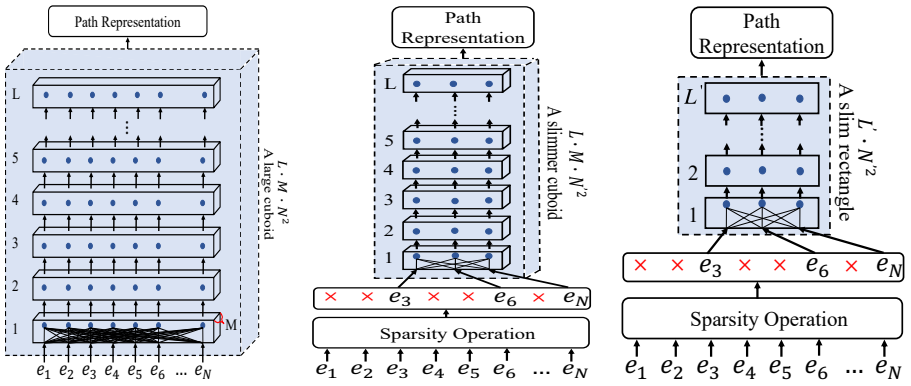
Chapter 5

LightPath: Lightweight and Scalable Path Representation Learning

This chapter offers a brief summary of Paper E [65] and does not offer additional contributions. This chapter reuses content from paper E when that is considered most effective.

1 Problem Motivation and Statement

Recently, we have witnessed increasing intelligent transportation and smart city services that run on movement paths. Thus, path representation learning (PRL) has attracted significant attention [11, 14, 73]. In particular, various intelligent transportation applications are related to paths, e.g., e.g., travel cost estimation [23, 48, 57, 62, 70, 71], routing [15, 26, 39, 53, 56], trajectory analysis [18, 43, 44, 51, 55] and path ranking [35, 62, 64]. To this end, accurate and compact path representations are in high demand due to their ability to greatly improve the services they are used in. However, existing path representation learning approaches primarily focus their attention on accuracy improvement, ignoring scalability and resource usage. This leads to the trained model containing many parameters and layers, resulting in higher computational costs and memory consumption, especially for the long paths (i.e., the path with many edges). More specifically, existing path representation approaches have two limitations: **Poor scalability w.r.t. path length** We know a path consists of a sequence of road network graph edges. Thus path representation learning takes advantage of models skilled in modeling sequence,



(a) Traditional Transformer Encoder **(b)** Sparse Transformer Encoder **(c)** LightPath

Figure 5.1: Encoder Architectures: (a) A traditional transformer encoder with L layers and M heads, takes as input a path (N-Length) and thus has complexity $\mathcal{O}\left(L \cdot M \cdot N^2\right)$; (b) A sparse transformer encoder takes as input a sparse path (i.e., reducing path length from N to N'), resulting in $\mathcal{O}\left(L \cdot M \cdot N'^2\right)$ complexity; (c) *LightPath* further compresses the traditional transformer in terms of layers and heads, yielding complexity $\mathcal{O}\left(L' \cdot N'^2\right)$, making it more scalable and lightweight than a traditional transformer encoder [65].

such as Transformer [49]. However, Transformer-based approaches [3] adopt a self-attention principle, where one edge attends to all other edges in a path in each attention, leading to quadratic complexity, $\mathcal{O}\left(N^2\right)$ of path length N. As a result, such models have poor scalability regarding path length, where path length denotes the number of edges that appear in a path.

Very large model size. To continuously improve the performance of different smart city applications, several existing PRL models are trained with many parameters and layers, which constrains their usage in resource-limited environments. Take Transformer-based approach [3] as example, We achieve Transformer model by stacking L transformer layers, where each layer conducts multi-head (i.e., M heads) attentions. This makes the Transformer models a large cuboid that has a total complexity of $\mathcal{O}\left(L \cdot M \cdot N^2\right)$, as shown in Figure 5.1a, which prevents the traditional Transformer encoder from being used in resource-limited environments.

To address the above-mentioned limitations, in Paper E [65], we propose a lightweight and scalable path representation learning framework *LightPath*. Specifically, we first study the sparse auto-encoder, which reduces the path length from N to N' to achieve good scalability, w.r.t. path length. Thus, this function reduces the complexity from $\mathcal{O}\left(L \cdot M \cdot N^2\right)$ to $\mathcal{O}\left(L \cdot M \cdot N'^2\right)$, as shown in Figure 5.1b. Then, we introduce relational reasoning to enhance the sparse path encoder training. Next, we propose a global-local knowledge distillation strategy to reduce the size of the path encoder, further reducing a slimmer cuboid to a slim rectangle (cf. Figure5.1c). This global-local knowl-

edge distillation enables the lightweight framework. The main contributions of Paper E are summarised as follows:

- *Sparse Auto-encoder.* We propose a sparse auto-encoder framework that takes as input sparse paths with reduced path lengths, achieve good scalability. w.r.t the path length.
- *Relational Reasoning.* We introduce relational reasoning to enable efficient sparse auto-encoder training. Specifically, we propose two types of relational reasoning objectives for accurate and efficient path representation learning. These two objectives regularize each other and yield a more effective path encoder.
- *Global-local Knowledge Distillation.* We propose a novel global-local knowledge distillation framework that enables a lightweight student sparse encoder to mimic a larger teacher sparse encoder from global and local perspectives. The resulting lightweight model can be deployed on edge devices while achieving accurate performance at different downstream tasks.
- *Extensive Experiments.* We report on extensive experiments for two large-scale, real-world datasets with two downstream tasks. The results offer evidence of the efficiency and scalability of the proposed framework as compared with nine baselines.

2 Preliminaries

We first give the preliminaries and problem definition. The following definitions in this section are reproduced from [65].

2.1 Basic Concepts

Road Network Graph. A road network graph is represented as $\mathbf{G} = (V, E)$, where $v_i \in V$ represents a vertex and V denotes a vertex set, $E \subseteq V \times V$ denotes edge set and $e_i = (v_j, v_k)$ is road segments [65].

GPS Trajectory. A GPS trajectory of a vehicle is represented as a sequence of tuple of (location, timestamp) [65].

Path. A path is defined as $p = \langle e_1, e_2, e_3, \cdots, e_N \rangle$, where $e_i \in E$ is an edge in path p. We define $p.\Phi = \langle 1, 2, 3, \cdots, N \rangle$ as a sequence edge order in p [65].

Sparse Path. A sparse path $p' = \{e_i\}_{i \in p'.\Omega}$ consists of a subset of edges in path p, where $p'.\Omega$ is a sub-sequence of $p.\Phi$. We define $p.\Phi$ as the original edge orders of p' in p and $p'.\Omega \subseteq p.\Phi$ [65].
Example. Given a path $p = \langle e_1, e_3, e_4, e_6, e_7 \rangle$ and $p.\Phi = \langle 1, 2, 3, 4, 5 \rangle$, then sparse path $p' = \{e_1, e_4, e_7\}$, where $p'.\Omega = \langle 1, 3, 5 \rangle$, is one of the sparse paths for p.

Edge Representation. The edge representation vector is a vector $\mathbb{R}^{d_k}$, where d_k denotes the dimensionality of the vector [65].

Transformer Layer. Given a sequence of edge representation vectors $\mathbf{X} = \langle x_1, x_2, x_3, \cdots, x_N \rangle$ for a path p. Transformer layer takes a $\mathbf{X}$ as input and outputs the encoded edge representation vector $\mathbf{Z} = \langle z_1, z_2, z_3, \cdots, z_N \rangle$ that capture the dependencies of different edges. Especially, each Transformer layer contains multi-head attention and position-wise feed-forward networks [65].

Multi-Head Attention. Multi-head attention aims to integrate multiple attention layers to linearly project the queries, keys and values into M subspaces with different, learned linear projections to d_k, d_k and d_v dimensions, respectively [65]. Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions [65]. Then, we define it as:

$$\begin{aligned} \mathbf{Z} = \mathbf{MultiHead}(\mathbf{X}) &= \text{Concat}\left(\text{ head }_1, \ldots, \text{ head }_M\right) \cdot \mathbf{W}^O \,, \\ \text{head }_i(\cdot) &= \text{softmax}\left(\left(\mathbf{X}\mathbf{W}_i^Q\right)\left(\mathbf{X}\mathbf{W}_i^K\right)^T / \sqrt{d_k}\right)\left(\mathbf{X}\mathbf{W}_i^V\right) \,, \end{aligned} \tag{5.1}$$

where $\text{Concat}(\cdot, \cdot)$ denotes concatenation. $\mathbf{W}_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $\mathbf{W}_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $\mathbf{W}_i^V \in \mathbb{R}^{d_{model} \times d_v}$, $\mathbf{W}^O \in \mathbb{R}^{M d_v \times d_{model}}$ represent parameter matrices of projection of scaled dot-product attention. M is heads number. d_{model} is final feature dimension.

Position-wise Feed-Forward Networks. Except the attention sub-layers, each of the layers in Transformer (encoder/decoder) contains a fully connected feed-forward network (FFN), which is employed to each position individually and identically [65]. This FFN contains two linear transformations with ReLU activation in between [65]. Specifically, we have

$$\text{FFN}(\mathbf{Z}) = \max\left(0, \mathbf{Z}\mathbf{W}_\mathbf{1}^\mathbf{FFN} + \mathbf{b}_\mathbf{1}^\mathbf{FFN}\right)\mathbf{W}_\mathbf{2}^\mathbf{FFN} + \mathbf{b}_\mathbf{2}^\mathbf{FFN} \,, \tag{5.2}$$

where $\mathbf{W}_\mathbf{1}^\mathbf{FFN}$, $\mathbf{W}_\mathbf{2}^\mathbf{FFN}$, $\mathbf{b}_\mathbf{1}^\mathbf{FFN}$, and $\mathbf{b}_\mathbf{2}^\mathbf{FFN}$ are learnable parameters of feed-forward network.

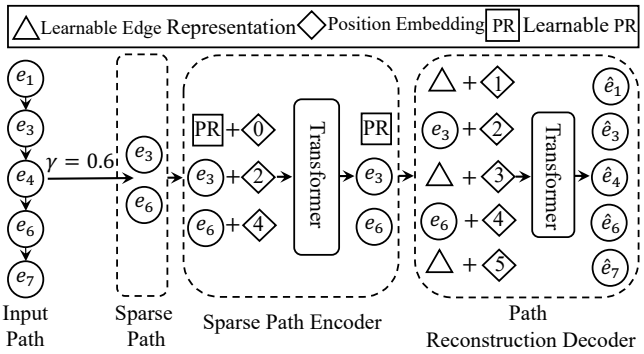


Figure 5.2: Sparse Auto-encoder. We remove a subset of edges from a path based on a reduction ratio γ to obtain a sparse path. We introduce a learnable path representation in front of the sparse path. And then, we fed the resulting sparse path vectors with position embeddings to a Transformer based encoder. We then introduce a learnable edge representation, denoted as a triangle, to represent the removed edges. The encoded edges in the sparse path and the removed edge representations with position embeddings are processed by a decoder that reconstructs the edges in the original path [65].

2.2 Problem Definition

Given a set of paths $\mathbb{P} = \{p_i\}_{i=1}^{|\mathbb{P}|}$ in a road network graph G, lightweight and scalable path representation learning aims at learning a function $SEPRL_\theta(\cdot)$ which can generate a task-unspecific path representation for each path $p_i \in \mathbb{P}$ by considering model scalability and resource-constrained environment applications, which can be defined as follows.

$$PR = SEPRL_\theta(p_i) : \mathbb{R}^{N \times d_k} \rightarrow \mathbb{R}^d , \tag{5.3}$$

where PR denotes the learned path representation. θ is the parameter of sparse path encoder. N is path length, d_k and d are the feature dimensions for an edge, and a final path representation, respectively.

3 Sparse Path Encoder

Figure 5.2 gives an overview of sparse path encoder.

3.1 Sparsity Operation

Sparsity operation takes a full path as input and outputs a sparse path, w.r.t., reduction ratio γ, which aims at reducing path length from N to N'. N' is much less than N. In particular, we randomly remove a subset of edges in a

path based on the corresponding reduction ratio γ (e.g., $\gamma = 0.6$ in Figure 5.2), enabling the scalability of the path. We also define sparsity operation as:

$$p' = f(p, \gamma) = \{e_j\}_{j \in \Omega}, \tag{5.4}$$

where p denotes an input path. p' represents a sparse path. For example, as shown in Figure 5.2, if we have a path $p = \langle e_1, e_3, e_4, e_6, e_7 \rangle$, then after conducting sparsity operation, we have a sparse path $p' = \{e_3, e_6\}$ and $p'.\Omega = [2, 4]$, where a subset of edge e_1, e_4, e_7 has removed from path p when $\gamma = 0.6$. This operation enables the path reduction, e.g., from 5 to 2 in this scenario.

3.2 Learnable Path Representation

We select Transformer as our path encoder since it supports the parallel operation using the self-attention principle. To achieve the path representations directly, we add a super extra learnable path representation vector PR in front of each sparse path. In particular, we attach PR at the position for all sparse paths, which makes it able to capture global information during the training procedure. We rewritten p' as:

$$p' = \{PR\} + \{e_j\}_{j \in \Omega} = \{e_k\}_{k \in \Omega'}, \tag{5.5}$$

where $e_0 = PR$ represents a virtual edge and $\Omega' = [0, \Omega]$.

To keep the sequential information for each path, we add the learnable position embedding into the sparse path feature vectors according to corresponding order information in Ω'. Thus, we have:

$$\mathbf{X} = \text{Concat}\{x_k\}_{k \in \Omega'}, \text{ where } x_k = e_k + pos_k, \tag{5.6}$$

where pos_k denotes the learnable position embedding. $\mathbf{X}$ denotes final sparse path edge representation through concatenation.

3.3 Transformer Path Encoder

The transformer contains multiple Transformer layers, and each of them consists of two sub-layer: multi-head attention and position-wise feed-forward network. Thus, we define our Transformer Path Encoder as:

$$\begin{aligned} Z &= \text{LayerNorm}(\mathbf{X} + \text{MultiHead}(\mathbf{X})), \\ PR &= \text{LayerNorm}(Z + \text{FFN}(Z)), \end{aligned} \tag{5.7}$$

where LayerNorm is layer normalization and PR is learned path representation. MultiHead and FFN are multi-head attention and position-wise feed forward network operation, respectively.

3.4 Path Reconstruction Decoder

To avoid the path information missing, the lightweight path encoder is introduced to reconstruct the removed edges in a path. Take Figure 5.2 as an example; we first compliment the encoded spare path with shared, learnable vectors that denotes the presence of removed edges based on the original index of each edge in a path. Then, we also add position embedding into the complemented sparse path representation. Next, the path decoder takes the complemented path representation as input and predicts the removed edges. We use mean square error (MSE) as our reconstruction loss function, which is given as:

$$\mathcal{L}_{rec} = \frac{1}{N}\sum_{i=1}^{N}(e_i - \hat{e}_i)^2 \ , \tag{5.8}$$

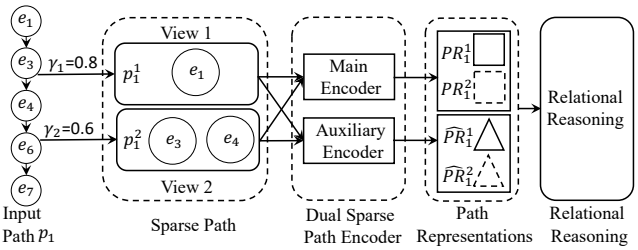
where e_i and $\hat{e}_i$ represent the initial and predicted removed edge representation, respectively. N is the number of edges in an input path.

4 Relational Reasoning Path Representation Learning

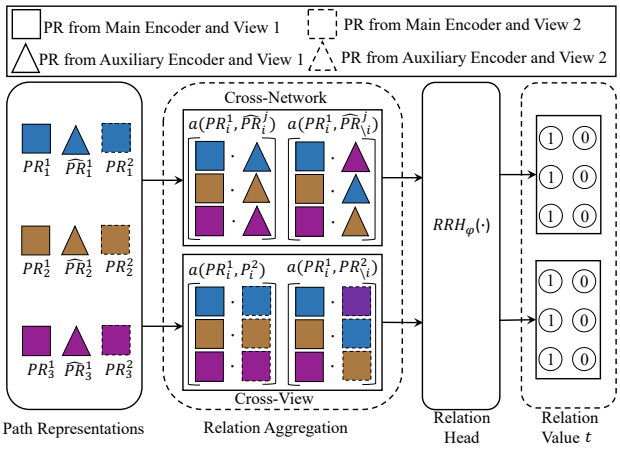
Figure 5.3 illustrates an overview of relational reasoning for path representation learning. Here, we aim to train a relation to head $RRH_{\varphi}(\cdot)$ to distinguish how path representations to themselves (same class) and other paths (different class). More specifically, this relational reasoning framework contains path representation construction and relational reasoning, whereas relational reasoning also includes cross-network relational reasoning and cross-view relational reasoning.

4.1 Dual Sparse Path Encoder

To enable effective relational reasoning training, we first introduce a dual sparse path encoder (SPE), which is used to construct different path representations according to different path views. Take Figure 5.3a as example, given a path p_1, we first achieve two sparse paths according to different reduction ratios, i.e., γ_1 and γ_2. We call them different path views, i.e., path view 1 p_1^1 and path view 2 p_1^2. Then, dual sparse path encoder, consisting of a main and an auxiliary encoders, takes p_1^1 and p_1^2 as input and returns four different path representations, where solid and dotted $\square$ represent the path representations output by main encoder in terms of p_1^1 and p_1^2, respectively, i.e., PR_1^1 and PR_1^2. While solid and dotted $\triangle$ denote the path representations returned by the auxiliary encoder with respect to both path views, respectively, i.e., $\hat{PR}_1^1$ and $\hat{PR}_1^2$. Finally, we define it as:



(a) Path Representation Construction Using Dual Sparse Path Encoders



(b) Relational Reasoning

Figure 5.3: Illustration of RR Training: (a) Given an input path p_1, we construct two path views (i.e., p_1^1 and p_1^2) through two reduction ratios γ_1 and γ_2, based on which a main encoder and an auxiliary encoder are employed to generate path representations for each view (i.e., PR_1^1, PR_1^2, $\widehat{PR}_1^1$, and $\widehat{PR}_1^2$). (b) After getting corresponding path representations for paths in a minibatch, a relational reasoning path representation learning scheme, which utilizes both cross-network and cross-view relational reasoning modules, is deployed. In particular, for both modules, an aggregation function a joins positives (representations of the same paths, e.g., $a(PR_1^1, \widehat{PR}_1^1)$, $a(PR_1^1, PR_1^2)$) and negatives (randomly paired representations, e.g., $a(PR_1^1, \widehat{PR}_3^1)$, $a(PR_1^1, PR_3^2)$) through a commutative operator. Then relation head module $RRH_\varphi(\cdot)$ estimates the relation score y, which must be 1 for positive and 0 for negatives. Both cross-network and cross-view objectives are optimized minimizing the binary cross-entropy (BCE) between prediction and target relation value t. In this example, $i \in [1, 2, 3]$ denotes the number of paths in the minibatch and $j \in [1, 2]$ represents the number of views [65].

$$PR_i^j = SPE_\theta(p_i^j, \gamma) \ , \ \hat{PR}_i^j = SPE_{\hat{\theta}}(p_i^j, \gamma) \ , \tag{5.9}$$

where PR_i^j and $\hat{PR}_i^j$ denote path representations returned by main encoder and auxiliary encoder, respectively. p_i is the i-th path in a path set. $j \in [1, 2]$ represents path views. θ and $\hat{\theta}$ represent parameters for the main encoder and auxiliary encoder.

4.2 Relation Reasoning

Cross-Network Relational Reasoning

As shown in Figure 5.3a, given a set of path $\{p_1, p_2, \cdots, p_K\}$, we are able to achieve a set of path representations $\{PR_1^1, PR_2^1, \cdots, PR_K^1\}$ from main encoder and $\{\hat{PR}_1^1, \hat{PR}_2^1, \cdots, \hat{PR}_K^1\}$ or $\{\hat{PR}_1^2, \hat{PR}_2^2, \cdots, \hat{PR}_K^2\}$ from auxiliary encoder, w.r.t., different path views. Then, we join the positive relation pairs $\langle PR_i^1, PR_i^2 \rangle$ (e.g., $\langle PR_1^1, \hat{PR}_1^1 \rangle$ in Figure 5.3) and negative relation pairs $\langle PR_i^1, PR_{\setminus i}^2 \rangle$(e.g., $\langle PR_1^1, \hat{PR}_2^1 \rangle$ in Figure 5.3) through aggregation function. Next, the relational head $RRH_\varphi(\cdot)$ takes representation relation pairs of cross-network as input and returns a relation score y. Finally, we formulate cross-network relational reasoning as a binary classification, where the binary cross-entropy loss is used to train a sparse path encoder, which is defined as follows.

$$\begin{aligned} \mathcal{L}_{cn} = \underset{\boldsymbol{\theta}, \varphi}{\operatorname{argmin}} \sum_{i=1}^{K} \sum_{j=1}^{2} \mathcal{L} \left(RRH_\varphi \left(a \left(PR_i^j, \hat{PR}_i^j \right) \right), t = 1 \right) \\ + \mathcal{L} \left(RRH_\varphi \left(a \left(PR_i^j, \hat{PR}_{\setminus i}^j \right) \right), t = 0 \right) \ , \end{aligned} \tag{5.10}$$

where K is the the number of path samples in the minibatchand $K = 3$ in Figure 5.3. $a(\cdot, \cdot)$is defined as an aggregation function. $\mathcal{L}$ represents a loss function between relation score and a target relation value. t is a target relation values.

To enable the cross-network framework training, we employ Siamese architecture for our dual sparse path encoder, where we update the parameters of the auxiliary encoder by leveraging the momentum updating principle based on the parameters of the main encoder. Then, we have:

$$\hat{\theta}^t = m \times \hat{\theta}^{(t-1)} + (1 - m) \times \theta^t, \tag{5.11}$$

where m represents momentum parameter. θ and $\hat{\theta}$ denote the parameters of main encoder and auxiliary encoder.

Cross-View Relational Reasoning

To further enhance the sparse path encoder training, we conduct the relational reasoning between different views within the main encoder, termed cross-view relational reasoning.

Similar to cross-network, given a set of paths $\{p_1, p_2, \cdots, p_K\}$. We first obtain two set of path representations regarding two path views using main encoder, i.e., $\{PR_1^1, PR_2^1, \cdots, PR_k^1\}$ and $\{PR_1^2, PR_2^2, \cdots, PR_K^2\}$. Then, we also join the positive relation pairs $\langle PR_i^1, PR_i^2 \rangle$ (e.g., $\langle PR_1^1, PR_1^2 \rangle$ in Figure 5.3) and negative relation pairs $\langle PR_i^1, PR_{\setminus i}^2 \rangle$ (e.g., $\langle PR_1^1, PR_3^2 \rangle$ in Figure 5.3) through aggregation function. Next, we learn relational head $RRH_\varphi(\cdot)$ to obtain the corresponding relation score y for the cross-view relational reasoning. Last, we define the objective function as:

$$\begin{aligned} \mathcal{L}_{cv} = \underset{\boldsymbol{\theta},\varphi}{\operatorname{argmin}} \sum_{i=1}^{K} \mathcal{L}\left(RRH_\varphi\left(a\left(PR_i^1, PR_i^2\right)\right), t=1\right) \\ + \mathcal{L}\left(RRH_\varphi\left(a\left(PR_i^1, PR_{\setminus i}^2\right)\right), t=0\right) \ , \end{aligned} \tag{5.12}$$

where K denotes the the number of path samples in the minibatch.

Objective for *RR*

To efficiently train our dual path encoder, we jointly minimize the cross-network and cross-view relational reasoning loss. The objective function for RR is defined as:

$$\min_{\theta,\varphi} \mathcal{L}_{RR} = \mathcal{L}_{cn} + \mathcal{L}_{cv} \tag{5.13}$$

4.3 LightPath Training

Finally, we jointly minimize the reconstruction and RR loss, which can be described as:

$$\mathcal{L} = \mathcal{L}_{rec} + \mathcal{L}_{RR} \tag{5.14}$$

5 Global Local Knowledge Distillation (GLKD)

We further introduce global local knowledge distillation (GLKD) to enable *LightPath* can be deployed in a resource-limited environment by reducing the model size, as shown in Figure 5.4.

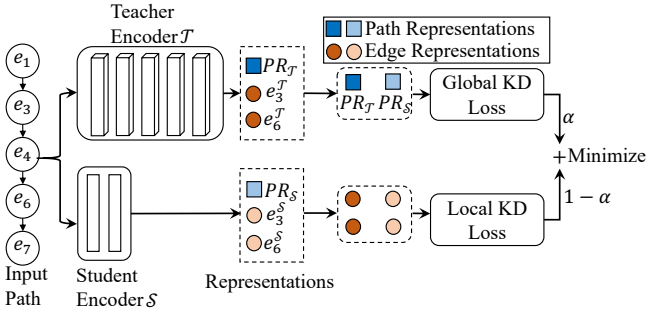


Figure 5.4: Illustration of GLKD. Given an input path, we formulate our GLKD as a weighted sum of global path representation knowledge distillation (GPRKD) loss and local edge representation knowledge distillation (LERKD) loss [65].

5.1 Global-path Representation Distillation

Given an input path $p_i = \langle e_1, e_2, e_3, \cdots, e_n \rangle$, where n represents the number of edges in an input path. We define $PR^{\mathcal{T}}(p_i)$ and $PR^{\mathcal{S}}(p_i)$ represent the path representations achieved from teacher encoder $\mathcal{T}_\theta$ and student encoder $\mathcal{S}_\theta$. The global path representation distillation aims to push the smaller student encoder to mimic the global characteristic of the large cuboid teacher encoder. This problem is formalized as minimizing the latent space distance between the large cuboid teacher encoder and the rectangle student encoder. Thus, we have:

$$\min_{\theta} \mathcal{L}_{global}\left(\mathbf{PR}^{\mathcal{T}}(p_i), \mathbf{PR}^{\mathcal{S}}(p_i)\right) = \left\| sp(\mathbf{PR}^{\mathcal{T}}(p_i)/t) - sp(\mathbf{PR}^{\mathcal{S}}(p_i)/t) \right\|^2, \quad (5.15)$$

where $sp(\cdot)$ denotes exponential function. t is the temperature.

5.2 Local-edge Correlation Distillation

Local-edge correlation distillation aims to keep the local smooth of the edge correlations in a path.

In particular, a rectangle student encoder tries to mimic the edge correlations in a path captured by a large cuboid teacher encoder.

In specific, given a path $p = \langle e_1, e_2, e_3, \cdots e_N \rangle$, where N represents the number of edges in a path. Through employing an L-layers Transformer encoder (i.e., teacher encoder $\mathcal{T}_\theta$) and L'-layers Transformer encoder (i.e., student encoder $\mathcal{S}_\theta$) upon sparse path p', where $L \ll L'$, the edge representation that captures spatial dependencies are derived as follows.

$$F^{\mathcal{T}}(e_i)_{i=1}^{N'} = \mathcal{T}_\theta(p)\ ,\ F^{\mathcal{S}}(e_i)_{i=1}^{N'} = \mathcal{S}_\theta(p)\ , \quad (5.16)$$

where $F^{\mathcal{T}}(e_i)_{i=1}^{N'}$ and $F^{\mathcal{S}}(e_i)_{i=1}^{N'}$ represent the edge representation with respect to the teacher and student encoder, respectively.

In this phase, we define the objective function as:

$$\min_{\theta} \mathcal{L}_{local}\left(\mathbf{F}^{\mathcal{T}}(e_i), \mathbf{F}^{\mathcal{S}}(e_i)\right) = \frac{1}{n}\sum_{i=1}^{n}\left\|sp(\mathbf{F}^{\mathcal{T}}(e_i)/t) - sp(\mathbf{F}^{\mathcal{S}}(e_i)/t)\right\|^2, \tag{5.17}$$

where $sp(\cdot)$ is exponential function. t denotes the temperature.

5.3 Objective for *GLKD*

To enable the GLKD, the overall objective function for GLKD is defined in Eq. 5.18.

$$\min_{\theta} \mathcal{L}_{GLKD} = \alpha * \mathcal{L}_{global} + (1-\alpha) * \mathcal{L}_{local}\ , \tag{5.18}$$

where α is balancing factor.

6 Experiments

6.1 Experimental Setup

Data sets

We consider two real-world road network data sets and one synthetic data set. We collect the road network of Aalborg, Denmark, from OpenStreetMap, which contains 10,017 nodes and 11,597 edges [65]. After map matching, we have 39,160 paths with lengths of 50. In addition, we collect the road network of Chengdu, China, from OpenStreetMap, which includes 6,632 nodes and 17,038 edges [65]. After map matching, we obtain 50,000 paths with lengths of 50. Due to the lack of large amounts of long paths in the real-world datasets, we also generate one synthetic data set that includes paths with a length of 100, 150, and 200, which is used to verify the efficiency and scalability of the *LightPath*.

Downstream Task

We consider two downstream tasks: path travel time estimation and path ranking. We evaluate the accuracy of travel time estimation based on Mean Absolute Error(**MAE**), Mean Absolute Relative Error(**MARE**), and Mean Absolute Percentage Error(**MAPE**) [65]. In contrast, we evaluate the performance of path ranking based on **MAE**, the Kendall rank correlation coefficient (τ), and Spearman's rank correlation coefficient (ρ) [65].

Models for Downstream Task

We select the ensemble model *Gradient Boosting Regressor (GBR)* as a regression model to estimate the travel time and ranking score based on the learn path representation through unsupervised learning methods.

Baselines

We compare *LightPath* with 9 baselines. We first consider **Node2vec** [12] since it is famous graph embedding methods. Then, we select MoCo [19] due to it being the representative work of self-supervised learning. Next, we choose three trajectory representation learning methods as our baselines, i.e., **Toast** [3], **t2vec** [34], and **NeuTraj** [67]. In addition, we also consider unsupervised path representation learning methods *PIM* [62]. Finally, we further consider three supervised methods: **HMTRL** [35], **PathRank** [64], and *LightPath-Sup*.

6.2 Experiment Results

Overall Performance

Table 5.1 reports the overall performance comparsion between our *LightPath* and all baselines on two downstream tasks regarding two real-world data sets. In particular, we use 30K unlabeled paths on Aalborg and Chengdu, respectively, but we only use 12K labeled paths to train GBR and supervised baseline methods. Overall, *LightPath* outperforms all the baselines on these two tasks for both data sets, which demonstrates the advance of our model.

Model Scalability

In the section, we conduct the experiments to study the model scalability regarding reduction ratio and path length based on the synthetic dataset. Table 5.2 reports the results for both *LightPath* and its teacher model, with varying $\gamma = 0, 0.1, 0.3, 0.5, 0.7, 0.9$. $\gamma = 0$ represents we do not consider sparsity operation for the input path, i.e., using a classic Transformer based encoder. We can observe that the GFLOPs and gMem. (GiB) decrease with the increase in the reduction ratio. It is because the higher value of γ is, the more edges we can remove. Second, *LightPath* has significantly reduced model complexity, w.r.t., GFLOPs and gMem.. For example, we can reduce the training GFLOPs by 2.54$\times$ for the *LightPath* by increasing the reduction ratio γ from 0 to 0.9 in terms of path length 200. Third, the parameters (Para. (Millions)) of teacher model is at least 3.5$\times$ of *LightPath*, which indicates the effectiveness of our proposed framework. Overall, *LightPath* shows potential of scalability to support path representation learning for long paths.

Table 5.1: Overall Accuracy on Travel Time Estimation and Ranking Score Estimation [65].

Method	Aalborg					Chengdu				
	Travel Time Estimation			Path Ranking		Travel Time Estimation			Path Ranking	
	MAE	MARE	MAPE	MAE	τ	ρ	MAE	MARE	MAPE	MAE	τ	ρ
Node2vec	154.07	0.20	25.22	0.24	0.59	0.64	267.28	0.23	26.30	0.15	0.74	0.77
MoCo	146.29	0.19	21.60	0.25	0.53	0.57	237.14	0.20	23.13	0.15	0.77	0.81
Toast	137.27	0.17	20.43	0.24	0.59	0.63	240.57	0.21	23.50	0.11	0.65	0.68
t2vec	147.24	0.19	22.13	0.25	0.52	0.56	242.96	0.21	23.65	0.14	0.77	0.82
NeuTraj	117.06	0.15	18.09	0.25	0.60	0.64	232.96	0.20	22.73	0.12	0.79	0.83
PIM	102.09	0.14	14.92	0.20	0.63	0.67	223.34	0.19	21.69	0.12	0.80	0.84
HMTRL	101.81	0.13	14.51	0.17	0.68	0.72	218.94	0.19	21.22	0.09	0.83	0.84
PathRank	115.37	0.15	16.41	0.21	0.64	0.68	229.85	0.20	22.53	0.11	0.81	0.82
LightPath-Sup	105.51	0.15	16.35	0.14	0.68	0.72	218.67	0.19	21.36	0.13	0.76	0.79
LightPath	**85.76**	**0.11**	**12.12**	**0.13**	**0.73**	**0.77**	**212.61**	**0.18**	**20.75**	**0.07**	**0.87**	**0.88**

Table 5.2: Model Scalability vs. Reduction Ratio (γ) and Path Length (N) [65].

N	***LightPath***												Para.
	$\gamma = 0$		$\gamma = 0.1$		$\gamma = 0.3$		$\gamma = 0.5$		$\gamma = 0.7$		$\gamma = 0.9$		
	GFLOPs	gMem.	GFLOPs	gMem.	GFLOPs	gMem.	GFLOPs	gMem.	GFLOPs	gMem.	GFLOPs	gMem.	
50	8.01	1.47	7.48	1.39	6.44	1.37	5.39	1.36	4.34	1.34	3.18	1.33	1.570
100	15.77	1.60	14.72	1.50	12.62	1.48	10.52	1.46	8.43	1.44	6.23	1.43	1.570
150	23.53	1.72	21.95	1.64	18.81	1.60	15.66	1.58	12.52	1.55	9.27	1.52	1.570
200	31.29	1.90	29.19	1.81	24.99	1.77	20.80	1.73	16.61	1.68	12.31	1.65	1.570
	LightPath w/o KD												
	$\gamma = 0$		$\gamma = 0.1$		$\gamma = 0.3$		$\gamma = 0.5$		$\gamma = 0.7$		$\gamma = 0.9$		Para.
	GFLOPs	gMem.	GFLOPs	gMem.	GFLOPs	gMem.	GFLOPs	gMem.	GFLOPs	gMem.	GFLOPs	gMem.	
50	33.68	1.78	30.64	1.70	22.55	1.61	18.47	1.53	12.39	1.47	5.70	1.41	5.525
100	66.60	2.53	60.52	2.37	48.36	2.11	36.19	1.86	24.03	1.72	11.26	1.58	5.525
150	99.53	3.44	90.41	3.23	72.16	2.76	53.91	2.35	35.65	2.03	16.82	1.82	5.525
200	132.54	4.74	120.29	4.30	95.96	3.53	71.64	2.94	47.31	2.43	22.37	2.10	5.525

Chapter 6

Conclusion and Future Work

1 Conclusion

This thesis focuses on path representation learning in road networks. It summarizes five research works. One is four pages report, while the other four studies are four novel advanced path representation learning frameworks. The contributions of each work are summarized as follows.

- **Paper A** [66] and **Paper B** [64] study the supervised path representation from the path ranking perspective. In particular, in Paper B, we first propose a training data enrichment strategy, which includes diversified multi-cost shortest paths, to enhance learning ability. Then, we propose an end-to-end multi-task framework *PathRank* that captures both road network topology and spatial proprieties to enable efficient path feature learning. Extensive experiments are conducted, and the corresponding results demonstrate the effectiveness of the proposed framework.

- **Paper C** [62] studies the unsupervised path representation learning framework Path InfoMax *PIM*, which learns task-unspecific path representations and works well on different downstream tasks. In particular, we first propose a curriculum negative sampling strategy to generate a small number of negative paths by following the curriculum learning principles. Next, *PIM* conduct mutual information maximization to learn task-unspecific path representations from a global and a local view. As for the global view, *PIM* aims to distinguish the input path representations from the representation of negative paths. In a local view, the goal of *PIM* is to distinguish the input path representation from the representation of nodes that only appear in negative paths. This enables the learned representations to capture both global and local

information. The extensive experiments demonstrate the efficiency of the learned task-unspecific path representations.

- **Paper D** [63] proposes weakly-supervised contrastive curriculum learning (WSCCL) framework to learn task-unspecific temporal path representations (TPR) by leveraging the weak label information. More specifically, we first employ a temporal path encoder that captures both spatial and temporal information to learn TPRs. We then introduce weak labels to enable the encoder training. These weak labels (e.g., peak hour vs. off-peak hour) are easy and economical to achieve but relevant to various smart-city applications. Next, we generate positive and negative temporal path samples with respect to weak labels, which benefits temporal path encoder training using weakly-supervised contrastive learning by pulling together positive paths' representations and pushing apart negative paths' representations. Subsequently, we integrate curriculum learning with weakly-supervised contrastive learning to further enhance the weakly-supervised contrastive learning. The extensive experiments indicate that our *WSCCL* performs best because it leverages the temporal information from weak labels.

- **Paper E** [65] proposes a lightweight and scalable path representation learning framework *LightPath*. *LightPath* aims at reducing resource consumption and achieving scalability without affecting accuracy, resulting in broader applicability. More specifically, we first study a sparse autoencoder that offers the framework with good scalability in terms of path length. Then, we propose a relational reasoning path representation learning framework, including cross-network relational reasoning and cross-view relational reasoning, to enable faster and more robust sparse path encoder training. Next, we investigate the global-local knowledge distillation to reduce the size of the sparse path encoder and further improve the downstream tasks' performance. Finally, the results of the experiments indicate the efficiency and scalability of the *LightPath*.

2 Future Work

As for future work, several meaningful directions could be considered.

- **Stochastic Path Representation Learning** Studying stochastic path representation learning instead of deterministic ones from the distribution perspective. This enables the learned path representations to be robust to the external uncertainty (e.g., traffic accidents), thus improving the performance for different downstream tasks.

- **AutoPRL** Automated machine learning (AutoML) has driven related innovation in different research communities. Such technology can also be employed to construct a unified task-unspecific path representation learning framework. For example, we can search between different self-supervised learning strategies to fit the different environment settings, e.g., with /without negative paths.

References

[1] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *ICML*, 2009, pp. 41–48.

[2] T. Chen and S. Lu, "Accurate and efficient traffic sign detection using discriminative adaboost and support vector regression," *IEEE Trans. Vehicular Technology*, vol. 65, no. 6, pp. 4006–4015, 2016.

[3] Y. Chen, X. Li, G. Cong, Z. Bao, C. Long, Y. Liu, A. K. Chandran, and R. Ellison, "Robust road network representation learning: When traffic patterns meet traveling semantics," in *CIKM*, 2021, pp. 211–220.

[4] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," in *EMNLP*, 2014, pp. 103–111.

[5] R.-G. Cirstea, T. Kieu, C. Guo, B. Yang, and S. J. Pan, "Enhancenet: Plugin neural networks for enhancing correlated time series forecasting." in *ICDE*, 2021, pp. 1739–1750.

[6] J. Dai, B. Yang, C. Guo, and Z. Ding, "Personalized route recommendation using big trajectory data," in *ICDE*, 2015, pp. 543–554.

[7] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *NIPS*, 2016, pp. 3837–3845.

[8] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *NAACL-HLT*, 2019, pp. 4171–4186.

[9] E. Erkut and V. Verter, "Modeling of transport risk for hazardous materials," *Operations research*, vol. 46, no. 5, pp. 625–642, 1998.

[10] X. Fang, J. Huang, F. Wang, L. Liu, Y. Sun, and H. Wang, "Ssml: self-supervised meta-learner for en route travel time estimation at baidu maps," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021, pp. 2840–2848.

[11] Z. Fang, L. Pan, L. Chen, Y. Du, and Y. Gao, "MDTP: A multi-source deep traffic prediction framework over spatio-temporal trajectory data," *Proc. VLDB Endow.*, vol. 14, no. 8, pp. 1289–1297, 2021.

[12] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *KDD*, 2016, pp. 855–864.

[13] ——, "node2vec: Scalable feature learning for networks," in *SIGKDD*, 2016, pp. 855–864.

[14] C. Guo, B. Yang, J. Hu, and C. S. Jensen, "Learning to route with sparse trajectory sets," in *ICDE*, 2018, pp. 1073–1084.

[15] C. Guo, B. Yang, J. Hu, C. S. Jensen, and L. Chen, "Context-aware, preference-based vehicle routing," *VLDB J.*, vol. 29, no. 5, pp. 1149–1170, 2020.

[16] D. Gupta and A. Garg, "Sustainable development and carbon neutrality: Integrated assessment of transport transitions in india," *Transportation Research Part D: Transport and Environment*, vol. 85, p. 102474, 2020.

[17] L. Han, B. Du, J. Lin, L. Sun, X. Li, and Y. Peng, "Multi-semantic path representation learning for travel time estimation," *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[18] X. Han, R. Cheng, C. Ma, and T. Grubenmann, "Deeptea: Effective and efficient online time-dependent trajectory outlier detection," *Proc. VLDB Endow.*, vol. 15, no. 7, pp. 1493–1505, 2022.

[19] K. He, H. Fan, Y. Wu, S. Xie, and R. B. Girshick, "Momentum contrast for unsupervised visual representation learning," in *CVPR*, 2020, pp. 9726–9735.

[20] R. D. Hjelm, A. Fedorov, S. Lavoie-Marchildon, K. Grewal, P. Bachman, A. Trischler, and Y. Bengio, "Learning deep representations by mutual information estimation and maximization," in *ICLR*, 2019.

[21] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[22] J. Hu, C. Guo, B. Yang, and C. S. Jensen, "Stochastic weight completion for road networks using graph convolutional networks," in *ICDE*, 2019, pp. 1274–1285.

[23] ——, "Stochastic weight completion for road networks using graph convolutional networks," in *ICDE*, 2019, pp. 1274–1285.

[24] J. Hu, B. Yang, C. Guo, and C. S. Jensen, "Risk-aware path selection with time-varying, uncertain travel costs: a time series approach," *VLDB J.*, vol. 27, no. 2, pp. 179–200, 2018.

[25] J. Hu, B. Yang, C. Guo, C. S. Jensen, and H. Xiong, "Stochastic origin-destination matrix forecasting using dual-stage graph convolutional, recurrent neural networks," in *ICDE*, 2020, pp. 1417–1428.

[26] S. Huang, Y. Wang, T. Zhao, and G. Li, "A learning-based method for computing shortest path distances on road networks," in *ICDE*, 2021, pp. 360–371.

[27] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *NIPS*, 2017, pp. 3146–3154.

[28] M. G. Kendall, "A new measure of rank correlation," *Biometrika*, vol. 30, no. 1/2, pp. 81–93, 1938.

[29] P. Khosla, P. Teterwak, C. Wang, A. Sarna, Y. Tian, P. Isola, A. Maschinot, C. Liu, and D. Krishnan, "Supervised contrastive learning," in *NeurIPS*, 2020, pp. 18 661–18 673.

[30] T. Kieu, B. Yang, C. Guo, and C. S. Jensen, "Distinguishing trajectories from different drivers using incompletely labeled trajectories," in *CIKM*, 2018, pp. 863–872.

[31] S. Y. Kim and A. Upneja, "Predicting restaurant financial distress using decision tree and adaboosted decision tree models," *Economic Modelling*, vol. 36, pp. 354–362, 2014.

[32] J. Li, Z. Han, H. Cheng, J. Su, P. Wang, J. Zhang, and L. Pan, "Predicting path failure in time-evolving graphs," in *KDD*, 2019, pp. 1279–1289.

[33] X. Li, G. Cong, A. Sun, and Y. Cheng, "Learning travel time distributions with deep generative model," in *WWW*. ACM, 2019, pp. 1017–1027.

[34] X. Li, K. Zhao, G. Cong, C. S. Jensen, and W. Wei, "Deep representation learning for trajectory similarity computation," in *ICDE*, 2018, pp. 617–628.

[35] H. Liu, J. Han, Y. Fu, J. Zhou, X. Lu, and H. Xiong, "Multi-modal transportation recommendation with unified route representation learning," *Proc. VLDB Endow.*, vol. 14, no. 3, pp. 342–350, 2020.

[36] H. Liu, C. Jin, B. Yang, and A. Zhou, "Finding top-k optimal sequenced routes," in *ICDE*, 2018, pp. 569–580.

[37] ——, "Finding top-k shortest paths with diversity," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 3, pp. 488–502, 2018.

[38] P. Newson and J. Krumm, "Hidden markov map matching through noise and sparseness," in *SIGSPATIAL*, 2009, pp. 336–343.

[39] S. A. Pedersen, B. Yang, and C. S. Jensen, "Anytime stochastic routing with hybrid learning," *PVLDB*, vol. 13, no. 9, pp. 1555–1567, 2020.

[40] ——, "Fast stochastic routing under time-varying uncertainty," *VLDB J.*, vol. 29, no. 4, pp. 819–839, 2020.

[41] Z. Peng, W. Huang, M. Luo, Q. Zheng, Y. Rong, T. Xu, and J. Huang, "Graph representation learning via graphical mutual information maximization," in *WWW*, 2020, pp. 259–270.

[42] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *SIGKDD*, 2014, pp. 701–710.

[43] Z. Shang, G. Li, and Z. Bao, "DITA: A distributed in-memory trajectory analytics system," in *SIGMOD*, 2018, pp. 1681–1684.

[44] ——, "DITA: distributed in-memory trajectory analytics," in *SIGMOD*, 2018, pp. 725–740.

[45] Y. Shi, J. Li, and Z. Li, "Gradient boosting with piece-wise linear regression trees," *arXiv preprint arXiv:1802.05640*, 2018.

[46] F. Sun, J. Hoffmann, V. Verma, and J. Tang, "Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization," in *ICLR*, 2020.

[47] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "LINE: large-scale information network embedding," in *WWW*, 2015, pp. 1067–1077.

[48] L. V. Tran, M. Mun, M. Lim, J. Yamato, N. Huh, and C. Shahabi, "Deeptrans: A deep learning system for public bus travel time estimation using traffic forecasting," *Proc. VLDB Endow.*, vol. 13, no. 12, pp. 2957–2960, 2020.

[49] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *NIPS*, 2017, pp. 5998–6008.

[50] P. Velickovic, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep graph infomax," in *ICLR*, 2019.

[51] S. Wang, Z. Bao, J. S. Culpepper, T. Sellis, and X. Qin, "Fast large-scale trajectory clustering," *Proc. VLDB Endow.*, vol. 13, no. 1, pp. 29–42, 2019.

[52] S. Wang, B. Ji, J. Zhao, W. Liu, and T. Xu, "Predicting ship fuel consumption based on lasso regression," *Transportation Research Part D: Transport and Environment*, vol. 65, pp. 817–824, 2018.

[53] Y. Wang, Q. Wang, H. Koehler, and Y. Lin, "Query-by-sketch: Scaling shortest path graph queries on very large networks," in *SIGMOD*, 2021, pp. 1946–1958.

[54] Y. Wang, Y. Zheng, and Y. Xue, "Travel time estimation of a path using sparse trajectories," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 25–34.

[55] Z. Wang, C. Long, G. Cong, and Y. Liu, "Efficient and effective similar subtrajectory search with deep reinforcement learning," *Proc. VLDB Endow.*, vol. 13, no. 11, pp. 2312–2325, 2020.

[56] V. J. Wei, R. C. Wong, and C. Long, "Architecture-intact oracle for fastest path and time queries on dynamic spatial networks," in *SIGMOD*, 2020, pp. 1841–1856.

[57] X. Wu, D. Zhang, C. Guo, C. He, B. Yang, and C. S. Jensen, "Autocts: Automated correlated time series forecasting," *Proc. VLDB Endow.*, vol. 15, no. 4, pp. 971–983, 2021.

[58] Z. Wu, Y. Xiong, S. X. Yu, and D. Lin, "Unsupervised feature learning via non-parametric instance discrimination," in *CVPR*, 2018, pp. 3733–3742.

[59] B. Yang, J. Dai, C. Guo, C. S. Jensen, and J. Hu, "PACE: a path-centric paradigm for stochastic path finding," *VLDB J.*, vol. 27, no. 2, pp. 153–178, 2018.

[60] B. Yang, C. Guo, C. S. Jensen, M. Kaul, and S. Shang, "Stochastic skyline route planning under time-varying uncertainty," in *ICDE*, 2014, pp. 136–147.

[61] B. Yang, C. Guo, Y. Ma, and C. S. Jensen, "Toward personalized, context-aware routing," *VLDB Journal*, vol. 24, no. 2, pp. 297–318, 2015.

[62] S. B. Yang, C. Guo, J. Hu, J. Tang, and B. Yang, "Unsupervised path representation learning with curriculum negative sampling," in *IJCAI*, 2021, pp. 3286–3292.

[63] S. B. Yang, C. Guo, J. Hu, B. Yang, J. Tang, and C. S. Jensen, "Weakly-supervised temporal path representation learning with contrastive curriculum learning," in *ICDE*, pp. 2873–2885.

[64] S. B. Yang, C. Guo, and B. Yang, "Context-aware path ranking in road networks," *IEEE TKDE*, 2020.

[65] S. B. Yang, J. Hu, C. Guo, B. Yang, and C. S. Jensen, "Lightpath: Lightweight and scalable path representation learning," in *In Submission*, 2022.

[66] S. B. Yang and B. Yang, "Learning to rank paths in spatial networks," in *ICDE*, 2020, pp. 2006–2009.

[67] D. Yao, G. Cong, C. Zhang, and J. Bi, "Computing trajectory similarity in linear time: A generic seed-guided neural metric learning approach," in *ICDE*, 2019, pp. 1358–1369.

[68] J. Y. Yen, "Finding the k shortest loopless paths in a network," *management Science*, vol. 17, no. 11, pp. 712–716, 1971.

[69] B. Yu, H. Yin, and Z. Zhu, "Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting," in *IJCAI*, 2018, pp. 3634–3640.

[70] H. Yuan, G. Li, Z. Bao, and L. Feng, "Effective travel time estimation: When historical trajectories over road networks matter," in *SIGMOD*, 2020, pp. 2135–2149.

[71] ——, "An effective joint prediction model for travel demands and traffic flows," in *ICDE*, 2021, pp. 348–359.

[72] J. H. Zar, "Significance testing of the spearman rank correlation coefficient," *Journal of the American Statistical Association*, vol. 67, no. 339, pp. 578–580, 1972.

[73] J. Zhang, Y. Zheng, J. Sun, and D. Qi, "Flow prediction in spatio-temporal networks based on multitask deep learning," *IEEE Trans. Knowl. Data Eng.*, vol. 32, no. 3, pp. 468–478, 2020.

[74] B. Zheng, Q. Hu, L. Ming, J. Hu, L. Chen, K. Zheng, and C. S. Jensen, "Soup: Spatial-temporal demand forecasting and competitive supply," *TKDE*, 2021.

Part II

Papers

Paper A

Learning to Rank Paths in Spatial Networks

Sean Bin Yang, Bin Yang

The paper has been published in the
IEEE 36th International Conference on Data Engineering (ICDE),
pp. 2006–2009, 2020.

Abstract

Modern navigation services often provide multiple paths connecting the same source and destination for users to select. Hence, ranking such paths becomes increasingly important, which directly affects service quality. We present PathRank*, a data-driven framework for ranking paths based on historical trajectories. If a trajectory used path P from source s to destination d,* PathRank *considers this as an evidence that P is preferred over all other paths from s to d. Thus, a path that is similar to P should have a larger ranking score than a path that is dissimilar to P. Based on this intuition,* PathRank *models path ranking as a regression problem that assigns each path a ranking score. We first propose an effective method to generate a compact set of diversified paths using trajectories as training data. Next, we propose an end-to-end deep learning framework to solve the regression problem. In particular, a spatial network embedding is proposed to embed each vertex to a feature vector by considering the road network topology. Since a path is represented by a sequence of vertices, which is now a sequence of feature vectors after embedding, recurrent neural network is applied to model the sequence. Empirical studies on a substantial trajectory data set offer insight into the designed properties of the proposed framework and indicating that it is effective and practical.*

1 Introduction

Vehicular transportation reflects the pulse of a city. It not only affects people's daily lives and also plays an essential role in many businesses as well as society as a whole [5]. A fundamental functionality in vehicular transportation is routing [14]. Given a source and a destination, classic routing algorithms, e.g., Dijkstra's algorithm, identify an optimal path connecting the source and the destination, where the optimal path is often the path with the least travel cost, e.g., the shortest path or the fastest path. However, a routing service quality study [2] shows that local drivers often choose paths that are neither shortest nor fastest, rendering classic routing algorithms often impractical in many real world routing scenarios.

To contend with this challenge, a wide variety of advanced routing algorithms, e.g., skyline routing [19] and k-shortest path routing [11, 21], are proposed to identify a set of optimal paths, where the optimality is defined based on, e.g., pareto optimality or top-k least costs, which provide drivers with multiple candidate paths. In addition, commercial navigation systems, such as Google Maps and TomTom, often follow a similar strategy by suggesting multiple candidate paths to drivers, although the criteria for selecting the candidate paths are often confidential.

Under this context, ranking the candidate paths is essential for ensuring high routing quality. Existing solutions often rely on simple heuristics, e.g.,

ranking paths w.r.t. their travel times. However, travel times may not always be the most important factor when drivers choose paths, as demonstrated in the routing quality study where drivers often do not choose the fastest paths [2]. In addition, existing solutions often provide the same ranking to all users but ignore distinct preferences which different drivers may have.

In this paper, we propose a data-driven ranking framework *PathRank*, which ranks candidate paths by taking into account the paths used by local drivers in their historical trajectories. More specifically, *PathRank* models ranking candidate paths as a "regression" problem—for each candidate path, *PathRank* estimates a ranking score for the candidate path.

We first prepare a training data set using historical trajectories. For each trajectory, we consider the path used by the trajectory, i.e., trajectory path, as the ground truth path and thus has the highest ranking score, say 1. We then propose an effective method to generate a diversified training path set for the trajectory path, where each training path is associated with a ranking score that equals to the similarity between the training path and the trajectory path. The intuition behind is that if a training path is similar to the ground truth trajectory path, it should also have a higher ranking score.

Next, we propose a deep learning framework to learn meaningful feature representations of paths, which enables effective ranking. The input for *PathRank* is a path and its label is a similarity score. In order to use deep learning to solve the similarity score regression problem, a prerequisite is to represent the input path into an appropriate feature space. To this end, we propose to use a vertex embedding network to transfer each vertex in the input path to a feature vector. Since a path is a sequence of vertices, after vertex embedding, the path becomes a sequence of feature vectors. Then, since RNNs are capable of capturing dependency for sequential data, we employ an RNN to model the sequence of feature vectors. The RNN finally outputs an estimated similarity score, which is compared against the ground truth similarity.

To the best of our knowledge, this is the first data-driven, end-to-end solution for ranking paths in spatial networks.

2 Related Work

2.1 Learning to rank

Learning to rank plays an important role in ranking in the context of information retrieval (IR), where the primary goal is to learn how to rank documents or web pages w.r.t. queries, which are all represented as feature vectors. Learning to rank methods in IR can be categorized into point-wise [3],

pair-wise [17], and list-wise [1] methods. In particular, point-wise methods estimate a ranking score for each individual document. Then, the documents can be ranked based on the ranking scores [3]. We follow the idea of the point-wise learning to rank techniques and propose *PathRank* to rank paths in spatial networks while considering road network topology.

2.2 Network Representation Learning

Network representation learning, a.k.a., graph embedding, aims to learn low-dimensional feature vectors for vertices while preserving network topology structure such that the vertices with similar feature vectors share similar structural properties [4, 16]. A representative method is DeepWalk [16]. DeepWalk first samples sequences of vertices based on truncated random walks, where the sampled vertex sequences capture the connections between vertices in the graph. Then, skip-gram model [13] is used to learn low-dimensional feature vectors based on the sampled vertex sequences. Node2vec [4] considers higher order proximity between vertices by maximizing the probability of occurrences of subsequent vertices in fixed length random walks. A key difference from DeepWalk is that node2vec employs biased-random walks that provide a trade-off between breadth-first and depth-first searches, and hence achieves higher quality and more informative embedding than DeepWalk does. We use node2vec in our experiments.

2.3 Machine Learning for Route Recommendation

Machine learning has been applied to improve route recommendation [5, 7]. Personalized routing aims to identify the best path for a specific driver but cannot rank a set of candidate paths [20]. Multitask learning is applied to model different drivers' driving behavior [9], but cannot be used directly for ranking paths. Additional attempts have been made for estimating travel time or fuel consumption distributions [6, 8, 15, 18], which are also different from ranking.

3 Preliminaries

3.1 Basic Concepts

A *road network* is modeled as a weighted, directed graph $G = (\mathbb{V}, \mathbb{E}, D, T, F)$. Vertex set $\mathbb{V}$ represents road intersections; edge set $\mathbb{E} \subset \mathbb{V} \times \mathbb{V}$ represents road segments. Functions D, T, and F maintain distances, travel times, and fuel consumption of traversing the edges in graph G.

A *path* $\boldsymbol{P} = (v_1, v_2, v_3, \ldots, v_X)$ is a sequence of vertices and each two adjacent vertices must be connected by an edge in $\mathbb{E}$.

A *trajectory* $\boldsymbol{T} = (p_1, p_2, p_3, \ldots, p_Y)$ is a sequence of GPS records pertaining to a trip, where each GPS record $p_i = (location, time)$ represents the location of a vehicle at a particular timestamp. Map matching is able to map a GPS record to a specific location on an edge in the underlying road network, thus aligning a trajectory with a path in the underlying road network. We call such paths *trajectory paths*.

We use the weighted Jaccard Similarity [5] to evaluate the similarity between two paths.

4 Training Data Generation

We proceed to elaborate how to generate a set of training paths for a trajectory path P from source s to destination d.

A naive way to generate the training path set is to simply include all paths from s to d. This is infeasible to use in real world settings since the training path set may contain a huge number of paths in a city-level road network graph, which in turn makes the training prohibitively inefficient. Thus, we aim to identify a *compact* training path set, where only a small number of paths, e.g., less than 10 paths, are included.

To this end, the classic top-k shortest path algorithm, e.g. Yen's algorithm [21], could be employed to choose top-9 shortest paths connecting s and d as the training paths. However, a serious issue of this strategy is that the top-k shortest paths are often highly similar. Thus, their similarities w.r.t. the ground truth, trajectory path P, are also similar, which adversely affect the effectiveness of the subsequent ranking score regression—if the similarities of training paths only spread over a small range, they only provide training instances for estimating ranking scores in the small range, which may make the trained model unable to make accurate estimations for ranking scores outside the small range. Thus, an ideal strategy should be providing a set of training paths whose similarities cover a large range.

To this end, we propose the second strategy using the diversified top-k shortest paths [11]. We first include the shortest path into the result. Next, we check the next shortest path P' and only include P' into the result if P' is dissimilar with all the existing paths in the result. In other words, if the similarity between P' and an existing path in the result is greater than a threshold, P' is not included into the result set. We keep this procedure until we find k paths or we examine all paths from s to d.

For each trajectory path P, we identify a set of diversified shortest paths $\{P'_i\}$. Then, $\{P'_i, sim_i\}$ serves the training data set where the similarity scores $sim_i = WeightedJaccard(P, P'_i)$ are considered as labels.

5 PathRank

We propose an end-to-end deep learning framework *PathRank* to estimate similarity scores for paths.

As shown in Figure A.1, *PathRank* consists of two neural networks—a vertex embedding network and a recurrent neural network (RNN).

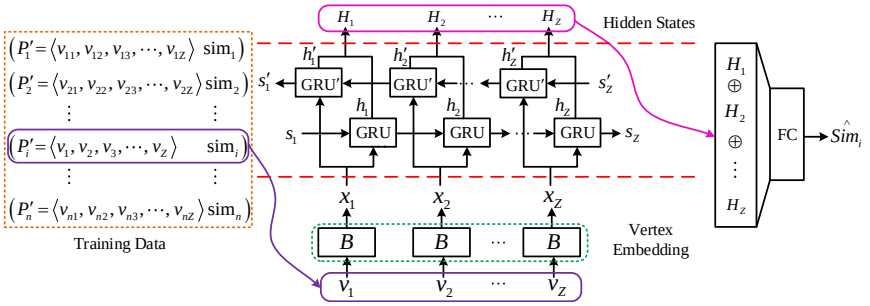


Figure A.1: Framework overview.

5.1 Vertex Embedding

We represent a vertex v_i in road network graph G as a one-hot vector $q_i \in \mathbb{R}^N$, where N represents the number of vertices in G, i.e., $N = |G.\mathbb{V}|$. Specifically, the i-th vertex v_i in graph G is represented as a vector q_i where the i-th bit is 1 and the other $N-1$ bits are 0.

Vertex embedding employs an embedding matrix $B \in \mathbb{R}^{M \times N}$ to transfer a vertex's one-hot vector q_i into a new feature vector $x_i = Bq_i \in \mathbb{R}^M$. The feature vector is often in a smaller space, where $M < N$.

Graph embedding aims at learning low dimensional, latent representations of vertices in a graph by taking into account the graph topology. Here, we apply an existing graph embedding method, node2vec [4], to initialize embedding matrix B so that the road network topology is well captured. In addition, during training, we allow *PathRank* to update B to better fit the similarity score regression task.

Given a training path $P'_i = \langle v_1, v_2, \ldots, v_Z \rangle$, we apply the same embedding matrix B to transfer each vertex to a feature vector. Thus, the training path P is represented as a sequence of features $\langle x_1, x_2, \ldots, x_Z \rangle$, where $x_j = Bq_j$ and $1 \leq j \leq Z$.

5.2 RNN

The feature sequence represents the flow of travel on path P'_i and we would like to capture the flow. To this end, we fed the feature sequence $\langle x_1, x_2, \ldots, x_Z \rangle$

into a recurrent neural network, which is known to be effective for modeling sequences. Specifically, we employ a bidirectional gated recurrent neural network (BD-GRU) [12] to capture the sequential dependencies in both the direction and the opposite direction of the travel flow.

We consider the direction of the travel flow first, i.e., from left to right. A GRU unit learns sequential correlations by maintaining a hidden state $\mathbf{h}_j \in \mathbb{R}^M$ at position j, which can be regard as an accumulated information of the positions on the left of position j. Specifically, $h_j = GRU(x_j, \mathbf{h}_{j-1})$, where x_j is the input feature vector at position j and $\mathbf{h}_{j-1}$ is the hidden state at position $j-1$, i.e., the hidden state of the left position.

For the opposite direction of the travel flow, i.e., from right to left, we apply another GRU to generate hidden state $\mathbf{h}'_j = GRU'(x_j, \mathbf{h}'_{j+1})$. Here, the input consists of the feature vector at position j and the hidden state at position $j+1$, i.e., the right hidden state. The final hidden state H_j at position j is the concatenation of the hidden states from both GRUs, i.e., $H_j = \mathbf{h}_j \oplus \mathbf{h}'_j$ where $\oplus$ indicates the concatenation operation.

5.3 Fully Connected Layer

We stack all outputs from the BD-GRU units into a long feature vector $f_i = \langle H_1 \oplus H_2 \oplus \ldots \oplus H_Z \rangle$ where $\oplus$ indicates the concatenation operation. Then, we apply a fully connected layer with weight vector $W_{FC} \in \mathbb{R}^{|f_i| \times 1}$ to produce a single value $\hat{sim}_i = f_i^T W_{FC}$, as the estimated similarity for the training path P'_i.

5.4 Loss Function

The loss function of the basic framework is shown in Equation A.1.

$$\mathcal{L}(\mathbf{W}) = \frac{1}{|n|} \sum_{i=1}^{n} \left(\hat{sim}_i - sim_i\right)^2 + \lambda \|\mathbf{W}\|_2^2 \tag{A.1}$$

The first term of the loss function measures the discrepancy between the estimated similarity $\hat{sim}_i$ and the ground truth similarity sim_i. We use the average of square error to measure the discrepancy, where n is the total number of training paths we used for training.

The second term of the loss function is a L2 regularizer on all learnable parameters in the model, including the embedding matrix B, multiple matrices used in BD-GRU, and the matrix in the final fully connected layer W_{FC}. Here, λ controls the relative importance of the second term w.r.t. the first term.

5.5 Experiments

5.6 Experiments Setup

Road Network and Trajectories

We consider the road network in North Jutland, Denmark. We use a substantial GPS data set occurred on the road network, which consists of 180 million GPS records for a two-year period from 183 vehicles. The sampling rate of the GPS data is 1 Hz (i.e., one GPS record per second). We split the GPS records into 22,612 trajectories representing different trips. We map match the GPS trajectories, such that for each trajectory, we obtain its corresponding trajectory path.

Ground Truth Data

For each trajectory T, we obtain its source s, destination d, and the trajectory path P_T. Then, we employ two different strategies to generate two sets of training paths according to the source-destination pairs (s, d)—top-k shortest paths (*TkDI*) and diversified top-k shortest paths (*D-TkDI*). For each training path P, we employ weighted Jaccard similarity $\mathit{WeightedJaccard}(P, P_T)$ as P's *ground truth* ranking score.

Evaluation Metrics

We evaluate the accuracy of the proposed *PathRank* framework based on two categories of metrics. The first category includes Mean Absolute Error (MAE) and Mean Absolute Relative Error (MARE). The second category includes Kendall rank correlation coefficient (denoted by τ) and Spearman's rank correlation coefficient (denoted by ρ), which measure the similarity, or consistency, between a ranking based on the estimated ranking scores and a ranking based on the ground truth ranking scores.

5.7 Experimental Results

We investigate how the different training data generation strategies affect the accuracy of *PathRank*. We first consider PR-A1, where we only use graph embedding method node2vec to initialize the vertex embedding matrix B and do not update B during training.

Table A.1 shows the results, where we categorize the training data generation strategies into two categories based on top-k shortest paths (*TkDI*) and diversified top-k shortest paths (*D-TkDI*). For each category, the best results are highlighted with underline. The best results overall is also highlighted with bold. We also show results when the embedding feature sizes are $M = 64$ and $M = 128$, respectively. The results show that (1) when using the diversified top-k paths for training, we have higher accuracy (i.e., lower MAE and MARE

and larger τ and ρ) compared to when using top-k paths, suggesting that using diversified paths is essential for accuracy; (2) a larger embedding feature size M achieves better results.

Table A.1: Training Data Generation Strategies, *PR-A1*

Strategies	M	**MAE**	**MARE**	τ	ρ
TkDI	64	0.1433	0.2300	0.6638	0.7044
	128	0.1168	0.1875	0.6913	0.7330
D-TkDI	64	0.1140	0.1830	0.6959	0.7346
	128	**0.0955**	**0.1533**	**0.7077**	**0.7492**

Next, we consider PR-A2, where the graph embedding matrix B is also updated during training to fit better the ranking score regression problem. Table A.2 shows the results. The two observations from Table A.1 also hold for Table A.2. In addition, PR-A2 achieves better accuracy than does *PR-A1*, meaning that updating embedding matrix B is useful.

Table A.2: Training Data Generation Strategies, *PR-A2*

Strategies	M	**MAE**	**MARE**	τ	ρ
TkDI	64	0.1163	0.1868	0.6835	0.7256
	128	0.1130	0.1814	0.7082	0.7481
D-TkDI	64	0.0940	0.1509	0.7144	0.7532
	128	**0.0855**	**0.1373**	**0.7339**	**0.7731**

6 Conclusion and Future work

We propose *PathRank*, a learning to rank technique for ranking paths in spatial networks. We propose an effective way to generate a set of training paths to enable effective and efficient learning. Then, we propose and end-to-end deep learning framework to enable graph embedding that takes into account road network topology. A recurrent neural network, together with the learned graph embedding, is employed to estimate the ranking scores which eventually enable ranking paths. Empirical studies conducted on a large real world trajectory set demonstrate that *PathRank* is effective and efficient for practical usage. As future work, it is of interest to apply outlier detection algorithms [10] to filter abnormal trajectories to improve the ranking quality of *PathRank*.

References

[1] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li, "Learning to rank: from pairwise approach to listwise approach," in *ICML*, 2007, pp. 129–136.

[2] V. Ceikute and C. S. Jensen, "Routing service quality–local driver behavior versus routing services," in *MDM*, vol. 1, 2013, pp. 97–106.

[3] F. C. Gey, "Inferring probability of relevance using the method of logistic regression," in *SIGIR'94*, 1994, pp. 222–231.

[4] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *SIGKDD*, 2016, pp. 855–864.

[5] C. Guo, B. Yang, J. Hu, and C. Jensen, "Learning to route with sparse trajectory sets," in *ICDE*, 2018, pp. 1073–1084.

[6] J. Hu, C. Guo, B. Yang, and C. S. Jensen, "Stochastic weight completion for road networks using graph convolutional networks," in *ICDE*, 2019, pp. 1274–1285.

[7] J. Hu, B. Yang, C. Guo, and C. S. Jensen, "Risk-aware path selection with time-varying, uncertain travel costs: a time series approach," *VLDB J.*, vol. 27, no. 2, pp. 179–200, 2018.

[8] J. Hu, B. Yang, C. S. Jensen, and Y. Ma, "Enabling time-dependent uncertain eco-weights for road networks," *GeoInformatica*, vol. 21, no. 1, pp. 57–88, 2017.

[9] T. Kieu, B. Yang, C. Guo, and C. S. Jensen, "Distinguishing trajectories from different drivers using incompletely labeled trajectories," in *CIKM*, 2018, pp. 863–872.

[10] ——, "Outlier detection for time series with recurrent autoencoder ensembles," in *IJCAI*, 2019, pp. 2725–2732.

[11] H. Liu, C. Jin, B. Yang, and A. Zhou, "Finding top-k shortest paths with diversity," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 3, pp. 488–502, 2018.

[12] S. Mangal, P. Joshi, and R. Modak, "Lstm vs. gru vs. bidirectional rnn for script generation," *arXiv preprint arXiv:1908.04332*, 2019.

[13] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[14] S. A. Pedersen, B. Yang, and C. S. Jensen, "Fast stochastic routing under time-varying uncertainty." *VLDB Journal*, to appear, 2019.

[15] ——, "A hybrid learning approach to stochastic routing," in *ICDE*, to appear, 2020.

[16] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *SIGKDD*, 2014, pp. 701–710.

[17] L. Rigutini, T. Papini, M. Maggini, and F. Scarselli, "Sortnet: Learning to rank by a neural-based sorting algorithm," in *SIGIR*, vol. 42, no. 2, 2008, pp. 76–79.

[18] B. Yang, J. Dai, C. Guo, C. S. Jensen, and J. Hu, "PACE: a path-centric paradigm for stochastic path finding," *VLDB J.*, vol. 27, no. 2, pp. 153–178, 2018.

[19] B. Yang, C. Guo, C. S. Jensen, M. Kaul, and S. Shang, "Stochastic skyline route planning under time-varying uncertainty," in *ICDE*, 2014, pp. 136–147.

[20] B. Yang, C. Guo, Y. Ma, and C. S. Jensen, "Toward personalized, context-aware routing," *VLDB Journal*, vol. 24, no. 2, pp. 297–318, 2015.

[21] J. Y. Yen, "Finding the k shortest loopless paths in a network," *management Science*, vol. 17, no. 11, pp. 712–716, 1971.

Paper B

Context-Aware Path Ranking in Road Networks

Sean Bin Yang, Chenjuan Guo, and Bin Yang

The paper has been published in the
IEEE Transactions on Knowledge and Data Engineering,
Vol. 34(7), pp. 3153–3168, 2022.

Abstract

Ranking paths becomes an increasingly important functionality in many transportation services, where multiple paths connecting a source-destination pair are offered to drivers. We study ranking such paths under specific contexts, e.g., at a departure time and for a specific driver. More specifically, we model ranking as a regression problem where we assign a ranking score to each path with the help of historical trajectories. The intuition is that if a driver's trajectory used path P at time t, we consider this as an evidence that path P is preferred by the driver at time t, thus should have a higher ranking score than other paths connecting the same source and destination. To solve the regression problem, we first propose an effective training data enriching method to obtain a compact *and* diversified *set of training paths using historical trajectories, which provides a data foundation for* efficient *and* effective *learning. Next, we propose a multi-task learning framework that considers features representing both candidate paths and contexts. Specifically, a road network embedding is proposed to embed paths into feature vectors by considering both road network topology and spatial properties, such as distances and travel times. By modeling different departure times as a temporal graph, graph embedding is used to embed departure times into feature vectors. The objective function not only considers the discrepancies on ranking scores but also the reconstruction errors of the spatial properties of the paths, which in turn improves the final ranking estimation. Empirical studies on a substantial trajectory data set offer insight into the designed properties of the proposed framework, indicating that it is effective and practical in real world settings.*

1 Introduction

Vehicular transportation reflects the pulse of a city. It plays an essential role in people's daily lives and many businesses as well as society as a whole [16]. With recent deployment of sensing technologies and continued digitization, large amounts of vehicle trajectory data are collected, which provide a solid data foundation to improve the quality of a wide variety of transportation services, such as vehicle routing [19], traffic prediction [23], and urban planning [12].

A fundamental functionality in vehicular transportation is routing. Given a source and a destination, classic routing algorithms, e.g., Dijkstra's algorithm, identify *a single optimal path* connecting the source and the destination, where the optimal path is the path with the least travel cost, e.g., the shortest path or the fastest path. However, a routing service quality study [6] shows that local drivers often choose paths that are neither shortest nor fastest, rendering classic routing algorithms often impractical in many real world routing scenarios. To contend with this challenge, a wide variety of advanced routing algorithms, e.g., skyline routing [55] and k-shortest path routing [58], are proposed to

identify *a set of optimal paths*, where the optimality is defined based on, e.g., pareto optimality or top-k least costs, which provide drivers with multiple candidate paths to choose. Commercial navigation systems, such as Google Maps and TomTom, often follow a similar strategy that suggests multiple candidate paths to drivers.

Under this context, ranking such candidate paths is essential for ensuring high routing quality. Existing solutions often rely on simple heuristics, e.g., ranking paths w.r.t. their travel times. However, travel times may not always be the most important factor when drivers choose paths, and a routing quality study shows that drivers often do not choose the fastest paths [6]. In addition, existing solutions often provide the same ranking to all drivers but ignore distinct preferences which different drivers may have.

In this paper, we propose a data-driven, context-aware ranking framework *PathRank* to rank paths in road networks. More specifically, *PathRank* models ranking candidate paths as a *regression* problem—for each candidate path, *PathRank* estimates a ranking score using local drivers' trajectories, which in turn enables ranking the candidate paths w.r.t. their ranking scores. The framework is flexible where different contextual information can be accommodated. For example, when accommodating driver information, it enables personalized ranking. To enable *PathRank*, two challenges must be addressed.
Enriching Training Data: To train any regression model, we need to prepare training data. We borrow the idea often used in ranking products in online shops. If a user clicks a specific product on a webpage, it provides evidence that the user is interested in the product than other products on the same webpage. Then, the clicked vs. not clicked products are considered as positive vs. negative training data to enable traning. Similarly, if a driver's trajectory used path P from source s to destination d at time t, it is an evidence that the driver considered path P as the preferred path over other paths from s to d at time t. Thus, path P is a positive training data and should have the largest ranking score. However, trajectories only provide positive training data and we still lack negative training data. Since there often exist a large amount of paths from a source to a destination, it is thus prohibitive to include all paths other than P as the negative paths. In contrast, randomly selecting a small subset of such paths may adversely affect the training effectiveness. Thus, it is challenging to select a *compact* and *diversified* training path set to represent the negative training data. A compact set ensures training *efficiency* and a diversified set ensure training *effectiveness*.
Effective Feature Representations: Effective regression models often rely on meaningful feature representations of input data. In our setting, an input to *PathRank* is a path that is a sequence of vertices in a road network graph. Here, a meaningful feature space should take into account both the topology of the underlying road network and the spatial properties of the road network, such as distances and travel times, which may influence drivers' choices.

However, no existing methods are able to capture both topological and spatial properties. In addition, it is also important to embed context information, such as departure times, into meaningful representations, where, for example, temporal closeness can be preserved. This calls far new feature learning methods.

To contend with the first challenge, we propose an effective method to generate a compact and diversified training path set. We consider different travel costs that drivers may consider, e.g., distance, travel time, and fuel consumption. Next, for each travel cost, we identify a set of *diversified*, *top-k least-cost paths*. Here, two paths are diversified if the path similarity between them is smaller than a threshold, where a number of different path similarity functions can be applied [39]. As an example, diversified top-3 shortest paths consist of three paths where the path similarity of every pair of paths is smaller than the threshold and there does not exist another set of three paths which are mutually diversified and whose total distance is shorter. Considering diversity avoids including top-3 shortest paths where they only differ slightly, e.g., only one or two edges. This method makes sure that the candidate path set is diversified because the set (i) considers multiple travel costs that a driver may consider when making routing decisions; and (ii) includes paths that are dissimilar with each other. These together represent a large feature space of the underlying road network. In addition, the set is also compact since it only includes a small number of top-k paths.

Next, we address the second challenge by proposing an end-to-end learning framework to learn feature representations of paths, which capture both topological and spatial properties. Recall that the input is a path that is represented as a sequence of vertices in a road network graph. To capture the topology of the road network, we utilize unsupervised graph embedding [15] to transform vertices into feature vectors by considering road network topology. Since recurrent neural networks (RNNs) are good at modeling sequential information and since a path is a sequence of vertices, we employ an RNN to model the sequence of the feature vectors of the vertices in a path. So far, the framework already considers the topology of the underlying road network, but still lacks spatial properties, which are not captured by classic graph embedding. To accommodate the spatial properties, we let the RNN estimate multiple values, including a ranking score of the input path and also the input path's spatial properties, such as the length, the travel time, and the fuel consumption of the path. This makes the framework a *multi-task* learning framework where the *main task* is to estimate the ranking score, which is used for the final ranking, and the *auxiliary tasks* enforce to update the feature vectors of the vertices to also capture the spatial properties of the underlying road network, which eventually also help improve the accuracy of the main task.

The proposed learning framework is flexible where contextual information

can be seamlessly integrated. For example, we propose a temporal graph to model peak vs offpeak periods in different days and then departure times can be converted into feature vectors that reflect temporal closeness. We show how the temporal features can be integrated into the learning framework and thus enable temporal ranking. Similarly, when incorporating feature vectors representing drivers, we enable personalized ranking.

This paper presents the first data-driven, end-to-end solution to context-aware ranking for paths in road networks. Specifically, we make four contributions. First, we propose a method to generate a compact and diversified set of training paths which enables effective and efficient learning. Second, we propose a multi-task learning framework to enable spatial network embedding that captures not only topological information but also spatial properties. Third, we integrate contextual information embedding into the framework to enable context-aware ranking. Fourth, we conduct extensive experiments using a large real world trajectory set to offer insight into the design properties of the proposed framework and to demonstrate that the framework is effective. A preliminary four-page report on the study appeared elsewhere [57].

2 Related Work

We review related studies on (1) learning to rank in the context of information retrieval, (2) graph representation learning, (3) machine learning techniques for path recommendation, and (4) top-k path finding.

2.1 Learning to rank

Learning to rank plays an important role in ranking in the context of information retrieval (IR), where the primary goal is to learn how to rank documents or web pages w.r.t. queries, which are all represented as feature vectors. Learning to rank methods in IR can be categorized into point-wise, pair-wise, and list-wise methods. Point-wise methods estimate a ranking score for each individual document. Then, the documents can be ranked based on the ranking scores [35]. Pair-wise methods focus on, for a given pair of documents, making a binary decision on which document is better, i.e., a relative order. Here, although we do not know the ranking scores for individual documents, we are still able to rank documents based on the estimated relative orders [27]. List-wise methods take into account a set of documents and estimate the ranking for the documents [1].

Although learning to rank techniques have been applied widely and successfully in IR, they only consider textual documents and queries and cannot be applied for ranking paths in road networks, since both graph topology and spatial properties, which are the two most important factors in

road networks, are ignored. We follow the idea of the point-wise learning to rank techniques in IR and propose *PathRank* to rank paths in road networks while considering both graph topology and spatial properties.

2.2 Graph Representation Learning

Graph representation learning, a.k.a., graph embedding, aims to learn low-dimensional feature vectors for vertices while preserving graph topology structure such that the vertices with similar feature vectors share similar structural properties [4, 15, 47, 49, 50]. We distinguish two categories of methods: random walk based methods and deep learning based methods.

A representative method in the first category is DeepWalk [47]. DeepWalk first samples sequences of vertices based on truncated random walks, where the sampled vertex sequences capture the connections between vertices in the graph. Then, skip-gram model [40] is used to learn low-dimensional feature vectors based on the sampled vertex sequences. Node2vec [15] considers higher order proximity between vertices by maximizing the probability of occurrences of subsequent vertices in fixed length random walks. A key difference from DeepWalk is that node2vec employs biased-random walks that provide a trade-off between breadth-first and depth-first searches, and hence achieves higher quality and more informative embedding than DeepWalk does.

To overcome the weaknesses of random walk based methods, e.g., the difficulty in determining the random walk length and the number of random walks, deep learning based methods utilize the random surfing model to capture contextual relatedness between each pair of vertices and preserves them into low-dimensional feature vectors for vertices [4]. Deep learning based methods are also able to take into account complex non-linear relations. GraphGAN [50] is proposed to learn vertex representations by modeling the connectivity behavior through an adversarial learning framework using a minimax game. LINE [49] does not fall into the above two categories. Instead of exploiting random walks to capture network structures, LINE [49] proposes a model with a carefully designed objective function that preserves both the first-order and second-order proximities.

However, all existing graph embedding methods consider non-spatial networks such as social networks, citation networks, and biology networks. They ignore spatial properties, e.g., distances and travel times, which are crucial features in spatial networks such as road networks. In this paper, we propose a multi-task learning framework to extend existing graph embedding to incorporate important spatial properties. Experimental results show that the graph embedding that considers spatial-properties gives the best performance when ranking paths in road networks.

2.3 Machine Learning on Spatio-Temporal Data

Machine learning has been applied to spatio-temporal data such as trajectories to improve path recommendation [19, 20, 25, 55, 56]. Personalized routing [56] and context-aware routing [19, 20] aim to identify a single, optimal path for a specific driver or under a specific context. Although such studies do not provide ranking functions directly, we derive a personalized ranking approach from [56] and compare with *PathRank* in Section 6.4. Skyline routing returns a set of non-dominated paths, which are considered to be incomparable to each other and thus no ranking is provided [45, 55]. Additional attempts have been made for estimating accurate travel time or fuel consumption distributions [23, 26, 44, 44, 46, 53, 54], which are also different from ranking paths. RoadRank [3] computes influence scores for all road segments, i.e., edges, in a road network and then ranks the edges according to the influence scores. In contrast, our paper proposes *PathRank* to rank paths, not edges. Multitask learning is applied to model different drivers' driving behavior [31] such that trajectories from a same driver can be clustered together. However, it cannot be used directly for ranking paths. In addition, one paper also considers trajectory clustering [52], which is an unsupervised learning problem. It cannot be used for solving the path ranking problem, which is a supervised learning problem.

Some traffic time series prediction methods also consider graph operations, e.g., graph convolution and graph attention, in RNNs [9, 10, 24, 37, 43], but the problem settings are different and their solutions cannot be used for ranking paths. In such models, the input to an RNN unit is a whole road network graph and the RNN units capture temporal dependency. In contrast, the input to our RNN unit is a vertex in a road network graph and the RNN units capture the spatial dependency along a path.

2.4 Top-k Queries on Road Networks

A wide variety of top-k queries on road networks exist [28, 38, 39, 41]. Top-k path selection algorithms often use simple ranking functions to rank paths [39, 58]. For example, top-k fastest path finding algorithms rank paths according to the paths' travel times. In the experiments, we compare *PathRank* with such baseline ranking functions used in top-k path finding algorithms. Some other top-k algorithms consider different problem settings. For example, top-k optimal sequenced paths aim at finding the top-k shortest paths that visit a set of points of interest (POIs) such as a post office, a bank, and a grocery store [38]. Another study considers ranking a set of POIs in a road network [41], which cannot be used for ranking paths. Probabilistic top-k shortest path queries [2, 28] rank paths w.r.t. the probability of arriving within a time budget, which is provided by end users. Our problem dose not require

end users to provide such time budgets.

3 Preliminaries

3.1 Basic Concepts

A *road network* is modeled as a weighted, directed graph $G = (\mathbb{V}, \mathbb{E}, D, T, F)$. Vertex set $\mathbb{V}$ represents road intersections and road ends; edge set $\mathbb{E} \subset \mathbb{V} \times \mathbb{V}$ represents road segments. Functions D, T, and F maintain the travel costs of the edges in graph G. Specifically, function $D : \mathbb{E} \rightarrow \mathbb{R}^+$ maps each edge to its length. Functions T and F have similar signatures and maps edges to their travel times and fuel consumption, respectively.

A *path* $P = (v_1, v_2, v_3, \ldots, v_X)$ is a sequence of X vertices where $X > 1$ and each two adjacent vertices must be connected by an edge in $\mathbb{E}$. We use $P.s$ and $P.d$ to denote the source and the destination of path P.

A *trajectory* $\boldsymbol{T} = (p_1, p_2, p_3, \ldots, p_Y)$ is a sequence of GPS records pertaining to a trip, where each GPS record $p_i = (location, time)$ represents the location of a vehicle at a particular timestamp. The GPS records are ordered according to their corresponding timestamps, where $p_i.time < p_j.time$ if $1 \leq i < j \leq Y$.

Map matching [42] is able to map a GPS record to a specific location on an edge in the underlying road network, thus aligning a trajectory $\boldsymbol{T}$ with a path in the underlying road network, denoted as $\boldsymbol{T}.P$. We call such paths *trajectory paths*. In addition, a trajectory $\boldsymbol{T}$ is also associated with a driver identifier, denoted as $\boldsymbol{T.driver}$, indicating who made the trajectory. From trajectory $\boldsymbol{T}$, we know that driver $\boldsymbol{T.driver}$ used path $\boldsymbol{T}.P$ at time $\boldsymbol{T}.p_1.time$. Thus, path $\boldsymbol{T}.P$ is considered as a ground truth path under the contexts, i.e., for driver $\boldsymbol{T.driver}$ at time $\boldsymbol{T}.p_1.time$.

Path Similarities: Multiple similarity functions [14, 19, 39, 56] are available to calculate the similarity between two paths, where the most popular functions belong to the Jaccard similarity function family, in particular, the weighted Jaccard similarity [19, 56]. In this paper, we use the weighted Jaccard Similarity (see Equation B.1) to evaluate the similarity between two paths.

$$sim\,(P_1, P_2) = \frac{\sum_{e \in P_1 \cap P_2} G.D(e)}{\sum_{e \in P_1 \cup P_2} G.D(e)} \tag{B.1}$$

Here, we use $P_1 \cap P_2$ and $P_1 \cup P_2$ to represent two edge sets: edge set $P_1 \cap P_2$ consists of the edges that appear in both P_1 and P_2; and edge set $P_1 \cup P_2$ consists of the edges that appear in either P_1 or P_2. Recall that function $G.D(e)$ returns the length of edge e. Then, the intuition of the weighted Jaccard similarity is two-fold: first, the more edges the two paths share, the more similar the two paths are; second, the longer the shared edges are, the more similar the two paths are. Note that the proposed *PathRank* is a generic

path ranking framework, which is able to easily incorporate other similarity functions.

Ranking scores: Given a trajectory path P and another path P' that also connects $P.s$ and $P.d$, we use the similarity between the two paths $sim(P, P')$ to represent the ranking score of P'. Since we consider trajectory paths as the ground truth path under the contexts, the more similar P' is w.r.t. P, the higher similarity score P' should have and thus should rank higher. The trajectory path P itself always has a ranking score of 1 and thus ranks the highest among all paths connecting $P.s$ and $P.d$.

3.2 Problem Definition

Given a set of N candidate paths $\mathbb{P}$ that connect the same source and destination and optional contexts such as a departure time and a driver identifier, we aim at (1) estimating a ranking score $sim(P, P'_i)$ for each candidate path $P'_i \in \mathbb{P}$; and (2) providing a ranked list of the candidate paths $\langle P'_1, P'_2, \ldots, P'_N \rangle$, such that $sim(P, P'_i) \geq sim(P, P'_j)$ when $1 \leq i < j \leq N$.

3.3 *PathRank* Overview

Fig. B.1 shows an overview of the proposed *PathRank*. We distinguish a training phase and a testing phase. The training phase employ historical trajectories to train *PathRank*, and we use the trained *PathRank* in the testing phase.

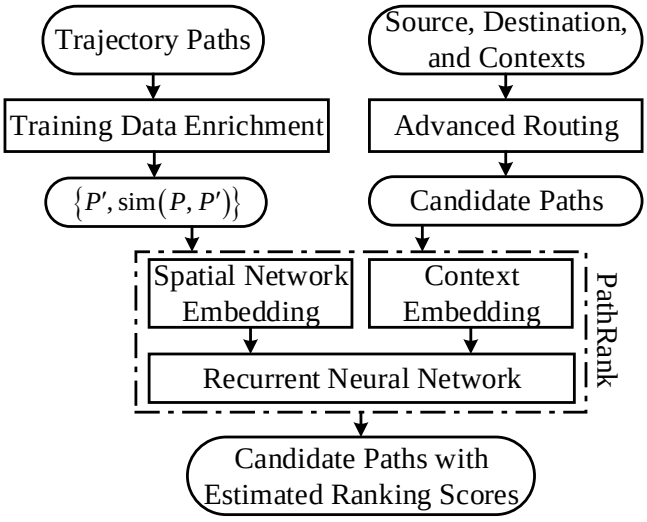


Figure B.1: Solution Overview.

We proceed to elaborate the training phase. Given a set of historical trajectory, we first map match them to obtain their corresponding *trajectory paths*. The trajectory paths are fed into the *Training Data Enrichment* module where an enriched training data set is generated. Specifically, for each trajectory path P, the training data enrichment module generates a compact and diversified set $\mathcal{PS}$ of candidate paths such that each candidate path $P' \in \mathcal{PS}$ also connects the same source and destination of the trajectory path P. In addition, for each path $P' \in \mathcal{PS}$, the module computes a similarity score $sim(P, P')$ as the ground truth ranking score of P'. Thus, the output of the training data enrichment module is a set of "candidate path" and "ranking score" pairs, denoted as $\{P', sim(P, P')\}$, where the ranking scores are labels. This set is used as the input for the *PathRank*.

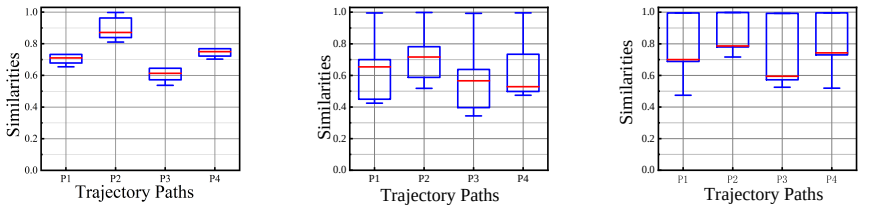
4 Training Data Enrichment

We proceed to elaborate how to generate a compact and diversified set of training paths for a trajectory path.

4.1 Intuitions

Ranking paths is similar to ranking products in online shops. If a user clicks a specific product, it provides evidence that the user is interested in the product than other similar products. Similarly, a trajectory path P from a source s to destination d at time t also provides strong evidence that a driver prefers path P than other paths that connect s to d at time t. The main difference is that, in online shops, the other similar products, i.e., competitor products, can be obtained explicitly, e.g., those products that are shown to the user in the same web page but are not clicked by the user. Based on the positive and negative training data, i.e., the products that are clicked and not clicked by the user, effective learning mechanism, e.g., learning to rank [5, 29], is available to learn an appropriate ranking function. However, in our setting, other candidate paths are often unknown and implicit because we do not know when the driver made the decision to take path P, what other paths were in driver's mind. Thus, the main target of the training data enrichment module is to generate a set of paths $\mathcal{PS}$ that include the other paths that the driver has considered. We call $\mathcal{PS}$ *competitive path set*.

A naive way to generate the competitive path set is to simply include all paths from s to d. This is infeasible to use in real world settings since the competitive path set may contain a huge number of paths in a city-level road network graph, which in turn makes the training prohibitively inefficient. Thus, we aim to identify a *compact* competitive path set, where only a small number of paths, e.g., less than 10 paths, are included. However, we cannot



(a) Top-9 Shortest Paths **(b)** Diversified Top-9 Shortest Paths **(c)** Multi-cost Diversified Paths

Figure B.2: Similarity Spreads of Different Strategies.

just randomly choose a small number of paths. We need to carefully choose such paths to resemble "the unclicked products" in online shopping.

4.2 Top-k Shortest Paths

The first strategy is to employ a classic top-k shortest path algorithm, e.g., Yen's algorithm [58], to include the top-k shortest paths from s to d into the competitive path set $\mathcal{PS}$. This provides us a compact set. In addition, this strategy is simple and efficient since a wide variety of efficient algorithms are available to generate top-k shortest paths in the literature [21, 58]. However, a serious issue of this strategy is that the top-k shortest paths are often highly similar. For example, the top-2 shortest paths may only differ with one or two edges. Thus, their similarities w.r.t. the ground truth, trajectory path P, are also similar, which adversely affect the effectiveness of the subsequent ranking score regression.

For example, we randomly choose four trajectory paths with different sources and destinations. For each trajectory path, we identify its origin and destination. Then, we use the origin and destination to generate top-9 shortest paths. Then, we compute the competitive paths' similarities w.r.t. the trajectory path. Figure B.2a shows the box plots of the similarities per trajectory path. We observe that the similarities often only spread over a very small range. For example, for the first trajectory path P_1, its corresponding top-9 shortest paths have similarities spreading from 0.65 to 0.75.

If the similarities of competitive paths only spread over a small range, they only provide training instances for estimating ranking scores in the small range, which may make the trained model unable to make accurate estimations for ranking scores outside the small range. Thus, an ideal strategy should be providing a set of training paths whose similarities cover a large range. In other words, we aim at getting a diversified competitive path set to ensure effectiveness. To this end, we propose the second strategy using the diversified top-k shortest paths [39].

4.3 Diversified Top-k Shortest Paths

Diversified top-k shortest paths finding aims at identifying top-k shortest paths such that the paths are mutually dissimilar, or diverse, with each other. First, we always include the shortest path into the diversified top-k shortest path set, say *DkPS*. Next, we iteratively check the next shortest path P_i until we have included k paths in *DkPS* or we have checked all paths connecting the source and destination. When checking the next shortest path P_i, we include P_i into *DkPS* if the similarity between P_i and each existing path in *DkPS* is smaller than a threshold δ. This means that P_i is sufficiently dissimilar with the paths in *DkPS*, thus making sure that *DkPS* is a diversified top-k shortest path set. The smaller the threshold δ is, the more diverse the paths in *DkPS* are. However, if the threshold δ is too small, it may happen that less than k diverse shortest paths or even only the shortest path can be included in *DkPS*.

Figure B.2b shows the similarities of the same four trajectory paths when using diversified top-9 shortest paths with threshold $\delta = 0.8$. We observe that the similarities spread over larger ranges compared to Figure B.2a when using classic top-k shortest paths.

4.4 Considering Multiple Travel Costs

Recent studies on personalized routing [11, 19, 56] suggest that a driver may consider different travel costs, e.g., travel time, distance, and fuel consumption, when making routing decisions. This motivates us to consider multiple travel costs, but not only distance, when generating competitive path sets. The first option to do so is to use Skyline routing [55], which is able to identify a set of pareto-optimal paths, a.k.a., Skyline paths, when considering multiple travel costs. However, Skyline routing also suffers the high similarity problem that the classic top-k shortest paths have—it often happens that the skyline paths are mutually similar, which may adversely affect the training effectiveness.

We propose a simple yet effective approach. We run the diversified top-k shortest paths x times where each time we consider a specific travel cost. Then, we use the union of the diverse paths as the final competitive path set *PS*. For example, when considering three travel costs, i.e., distances, travel times, and fuel consumption, we set $x = 3$ and identify the diversified top-k shortest, fastest, and most fuel efficient paths, respectively. Then, the union of the diversified top-k shortest, fastest, and most fuel efficient paths is used as the final competitive path set *PS*.

Since we run the diversified top-k shortest path finding multiple times for different travel costs, we can use a small k for each run. For example, when we set $k = 3$ and consider three travel costs, this makes *PS* also consist of up to 9 paths including the top-3 shortest, fastest, and most fuel efficient paths. Figure B.2c shows the similarities of the same four trajectory paths when using

the multi-cost diversified paths that include the top-3 shortest, fastest, and most fuel efficient paths. The similarities in Figure B.2c spread over larger ranges and the ranges are closer to 1. This is preferred since it helps us to distinguish the rankings of "good enough" candidate paths.

To summarize, we use multi-cost, diversified top-k least-cost paths as the compact competitive path set PS for each trajectory path P. We use paths in PS and trajectory path P together as the training data, denoted as $\{(P'_i, sim_i)\}$. Here, path $P'_i \in PS \cup \{P\}$ is associated with a ranking score label $sim_i = sim(P'_i, P)$. If P'_i is a trajectory path, its ranking score sim_i is 1, which serves as a positive training data. Otherwise, the ranking score is smaller than 1, which serves as a negative training data. After identifying competitive path sets for all trajectory paths, we use $\{(P'_i, sim_i)\}$ as the training data for *PathRank*. If we do not enrich training data and only use trajectory paths for training, then they all have ranking score of 1, making it impossible to rank different paths.

5 Ranking Framework

We propose an end-to-end deep learning framework to estimate similarity scores for paths. We first propose a basic framework that consists of a spatial network embedding network and a recurrent neural network. Next, we extend the basic framework with the help of contextual embedding by considering two contexts, i.e., departure time and driver identifiers.

5.1 Basic Framework

Recall that the input for *PathRank* is a path, i.e., competitive path P'_i, and the label of the input is its ranking score, i.e., similarity, sim_i. To solve the ranking score regression problem, a prerequisite is to represent the input path P'_i into an appropriate feature space. To this end, we propose to use a vertex embedding network to convert each vertex in the input path to a feature vector. Since a path is a sequence of vertices, after vertex embedding, the path becomes a sequence of feature vectors. Next, since recurrent neural networks (RNNs) are capable of capturing dependency for sequential data, we employ an RNN to model the sequence of feature vectors. The RNN finally outputs an estimated ranking score, which is compared against the ground truth ranking score sim_i. This results in the basic framework of *PathRank*, which consists of two neural networks—a vertex embedding network and a recurrent neural network (RNN), as shown in Figure B.3.

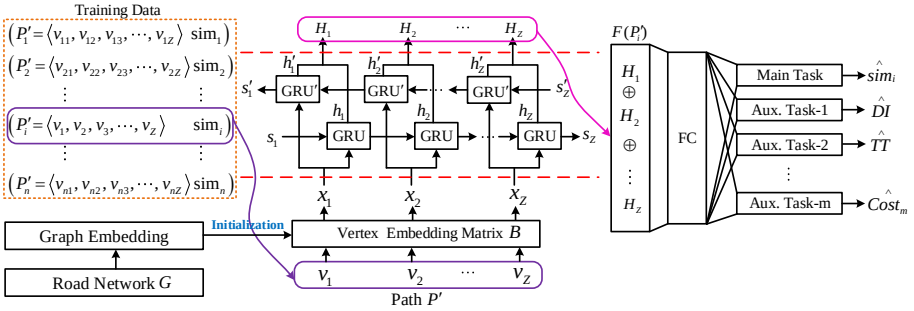


Figure B.3: Basic Framework of *PathRank*.

Vertex Embedding

We represent a vertex v_i in road network graph G as a one-hot vector $q_i \in \mathbb{R}^N$, where N represents the number of vertices in G, i.e., $N = |G.\mathbb{V}|$. Specifically, the i-th vertex v_i in graph G is represented as a vector q_i where the i-th bit is 1 and the other $N-1$ bits are 0. Vertex embedding employs an embedding matrix $B \in \mathbb{R}^{M \times N}$ to transfer a vertex's one-hot vector q_i into a new feature vector $x_i = Bq_i \in \mathbb{R}^M$. The feature vector is often in a smaller space, where $M < N$.

Given a competitive path $P'_i = \langle v_1, v_2, \ldots, v_Z \rangle$, we apply the same embedding matrix B to transfer each vertex to a feature vector. Thus, the competitive path P is represented as a sequence of features $\langle x_1, x_2, \ldots, x_Z \rangle$, where $x_j = Bq_j$ and $1 \leq j \leq Z$.

Next, we elaborate different means of obtaining embedding matrix B. An naive method to obtain B is to simply initialize a random matrix, which is then updated through back-propagation in the training phase. However, the naive method does not consider the graph topology and spatial properties, which hinders accuracy.

Capturing Graph Topology with Graph Embedding: Graph embedding, e.g., DeepWalk [47], node2vec [15], LINE [49], GraphGAN [50], aims at learning low-dimensional, feature vectors of vertices in a graph by taking into account the graph topology. A typical way to enable graph embedding is to mimic the way of embedding words for natural languages [15, 47]. In particular, multiple vertex sequences can be generated by using random walks, where random walks can consider edge weights or ignore edge weights. Next, vertices are considered as words and the generated vertex sequences are considered as sentences, which enables the use of word embedding techniques to generate embeddings for vertices. Since the vertex sequences are generated by applying random walks on the graph, the obtained vertex embedding actually already takes into account the graph topology. The output of graph embedding is an

embedding matrix B that considers graph topology.

We propose two different strategies to incorporate graph embedding into the framework. First, we simply apply an existing graph embedding method, e.g., DeepWalk or node2vec, to obtain embedding matrix B that embeds a one-hot representation of a vertex to a low dimensional feature vector. Then, we use the feature vector as the input to the RNN. This means that *PathRank* only includes an RNN module, whose inputs are sequences of feature vectors, and the vertex embedding module only provides the inputs and are then disconnected from *PathRank*.

Second, inspired by the well-known practice of unsupervised pre-training [13], we use the embedding matrix obtained from an existing graph embedding method to initialize the embedding matrix B in the vertex embedding module in *PathRank*. This allows *PathRank* to update the embedding matrix B during training such that it not only captures the graph topology but also better fits the similarity regression.

Capturing Spatial Properties with Multi-Task Learning: Although many vertex embedding algorithms exist, they are only able to capture graph topology because they only focus on graphs representing, e.g., social networks and citation network. In other words, they do not consider graphs representing spatial networks such as road networks. However, in road network graphs, many spatial attributes, in addition to topology, are also very important. For example, distances and travel times between two vertices are crucial features for road networks and also influence drivers' path choices. To let the graph embedding also maintain the spatial properties, we design a multi-task learning framework using pre-trained graph embedding.

We first employ an existing graph embedding algorithm to initialize the vertex embedding matrix B in the vertex embedding module of *PathRank*. This pre-trained embedding matrix captures the graph topology. Next, we try to update B such that it also captures relevant spatial properties during training. To this end, we employ multi-task learning principles, where the main task is to estimate similarity and the auxiliary tasks are to reconstruct travel costs of competitive paths which help learning an appropriate embedding matrix B that also considers spatial properties of the underlying road network.

RNN

After vertex embedding, a path is represented by a sequence of feature vectors $\langle x_1, x_2, \ldots, x_Z \rangle$. The feature sequence represents the flow of travel on path P'_i. As a recurrent neural network (RNN) is known to be effective for modeling sequences, we fed the feature sequence $\langle x_1, x_2, \ldots, x_Z \rangle$ into an RNN. Specifically, we employ a bidirectional gated recurrent neural network (BD-GRU) [8] to capture the sequential dependencies in both the direction and the opposite direction of the travel flow on path P'_i.

We consider the direction of the travel flow first, i.e., from left to right. A GRU unit learns sequential correlations by maintaining a hidden state $h_j \in \mathbb{R}^Q$ at position j, which can be regard as an accumulated information of the positions to the left of position j. Specifically, $h_j = GRU(x_j, h_{j-1})$, where x_j is the input feature vector at position j and h_{j-1} is the hidden state at position $j-1$, i.e., the hidden state of the left position. More specifically, the GRU unit is composed of the following computations.

First, the GRU unit employs a reset gate r_j, shown in Equation B.2, to decide how much information from the previous position should be forgotten. Equation B.2 computes r_j, which is a value between 0 and 1, meaning that the reset gate may fully forget to fully remember. The GRU then uses a similar gate called update gate to compute z_j using Equation B.3. Both the reset and update gates are contributed to control how much information from the left hidden states should be considered in order to make the final similarity score estimation accurate. More specifically, In Equation B.4, the GRU computes an internal state $\tilde{h}_j$ that considers both inputs x_j and h_{j-1}. Here, the output of the reset gate r_j is used to control how much we want to consider the output from the previous position h_{j-1}. Finally, In Equation B.5, the GRU uses the update gate z_j to combine the internal state $\tilde{h}_j$ and the output from the previous position h_{j-1}, which produces the output state h_j for the current GRU unit at position j. By doing this, it is possible to remember and forget left hidden states which are found to be relevant and irrelevant for the final similarity score estimation.

$$\mathbf{r}_j = \sigma\left(\mathbf{W}_r x_j + \mathbf{U}_r \mathbf{h}_{j-1}\right) \tag{B.2}$$

$$\mathbf{z}_j = \sigma\left(\mathbf{W}_z x_j + \mathbf{U}_z \mathbf{h}_{j-1}\right) \tag{B.3}$$

$$\tilde{\mathbf{h}}_j = \phi\left(\mathbf{W}_h x_j + \mathbf{U}_h\left(\mathbf{r}_j \odot \mathbf{h}_{j-1}\right)\right) \tag{B.4}$$

$$\mathbf{h}_j = \mathbf{z}_j \odot \mathbf{h}_j + \left(1 - \mathbf{z}_j\right) \odot \tilde{\mathbf{h}}_j \tag{B.5}$$

where σ is the logistic function, and $\odot$ denotes Hadamard product and ϕ is hyperbolic tangent function. x_j and $\mathbf{h}_j$ are the feature vector and hidden state at position j, respectively. $\mathbf{W}_r$, $\mathbf{W}_z$, $\mathbf{W}_h$, $\mathbf{U}_r$, $\mathbf{U}_z$ and $\mathbf{U}_h$ are parameters to be learned.

For the opposite direction of the travel flow, i.e., from right to left, we apply another GRU to generate hidden state $\mathbf{h}'_j = GRU'(x_j, \mathbf{h}'_{j+1})$. Here, the input consists of the feature vector at position j and the hidden state at position $j+1$, i.e., the right hidden state.

The final hidden state H_j at position j is the concatenation of the hidden states from both GRUs, i.e., $H_j = \mathbf{h}_j \oplus \mathbf{h}'_j$ where $\oplus$ indicates the concatenation operation. We stack all outputs from the BD-GRU units into a long feature vector $F(P'_i) = \langle H_1 \oplus H_2 \oplus \ldots \oplus H_Z \rangle$ where $\oplus$ indicates the concatenation

operation. Now, the competitive path P_i' is converted to a feature vector $F(P_i')$.

Fully Connected Layer

For each competitive path P_i', we apply a fully connected layer with weight vector $W_{FC} \in \mathbb{R}^{|F(P_i')| \times X}$ to produce a vector of X values, including the estimated similarity score $\hat{sim}_i$ and a number of spatial properties, such as travel time, distance, and fuel consumption.

Loss Function

To enable the multi-task learning framework, in the final fully connected layer, *PathRank* not only estimates a similarity score but also reconstruct the spatial properties of the corresponding competitive path P_i', such as the distance, travel time, and fuel consumption of P_i'. The loss function for the multi-task learning framework is defined in Equation B.6.

$$\mathcal{L}(\mathbf{W}) = \frac{1}{|n|}[(1-\alpha) \cdot \sum_{i=1}^{n} \left(\hat{sim}_i - sim_i\right)^2 + \alpha \cdot \sum_{i=1}^{n} \sum_{k=1}^{m} \left(\hat{y}_i^{(k)} - y_i^{(k)}\right)^2] + \lambda \|\mathbf{W}\|_2^2 \tag{B.6}$$

The first term of the loss function measures the discrepancy between the estimated similarity $\hat{sim}_i$ and the ground truth similarity sim_i. We use the average of square error to measure the discrepancy, where n is the total number of competitive paths we used for training. The second term of the loss function represents auxiliary tasks that consider the discrepancies between the actual spatial properties vs. the estimated spatial properties. More specifically, $\hat{y}_i^{(k)}$ and $y_i^{(k)}$ denote the estimated cost of the k-th auxiliary task and the ground truth of the k-th auxiliary task, respectively. For example, when considering distance, travel time, and fuel consumption, we set m to 3; and $\hat{y}_i^{(k)}$ and $y_i^{(k)}$ represent the estimated and ground truth distance, travel time, or fuel consumption of the i-th competitive path P_i'. α is a hyper parameter that controls the trade-off between main task and auxiliary tasks. Finally, the loss function uses a L2 regularizer on all learnable parameters in the model, including the embedding matrix B, multiple matrices used in BD-GRU, and the matrix in the final fully connected layer W_{FC}.

5.2 Advanced Framework

Path ranking is often context dependent. For example, during peak vs off-peak hours, drivers may consider different paths as the best paths. To accommodate such contexts, we design an advanced framework to extend the basic framework with the help of contextual embedding by considering a departure time

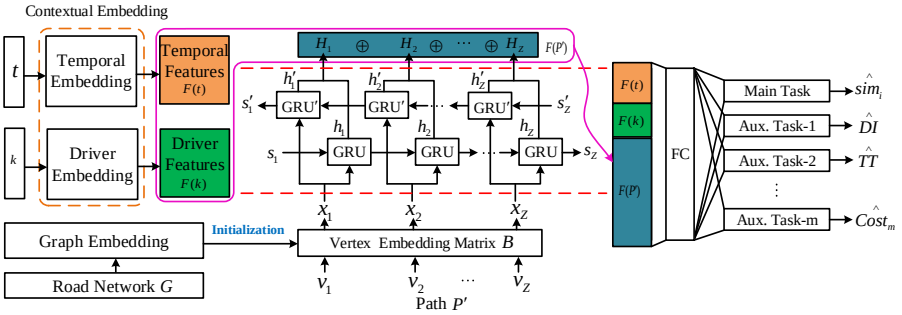


Figure B.4: Advanced *PathRank* Overview.

t and a driver ID k. The advanced *PathRank* framework is shown in Figure B.4. We proceed to describe the embedding of various contexts such as departure times and driver IDs. Departure time is an important context when drivers make routing decisions as it often correlates with traffic conditions which affect heavily drivers' routing decisions. We aim at embedding departure time into a meaningful feature space such that the ranking model is able to take into account departure time.

We first partition a day into five intervals—a morning peak interval, an afternoon peak interval, and three off-peak intervals. Various methods are available to partition a day into peak and off-peak intervals [36, 55]. For example, an example partition is shown at the top of Figure B.5.

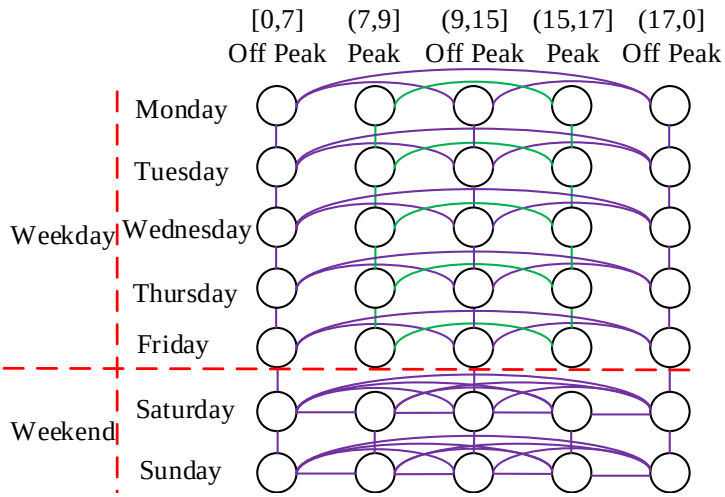


Figure B.5: Temporal Graph.

Next, we construct a temporal graph based on the intervals in different

days in a week, as shown in Figure B.5. In the graph, each node represents an interval in a specific day. If two nodes are supposed to have similar traffic, we connect the two nodes by an edge. For each weekday, we assume that the peak intervals have similar traffic situations, thus we connect the two nodes representing the two peak intervals. Similarly, we connect the nodes representing offpeak hours in the same weekday. We also connect the two nodes which represent the same interval but on two adjacent weekdays, e.g., two nodes representing the morning peak on Wednesday and the morning peak on Thursday. For each weekend, we connect all nodes in the weekend. Between the two weekends, we connect the two nodes representing the same interval.

Based on the temporal graph, we apply graph embedding to embed the nodes in the temporal graph into feature vectors. Given a departure time t, we first identify the node $node(t)$ that t belongs to in the temporal graph and then obtain its embedding $F(t) = GraphEmbed(node(t))$. Like the road network embedding, we allow the learning framework to update the temporal graph embedding to better fit the ranking score regression.

We use one-hot encoding to convert driver ID k into a multidimensional feature vector $F(k)$.

To incorporate the context features into the framework, we concatenate the context features with the path feature. Specifically, assume that the competitive path P_i' corresponds to a trajectory path that is made by driver k at departure time t, the final feature vector for the competitive path P_i' is $F(t) \oplus F(k) \oplus F(P_i')$. Then, similar to the basic framework, the feature vector is fed into a fully connected layer to estimate similarity scores and different spatial properties using the same loss function shown in Equation B.6.

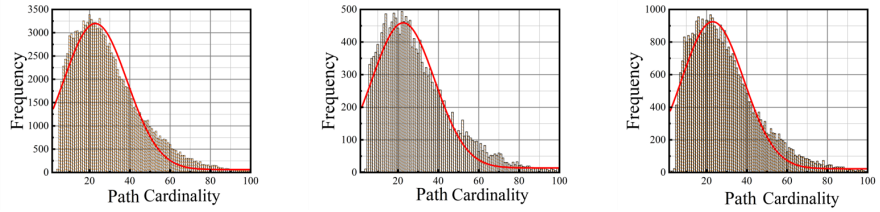
6 Experiments

We conduct a comprehensive empirical study to investigate the effectiveness of the proposed *PathRank* framework.

6.1 Experiments Setup

Road Network and Trajectories

We obtain the Danish road network from OpenStreetMap, which consists of 667,950 vertices and 818,020 edges. We use a substantial GPS data set occurred on the road network, which consists of 180 million GPS records for a two-year period from 166 drivers. The sampling rate of the GPS data is 1 Hz (i.e., one GPS record per second) and each GPS record is associated with a driver identifier. We split the GPS records into 22,612 trajectories representing



(a) Training Set **(b)** Validation Set **(c)** Testing Set

Figure B.6: Cardinalities of the trajectory paths.

different trips. A well-known map matching method [42] is used to map match the GPS trajectories such that for each trajectory, we obtain its corresponding trajectory path.

Travel costs

We consider three travel costs: travel distance (DI), travel time (TT), and fuel consumption (FC). The travel distances are computed based on the geometric information provided by OpenStreetMap. Travel times are obtained as the difference between the times of the last and first GPS records of the trajectories. We use the SIDRA-running model to compute fuel consumption based on the speeds that are obtained from the available GPS records [17]. A recent benchmark indicates that the SIDRA-running is appropriate for this purpose [18].

Ground Truth Data

We split the trajectories into three sets—70% for training, 10% for validation, and 20% for testing. The distributions of the cardinalities of the trajectory paths in training, validation, and testing sets are shown in Figure B.6.

For each trajectory T, we obtain its source s, destination d, and the trajectory path P_T. Then, we employ seven different strategies to generate seven sets of competitive paths according to the source-destination pairs (s, d).

1. Top-k shortest paths (*TkDI*);
2. Top-k fastest paths (*TkTT*);
3. Top-k most fuel efficient paths (*TkFC*);
4. Diversified top-k shortest paths (*D-TkDI*);
5. Diversified top-k fastest paths (*D-TkTT*);
6. Diversified top-k most fuel efficient paths (*D-TkFC*);

7. Diversified, multi-cost top-k paths (*D-TkM*).

For each competitive path P, we employ weighted Jaccard similarity $sim(P, P_T)$ as the *ground truth* ranking score of path P.

When training and validation, we use the competitive path set generated by a specific training data generation strategy to train a *PathRank* model. Thus, we are able to train seven different *PathRank* models using the same set of training and validation trajectories, but seven different sets of competitive paths.

When testing, to make the comparison among different *PathRank* models fair, for each testing trajectory, we merge all competitive paths generated by the 7 different strategies and randomly choose 10 paths from them. This makes sure that (1) *PathRank* models that are trained on different training data sets are tested against on the same set of competitive paths; (2) a *PathRank* model that is trained on a specific strategy is tested against competitive paths that are not generated from the same strategy.

PathRank Frameworks

We consider different variations of *PathRank*.

1. *PR-B*: the vertex embedding just employs a random initialized embedding matrix B, which ignores the graph topology. We also let $\alpha = 0$, meaning that *PR-B* has a single task on estimate similarity scores, where α is a parameter that controls the relative importance between the main task and auxiliary tasks as shown in Equation B.6.

2. *PR-A1*: vertex embedding employs graph embedding that considers graph topology, but the vertex embedding is static and is not updated during training. Only the main task is considered, i.e., $\alpha = 0$.

3. *PR-A2*: similar to *PR-A1*, graph embedding is used. In addition, the vertex embedding is updated during training. Only the main task is considered, i.e., $\alpha = 0$.

4. *PR-A2-Mx*: Similar to *PR-A2*, graph embedding is used and the vertex embedding is updated during training. In addition, multi-task learning that considers spatial properties is used. We use *PR-A2-Mx* to indicate a *PathRank* model that uses an objective function considering x spatial properties, i.e., x auxiliary tasks.

5. *PRC*: the advanced framework *PRC* with contextual embedding and multi-task learning.

For all frameworks that use graph embedding, i.e., *PR-A1*, *PR-A2*, *PR-A2-Mx* and *PRC*, we choose node2vec [15] as the graph embedding method.

Node2vec is a generic random walk based graph embedding method, which outperforms alternative methods such as DeepWalk [47] and LINE [49]. When new, better unsupervised graph embedding method becomes available, it can be easily integrated into *PathRank* to replace node2vec.

Parameters

When generating diversified top-k paths, we consider two different similarity thresholds δ—0.6 and 0.8. A smaller threshold enforces more diversified paths. However, it is also more likely that we cannot identify k paths that are significantly diversified paths, especially when k is large. Recall that the vertex embedding utilizes a embedding matrix $B \in \mathbb{R}^{M \times N}$ to embed each vertex into a M-dimensional feature vector, where N is the number of vertices. We consider two settings of M, namely 64 and 128. For the multi-task learning framework, we vary α from 0, 0.2, 0.4, 0.6, to 0.8 to study the effect on learning additional spatial properties.

We summary different parameter settings in Table B.1, where the default values are shown in bold.

Table B.1: Parameters of *PathRank*

Parameters	Values
Similarity Threshold δ	0.6, **0.8**
Vertex Embedding Feature Size M	64, **128**
Multi-task Learning Parameter α	0, 0.2, 0.4, 0.6, 0.8

Evaluation Metrics

We evaluate the accuracy of the proposed *PathRank* framework based on two categories of metrics. The first category includes metrics that measure how accurate the estimated ranking scores w.r.t. the ground truth ranking scores. This category includes Mean Absolute Error (MAE) and Mean Absolute Relative Error (MARE). Smaller MAE and MARE values indicate higher accuracy. Specifically, we have

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|x_i - \hat{x}_i|; \quad \text{MARE} = \frac{\sum_{i=1}^{n}|x_i - \hat{x}_i|}{\sum_{i=1}^{n}|x_i|} \tag{B.7}$$

where x_i and $\hat{x}_i$ represent the ground truth ranking score and the estimated ranking score, respectively; and n is the total number of estimations.

The second category includes Kendall rank correlation coefficient (denoted by τ) and Spearman's rank correlation coefficient (denoted by ρ), which measure the similarity, or consistency, between a ranking based on the estimated

ranking scores and a ranking based on the ground truth ranking scores. Sometimes, although the estimated ranking scores deviate from the ground truth ranking scores, the two rankings derived by both scores can be consistent. In this case, we consider the estimated ranking scores also accurate, since we eventually care the final rankings of the candidate paths but not the specific ranking scores for individual candidate paths. Both τ and ρ are able to measure how consistent between the two rankings. The higher the values are, the more consistent the two rankings are. If the two rankings are identical, both τ and ρ values are 1. Specifically, we have

$$\tau = \frac{N_{con} - N_{dis}}{n(n-1)/2}; \quad \rho = 1 - \frac{6\sum_{i=1}^{n} d_i^2}{n(n^2-1)} \tag{B.8}$$

Assume that we have a set of $n = 3$ candidate paths $\{P_1, P_2, P_3\}$, the ground truth ranking is $\langle P_1, P_2, P_3 \rangle$, and the estiamted ranking is $\langle P_2, P_3, P_1 \rangle$.

In τ, N_{con} and N_{dis} represent the number of path pairs are consistent and inconsistent in the two rankings. We have $N_{con} = 1$ since in both ranking, P_2 appears before P_3; and $N_{dis} = 2$ since P_1 appears before P_3 in the ground truth ranking, but P_3 appears before P_1 in the estimated ranking. Similarly, the orderings between P_1 and P_2 are also inconsistent in two rankings.

In ρ, d_i represents the rank difference on the i-th competitive path in both rankings. Following the running example, we have $d_1 = 1 - 3 = -2$ because path P_1 has rank 1 and rank 3 in both rankings, respectively.

Baselines

Baseline Ranking Heuristics: We consider seven baseline ranking heuristics. The first three baseline ranking heuristics consider a single cost, i.e., ranking the candidate paths according to only distances (DI), travel times (TT), and fuel consumption (FC). The three baseline ranking heuristics represent the ranking part of top-k path selection algorithms. For example, DI represents the ranking part of top-k shortest path selection.

Next, we consider four more baseline ranking heuristics that consider multiple travel costs—ranking the candidate paths according to both distance and travel times (DI+TT), both distance and fuel consumption (DI+FC), both travel times and fuel consumption (TT+FC), and distance, travel times and fuel consumption (DI+TT+FC). When considering more than one travel cost, we consider each travel cost equally.

1. Distances (DI);
2. Travel times (TT);
3. Fuel consumption (FC);

4. Distance and travel times (DI+TT);
5. Distance and fuel consumption (DI+FC);
6. Travel times and fuel consumption (TT+FC);
7. Distance, travel times and fuel consumption (DI+TT+FC).

Baseline Regression Methods: To justify the effectiveness of *PathRank*, we consider six regression baselines.

1. Linear Regression (LR) [48];
2. Lasso Regression [51];
3. Support Vector Regression (SVR) [7];
4. Decision Tree Regression (DT) [30];
5. Decision Tree Regression with Adaboost (DTA) [34];
6. Long Short-Term Memory (LSTM) [22], we replace the bi-directional GRU units by LSTM units.

Implementation Details

All algorithms are implemented in Tensorflow. Code is available at `https://github.com/Sean-Bin-Yang/Learning-to-Rank-Paths`. We conduct experiments on a computer node on the CLAAUDIA cloud (`www.claaudia.aau.dk`), running Ubuntu 16.04.6 LTS, with one Intel(R) Xeon(R) CPU @2.50GHz and one Tesla V100 GPU card.

6.2 Verifying the Design Choices of *PathRank*

Effects of Training Data Generation Strategies

We investigate how the different training data generation strategies affect the accuracy of *PathRank*. We first consider *PR-A1*, where we only use graph embedding method node2vec to initialize the vertex embedding matrix B and do not update B during training.

Table B.2 shows the results, where we categorize the training data generation strategies into three categories based on top-k paths, diversified top-k paths, and multi-cost, diversified top-k paths. For each category, the best results are highlighted with underline. The best results over all categories is also highlighted with bold. We also show results when the embedding feature sizes are $M = 64$ and $M = 128$, respectively.

The results show that (1) when using the diversified top-k paths for training, we have higher accuracy (i.e., lower MAE and MARE and larger τ and ρ) compared to when using top-k paths; (2) using multi-cost, diversified top-k paths achieves better accuracy compared to single-cost, diversified top-k paths, thus achieving the best results; (3) a larger embedding feature size M achieves better results.

Table B.2: Training Data Generation Strategies, *PR-A1*

Strategies	M	**MAE**	**MARE**	τ	ρ
TkDI	64	0.1433	0.2300	0.6638	0.7044
	128	0.1168	0.1875	0.6913	0.7330
TkTT	64	0.1302	0.2090	0.6642	0.7046
	128	0.1181	0.1896	0.6818	0.7208
TkFC	64	0.1208	0.1940	0.6692	0.7131
	128	0.1257	0.2019	0.6699	0.7110
D-TkDI	64	0.1140	0.1830	0.6959	0.7346
	128	0.0955	0.1533	0.7077	0.7492
D-TkTT	64	0.1050	0.1686	0.7124	0.7554
	128	0.0974	0.1564	0.7271	0.7714
D-TkFC	64	0.1045	0.1678	0.7100	0.7544
	128	0.0900	0.1445	0.7238	0.7685
D-TkM	64	0.1077	0.1729	0.7261	0.7679
	128	**0.0792**	**0.1271**	**0.7478**	**0.7876**

Next, we consider *PR-A2*, where the graph embedding matrix B is also updated during training to fit better the ranking score regression problem. Table B.3 shows the results. The three observations from Table B.2 also hold for Table B.3. In addition, *PR-A2* achieves better accuracy than does *PR-A1*, meaning that updating embedding matrix B is useful.

From the above experiments, the multi-cost, diversified top-k strategy *D-TkM* is the most promising strategy. Thus, we only use *D-TkM* for the remaining experiments.

Next, we investigate the effects on the similarity threshold δ used in the diversified top-k path finding. Specifically, we consider two threshold values 0.6 and 0.8 and the results are shown in Table B.4. When a smaller threshold is used, i.e., higher diversity in the top-k paths, the accuracy is improved.

Effects of Vertex Embedding

We investigate the effects of different vertex embedding strategies. We consider *PR-B* where we just use a randomly initialized embedding matrix B, which totally ignores graph topology. For *PR-A1* and *PR-A2* where we both use

Table B.3: Training Data Generation Strategies, *PR-A2*

Strategies	M	**MAE**	**MARE**	τ	ρ
TkDI	64	0.1163	0.1868	0.6835	0.7256
	128	0.1130	0.1814	0.7082	0.7481
TkTT	64	0.1218	0.1956	0.6858	0.7282
	128	0.1161	0.1864	0.7026	0.7446
TkFC	64	0.1216	0.1952	0.6911	0.7321
	128	0.1082	0.1737	0.7070	0.7477
D-TkDI	64	0.0940	0.1509	0.7144	0.7532
	128	0.0855	0.1373	0.7339	0.7731
D-TkTT	64	0.1010	0.1622	0.7283	0.7693
	128	0.0997	0.1600	0.7169	0.7596
D-TkFC	64	0.0938	0.1506	0.7318	0.7743
	128	0.0809	0.1299	0.7386	0.7811
D-TkM	64	0.0966	0.1551	0.7393	0.7771
	128	**0.0725**	**0.1164**	**0.7528**	**0.7905**

Table B.4: Effects of Similarity Threshold δ

	δ	M	**MAE**	**MARE**	τ	ρ
PR-A1	0.6	64	0.1006	0.1615	0.7321	0.7733
		128	0.0770	0.1237	0.7496	0.7874
	0.8	64	0.1077	0.1729	0.7261	0.7679
		128	0.0792	0.1271	0.7478	0.7876
PR-A2	0.6	64	0.0817	0.1311	0.7404	0.7792
		128	**0.0710**	**0.1140**	**0.7751**	**0.8109**
	0.8	64	0.0966	0.1551	0.7393	0.7771
		128	0.0725	0.1164	0.7528	0.7905

node2vec to embed vertices. Here, we use node2vec to embed both weighted and unweighted graphs, respectively. When embedding weighted graphs, we simply use distances as edge weights.

Based on the results in Table B.5, we observe the following. First, *PR-B* gives the worst accuracy: the estimated ranking scores have the largest errors in terms of both MAE and MARE; and the ranking based on estimated ranking scores deviates the most from the ground truth ranking in terms of both τ and ρ. This suggests that ignoring graph topology when embedding vertices is not a good choice.

Second, when embedding vertices using node2vec, whether or not considering edge weights does not significantly change the accuracy. Thus, it is not a significant design choice.

Third, *PR-A2* achieves the best accuracy in terms of both errors on estimated ranking scores and consistency between two rankings. Thus, this suggests that considering graph topology improves accuracy and updating the embedding matrix B according to the loss function on ranking scores makes the embedding matrix fit better the ranking score regression problem. This also suggests that, by including spatial properties in the loss function, it has a potential to tune the embedding matrix B to capture spatial properties, which in turn should improve ranking score regression. This is verified in the following experiments on the multi-task framework.

Table B.5: Effects of Vertex Embedding Strategies

	Embedding	**MAE**	**MARE**	τ	ρ
PR-B	—	0.1159	0.1816	0.7233	0.7611
PR-A1	unweighted	0.0878	0.1410	0.7453	0.7852
	weighted	0.0792	0.1271	0.7478	0.7876
PR-A2	unweighted	0.0734	0.1178	**0.7640**	**0.8012**
	weighted	**0.0725**	**0.1164**	0.7528	0.7905

Effects of Multi-task Learning

In the following set of experiments, we study the effects of the proposed multi-task learning framework. In particular, we investigate how much we are able to improve when incorporating different spatial properties in the loss function to let the vertex embedding also consider spatial properties, which may potentially contribute to better ranking score regression.

We start by *PR-A2-M1*, which considers only one auxiliary task on reconstructing distances. This means that *PathRank* not only estimate the ranking score of a competitive path but also tries to reconstruct the distance of the competitive paths. Table B.6 shows the results with varying α values. When $\alpha = 0$, the auxiliary task is ignored, which makes *PR-A2-M1* into *PR-A2*, i.e., its corresponding model with only the main task on estimating ranking scores. When $\alpha > 0$, i.e., the auxiliary task on distances is considered while learning, we observe that the estimated ranking scores are improved. In particular, the setting with $\alpha = 0.6$ gives the best results in terms both τ and ρ, indicating that the ranking w.r.t. the estimated ranking scores is more consistent with the ground truth ranking. When $\alpha = 0.8$, it achieves the smallest MAE and MARE. Both settings suggest that considering the additional auxiliary task on reconstructing distance helps improve the final ranking.

PR-A2-M2 includes two auxiliary tasks on reconstructing both distances and travel times, and *PR-A2-M3* includes three auxiliary tasks on reconstructing distances, travel times, and fuel consumption. All the three multi-task

models show that considering spatial properties improve the final ranking. In particular, when considering all the three spatial properties give the best final ranking in terms of τ and ρ, i.e., achieving the most consistent ranking w.r.t. the ground truth ranking.

Table B.6: Effects of α, *PR-A2-Mx*

	α	**MAE**	**MARE**	τ	ρ
PR-A2	0	0.0725	0.1164	0.7528	0.7905
PR-A2-M1	0.2	0.0756	0.1214	0.7713	0.8057
	0.4	0.0704	0.1129	0.7765	0.8110
	0.6	0.0693	0.1113	0.7783	0.8141
	0.8	0.0680	**0.1029**	0.7712	0.8057
PR-A2-M2	0.2	**0.0653**	0.1048	0.7727	0.8089
	0.4	0.0701	0.1125	0.7869	0.8235
	0.6	0.0777	0.1247	0.7752	0.8100
	0.8	0.0807	0.1296	0.7616	0.7973
PR-A2-M3	0.2	0.0724	0.1162	0.7732	0.8092
	0.4	0.0740	0.1188	0.7711	0.8090
	0.6	0.0662	0.1063	**0.7923**	**0.8261**
	0.8	0.0695	0.1116	0.7842	0.8177

Effects of Contexts Embedding

We also investigate how much we improve when adding the contextual information to the basic framework. To this end, we consider the advanced framework *PRC* where we include the departure time feature $F(t)$ and driver feature $F(k)$. Table B.7 shows that contextual information contributes to improve the overall accuracy. This also suggests that the proposed temporal graph embedding is effective. However, recall that the contextual information is an optional input. In case that departure time and driver identifiers are not provided as inputs, we can only use the basic framework.

Table B.7: Effects on Context Embeddings

	MAE	**MARE**	τ	ρ
PR-A2-M3	0.0662	0.1063	0.7923	0.8261
PRC	**0.0611**	**0.0929**	**0.8178**	**0.8454**

6.3 Comparison with Baselines

Comparison with Baseline Ranking Heuristics

We consider the baseline ranking heuristics covered in Section 6.1. For each heuristics, we obtain a ranking. Then, we compare the ranking with the ground truth ranking to compute the corresponding τ and ρ.

Table B.8 shows the comparison, where we categorize the testing cases based on the distances of the lengths of their corresponding trajectory paths into three categories (0, 5], (5, 10], and (10, 15] km. The results show that the ranking obtained by the proposed framework *PRC* is clearly the best in all categories, suggesting that the proposed multitask framework outperforms baseline heuristics.

Table B.8: Comparison with Baseline Ranking Heuristics

	(0, 5]	(5, 10]	(10, 15]
	τ/ρ	τ/ρ	τ/ρ
DI	0.7515/0.7806	0.6630/0.6860	0.3784/0.3370
TT	0.6776/0.7054	0.6712/0.7024	0.6053/0.6625
FC	0.6885/0.7192	0.3920/0.3986	0.0279/-0.0210
DI+TT	0.7146/0.7430	0.6681/0.6942	0.4919/0.4998
DI+FC	0.7200/0.7499	0.5275/0.5423	0.2032/0.1580
TT+FC	0.6831/0.7123	0.5316/0.5505	0.3166/0.3208
DI+TT+FC	0.7059/0.7351	0.5754/0.5957	0.3372/0.3262
PRC	**0.8239/0.8521**	**0.8115/0.8382**	**0.6497/0.6620**

Comparison with Regression Baselines

For the regression baseline methods covered in Section 6.1, we consider two different types of features.

- Basic features (BF): each path is represented as a 3-dimensional vector that represents its distance, travel time, and fuel consumption.
- Advanced features (AF): each path is represented as an $N \times M$ matrix, where N is the cardinality of the path. For each vertex in the path, we obtain a M-dimensional vector using node2vec. This makes an $N \times M$ matrix.

Table B.9 shows the comparison. The results show that the ranking obtained by the proposed framework *PRC* outperforms all baselines. This suggests that simply using basic features and advanced features do not offer meaningful representations for ranking paths. Our design on path representation that captures both road network topology and spatial properties is effective.

Table B.9: Comparison with Regression Baselines

	Method	MAE	MARE	τ	ρ
BF	**LR**	0.2640	0.4012	0.6879	0.7150
	Lasso	0.2876	0.4371	0.6245	0.6678
	SVR	0.2390	0.3632	0.6543	0.6683
	DT	0.2516	0.3824	0.6530	0.6777
	DTA	0.2686	0.4082	0.6784	0.7135
AF	**LR**	0.3430	0.5213	0.0864	0.0854
	Lasso	0.2955	0.4484	0.6260	0.6686
	SVR	0.3369	0.5120	0.0857	0.0846
	DT	0.4141	0.6284	0.0450	0.0693
	DTA	0.4301	0.6527	0.0812	0.0395
Deep Learning	**LSTM**	0.2682	0.4076	0.4569	0.4619
	PRC	**0.0611**	**0.0929**	**0.8178**	**0.8454**

6.4 Comparison with Driver Specific *PathRank*

We investigate if driver specific models are able to provide more accurate personalized ranking. We select two drivers with the largest amount training trajectories. Driver 1 has 2,068 trajectories and Driver 2 has 1,457 trajectories. We train two driver-specific *PRC* models, *PRC-Dr1* and *PRC-Dr2*, using only the trajectories from the corresponding driver. In addition, we consider a baseline *BA* from a personalized routing algorithm [56], which learns a 3-dimensional vector to combine the distance, travel time, and fuel consumption of a path to derive a personalized cost for the path. Then, the paths are ranked according to their personalized costs. The vector is learned from individual drivers' trajectories and thus different drivers have different vectors. *BR-Dr1* and *BR-Dr2* use trajectories from the corresponding drivers to learn the vectors.

We test the models on two testing sets *Dr*1 and *Dr*2 which consist of testing trajectories from Driver 1 and Driver 2, respectively. Table B.10 shows that for the testing trajectories from Driver 1, *PRC-Dr1* outperforms *PRC-Dr2*; and for the testing trajectories from Driver 2, *PRC-Dr2* outperforms *PRC-Dr1*. This is not surprising and this indicates that different drivers do have different preferences; and a ranking model trained on one driver may not provide accurate ranking for a different driver. In addition, *PRC-Dr1* outperforms *BA-Dr1* and *PRC-Dr2* outperforms *BA-Dr2*, indicating that the proposed *PathRank* outperforms the baseline. Note that *BA* ranks path according to the personalized costs but does not estimate the ranking scores. Thus, *BA* has no MAE and MARE values but only τ and ρ values.

The proposed *PRC* performs the best on both testing sets. This suggests

that the proposed method is able to learn a better ranking when using much more trajectories from different drivers. Together with the context embedding, it enables accurate personalized ranking.

Table B.10: Comparison with Driver Specific *PathRank*

Testing Data	Model	MAE	MARE	τ	ρ
Dr1	*PRC-Dr1*	0.1037	0.1532	0.8395	0.8531
	PRC-Dr2	0.2557	0.3975	0.6544	0.6419
	PR-A2-M3	0.0786	0.1162	0.8309	0.8513
	PRC	**0.0658**	**0.0972**	**0.8466**	**0.8710**
	BA-Dr1	—	—	0.7298	0.7392
Dr2	*PRC-Dr1*	0.1476	0.2182	0.7741	0.7851
	PRC-Dr2	0.1079	0.1677	0.8535	0.8750
	PR-A2-M3	0.0851	0.1323	0.8571	0.8900
	PRC	**0.0625**	**0.0971**	**0.8668**	**0.8945**
	BA-Dr2	—	—	0.7573	0.7800

Next, we report statistics on a case-by-case comparison, where Table B.11 shows the percentages of the cases where a driver specific *PathRank* outperforms *PRC*. Specifically, *PRC-Dr1* outperforms *PRC* in ca. 22% of the testing cases from Driver 1, and *PRC-Dr2* outperforms *PRC* in ca. 19% of the testing cases from Driver 2.

Table B.11: Percentage when *PR-Dr* Outperforms *PRC*

PR-Dr1		*PR-Dr2*	
τ	ρ	τ	ρ
22.70%	22.70%	19.31%	18.81%

The results from the above two tables suggest that user-specific *PathRank* models still have a potential to achieve personalized ranking, which may outperform the *PathRank* model trained on all trajectories, i.e., *PRC*. We plan to explore attention mechanisms on driver feature vectors to achieve this in future work.

6.5 Effects on Training Data Size

We conduct the next experiment to investigate the performance when varying the sizes of training data. Specifically, we use 25%, 50%, 75%, 100% of the total training data to train *PRC*, respectively. Based on the results shown in Table B.12, more training data gives better performance.

Table B.12: Effects of the Size of Training Data

Percentage	MAE	MARE	τ	ρ
25%	0.1071	0.1672	0.7574	0.7898
50%	0.0871	0.1323	0.7873	0.8179
75%	0.0892	0.1355	0.7928	0.8227
100%	**0.0611**	**0.0929**	**0.8178**	**0.8454**

6.6 Online Efficiency

Since ranking candidate paths is conducted online, we report the runtime. Table B.13 reports the runtime for estimating a path when using different *PathRank* models. It shows that the non-multi-task learning models, i.e., *PR-B*, *PR-A1*, and *PR-A2*, have similar run time. Multi-task learning models take longer time and the more auxiliary tasks are included in a model, the longer time the model takes. *PRC* takes the longest time, on average 58.2 ms. Suppose that an advanced routing algorithm or a commercial navigation system returns 10 candidate paths, *PRC* is able to return a ranking in 58.2 ms on average, which is within a reasonable response time.

Table B.13: Average Testing Runtime Per Path (ms)

PR-B	*PR-A1*	*PR-A2*	*PR-A2-M1*	*PR-A2-M2*	*PR-A2-M3*	*PRC*
11.4	11.3	11.5	22.8	34.4	45.1	58.2

6.7 Offline Training Efficiency

We study training efficiency by varying road network graph sizes and path lengths. First, we consider three road network graphs with different sizes in Table B.14. When the road network graph has 1 million vertices, each epoch takes 24.7 seconds, which is still within a reasonable time. Note that the main application scenario is intra-city, where multiple path candidates connecting the same source and destination exist. When traveling inter-cities, there are often very few path alternatives, e.g., using highways. Thus, a road network graph with 1 million vertices should already be able to model a very large city.

Table B.14: Effects of Graph Size for Training Time

# of Vertices	*50K*	*500K*	*1000K*
Run time per epoch (s)	8.3	12.1	24.7

Next, Table B.15 shows the training time when the training paths are with different numbers of vertices. For long paths with more vertices, the RNN needs to go through more GRU units. Thus, it takes longer time.

Table B.15: Effects of Path Lengths, Training Time

# of vertices per path	*60*	*120*	*180*	*240*
Run time per epoch (s)	7.8	12.6	21.3	24.8

7 Conclusion and Future Work

We propose a context-aware, multitask learning framework to rank paths in road networks. We propose an effective method to generate a compact and diverse set of training paths to enable efficient and effective learning. Then, we propose a multi-task learning framework to enable road network embedding that takes into account spatial properties. A recurrent neural network, together with the learned road network embedding, is employed to estimate the ranking scores which eventually enable ranking paths. In addition, a temporal graph is proposed to embed temporal contexts. Empirical studies conducted on a large real world trajectory set demonstrate that the proposed framework is effective and efficient for practical usage. As future work, it is of interest to exploit different means, e.g., attention mechanisms on path lengths and outlier trajectories removal [32, 33], to further improve the ranking quality of *PathRank*. It is also of interest to explore parallel computing [59] to improve efficiency.

References

[1] Q. Ai, K. Bi, J. Guo, and W. B. Croft, "Learning a deep listwise context model for ranking refinement," in *SIGIR*, 2018, pp. 135–144.

[2] G. Andonov and B. Yang, "Stochastic shortest path finding in path-centric uncertain road networks," in *MDM*, 2018, pp. 40–45.

[3] T. Anwar, C. Liu, H. L. Vu, and M. S. Islam, "Roadrank: Traffic diffusion and influence estimation in dynamic urban road networks," in *CIKM*, 2015, pp. 1671–1674.

[4] S. Cao, W. Lu, and Q. Xu, "Deep neural networks for learning graph representations," in *AAAI*, 2016, pp. 1145–1152.

[5] Z. Cao, T. Qin, T. Liu, M. Tsai, and H. Li, "Learning to rank: from pairwise approach to listwise approach," in *ICML*, 2007, pp. 129–136.

[6] V. Ceikute and C. S. Jensen, "Routing service quality - local driver behavior versus routing services," in *MDM*, 2013, pp. 97–106.

[7] T. Chen and S. Lu, "Accurate and efficient traffic sign detection using discriminative adaboost and support vector regression," *IEEE Trans. Vehicular Technology*, vol. 65, no. 6, pp. 4006–4015, 2016.

[8] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," in *EMNLP*, 2014, pp. 103–111.

[9] R. Cirstea, D. Micu, G. Muresan, C. Guo, and B. Yang, "Correlated time series forecasting using multi-task deep neural networks," in *CIKM*, 2018, pp. 1527–1530.

[10] R.-G. Cirstea, B. Yang, and C. Guo, "Graph attention recurrent neural networks for correlated time series forecasting." in *MileTS19@KDD*, 2019.

[11] J. Dai, B. Yang, C. Guo, and Z. Ding, "Personalized route recommendation using big trajectory data," in *ICDE*, 2015, pp. 543–554.

[12] Z. Ding, B. Yang, Y. Chi, and L. Guo, "Enabling smart transportation systems: A parallel spatio-temporal database approach," *IEEE Trans. Computers*, vol. 65, no. 5, pp. 1377–1391, 2016.

[13] D. Erhan, Y. Bengio, A. C. Courville, P. Manzagol, P. Vincent, and S. Bengio, "Why does unsupervised pre-training help deep learning?" *J. Mach. Learn. Res.*, vol. 11, pp. 625–660, 2010.

[14] E. Erkut and V. Verter, "Modeling of transport risk for hazardous materials," *Operations research*, vol. 46, no. 5, pp. 625–642, 1998.

[15] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *KDD*, 2016, pp. 855–864.

[16] C. Guo, C. S. Jensen, and B. Yang, "Towards total traffic awareness," *SIGMOD Record*, vol. 43, no. 3, pp. 18–23, 2014.

[17] C. Guo, Y. Ma, B. Yang, C. S. Jensen, and M. Kaul, "Ecomark: evaluating models of vehicular environmental impact," in *SIGSPATIAL*, 2012, pp. 269–278.

[18] C. Guo, B. Yang, O. Andersen, C. S. Jensen, and K. Torp, "Ecomark 2.0: empowering eco-routing with vehicular environmental models and actual vehicle fuel consumption data," *GeoInformatica*, vol. 19, no. 3, pp. 567–599, 2015.

[19] C. Guo, B. Yang, J. Hu, and C. S. Jensen, "Learning to route with sparse trajectory sets," in *ICDE*, 2018, pp. 1073–1084.

[20] C. Guo, B. Yang, J. Hu, C. S. Jensen, and L. Chen, "Context-aware, preference-based vehicle routing." *VLDB Journal*, online first, 2020.

[21] J. Hershberger, M. Maxel, and S. Suri, "Finding the k shortest simple paths: A new algorithm and its implementation," *ACM Trans. Algorithms*, vol. 3, no. 4, p. 45, 2007.

[22] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[23] J. Hu, C. Guo, B. Yang, and C. S. Jensen, "Stochastic weight completion for road networks using graph convolutional networks," in *ICDE*, 2019, pp. 1274–1285.

[24] J. Hu, C. Guo, B. Yang, C. S. Jensen, and H. Xiong, "Stochastic origin-destination matrix forecasting using dual-stage graph convolutional, recurrent neural networks," in *ICDE*, 2020, pp. 1417–1428.

[25] J. Hu, B. Yang, C. Guo, and C. S. Jensen, "Risk-aware path selection with time-varying, uncertain travel costs: a time series approach," *VLDB J.*, vol. 27, no. 2, pp. 179–200, 2018.

[26] J. Hu, B. Yang, C. S. Jensen, and Y. Ma, "Enabling time-dependent uncertain eco-weights for road networks," *GeoInformatica*, vol. 21, no. 1, pp. 57–88, 2017.

[27] Z. Hu, Y. Wang, Q. Peng, and H. Li, "Unbiased lambdamart: An unbiased pairwise learning-to-rank algorithm," in *WWW*, 2019, pp. 2830–2836.

[28] M. Hua and J. Pei, "Probabilistic path queries in road networks: traffic uncertainty aware path selection," in *EDBT*, 2010, pp. 347–358.

[29] T. Joachims, "Optimizing search engines using clickthrough data," in *KDD*, 2002, pp. 133–142.

[30] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *NIPS*, 2017, pp. 3146–3154.

[31] T. Kieu, B. Yang, C. Guo, and C. S. Jensen, "Distinguishing trajectories from different drivers using incompletely labeled trajectories," in *CIKM*, 2018, pp. 863–872.

[32] ——, "Outlier detection for time series with recurrent autoencoder ensembles," in *IJCAI*, 2019, pp. 2725–2732.

[33] T. Kieu, B. Yang, and C. S. Jensen, "Outlier detection for multidimensional time series using deep neural networks," in *MDM*, 2018, pp. 125–134.

[34] S. Y. Kim and A. Upneja, "Predicting restaurant financial distress using decision tree and adaboosted decision tree models," *Economic Modelling*, vol. 36, pp. 354–362, 2014.

[35] Y. Lei, W. Li, Z. Lu, and M. Zhao, "Alternating pointwise-pairwise learning for personalized item ranking," in *CIKM*, 2017, pp. 2155–2158.

[36] Y. Li, K. Fu, Z. Wang, C. Shahabi, J. Ye, and Y. Liu, "Multi-task representation learning for travel time estimation," in *KDD*, 2018, pp. 1695–1704.

[37] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting," in *ICLR*, 2018.

[38] H. Liu, C. Jin, B. Yang, and A. Zhou, "Finding top-k optimal sequenced routes," in *ICDE*, 2018, pp. 569–580.

[39] ——, "Finding top-k shortest paths with diversity," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 3, pp. 488–502, 2018.

[40] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *ICLR*, 2013.

[41] K. Mouratidis, Y. Lin, and M. L. Yiu, "Preference queries in large multi-cost transportation networks," in *ICDE*, 2010, pp. 533–544.

[42] P. Newson and J. Krumm, "Hidden markov map matching through noise and sparseness," in *SIGSPATIAL*, 2009, pp. 336–343.

[43] Z. Pan, Y. Liang, W. Wang, Y. Yu, Y. Zheng, and J. Zhang, "Urban traffic prediction from spatio-temporal data using deep meta learning," in *KDD*, 2019, pp. 1720–1730.

[44] S. A. Pedersen, B. Yang, and C. S. Jensen, "Anytime stochastic routing with hybrid learning," *PVLDB*, vol. 13, no. 9, pp. 1555–1567.

[45] ——, "Fast stochastic routing under time-varying uncertainty." *VLDB Journal*, to appear, 2019.

[46] ——, "A hybrid learning approach to stochastic routing," in *ICDE*, to appear, 2020.

[47] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: online learning of social representations," in *KDD*, 2014, pp. 701–710.

[48] Y. Shi, J. Li, and Z. Li, "Gradient boosting with piece-wise linear regression trees," *arXiv preprint arXiv:1802.05640*, 2018.

[49] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *WWW*, 2015, pp. 1067–1077.

[50] H. Wang, J. Wang, J. Wang, M. Zhao, W. Zhang, F. Zhang, X. Xie, and M. Guo, "Graphgan: Graph representation learning with generative adversarial nets," *CoRR*, vol. abs/1711.08267, 2017.

[51] S. Wang, B. Ji, J. Zhao, W. Liu, and T. Xu, "Predicting ship fuel consumption based on lasso regression," *Transportation Research Part D: Transport and Environment*, vol. 65, pp. 817–824, 2018.

[52] J. Won, S. Kim, J. Baek, and J. Lee, "Trajectory clustering in road network environment," in *CIDM*, 2009, pp. 299–305.

[53] B. Yang, J. Dai, C. Guo, C. S. Jensen, and J. Hu, "PACE: a path-centric paradigm for stochastic path finding," *VLDB J.*, vol. 27, no. 2, pp. 153–178, 2018.

[54] B. Yang, C. Guo, and C. S. Jensen, "Travel cost inference from sparse, spatio-temporally correlated time series using markov models," *PVLDB*, vol. 6, no. 9, pp. 769–780, 2013.

[55] B. Yang, C. Guo, C. S. Jensen, M. Kaul, and S. Shang, "Stochastic skyline route planning under time-varying uncertainty," in *ICDE*, 2014, pp. 136–147.

[56] B. Yang, C. Guo, Y. Ma, and C. S. Jensen, "Toward personalized, context-aware routing," *VLDB Journal*, vol. 24, no. 2, pp. 297–318, 2015.

[57] S. B. Yang and B. Yang, "Learning to rank paths in spatial networks," in *ICDE*, 2020, pp. 2006–2009.

[58] J. Y. Yen, "Finding the k shortest loopless paths in a network," *management Science*, vol. 17, no. 11, pp. 712–716, 1971.

[59] P. Yuan, C. Sha, X. Wang, B. Yang, A. Zhou, and S. Yang, "XML structural similarity search using mapreduce," in *WAIM*, 2010, pp. 169–181.

Paper C

Unsupervised Path Representation Learning with Curriculum Negative Sampling

Sean Bin Yang, Chenjuan Guo, Jilin Hu, Jian Tang, and Bin Yang

The paper has been published in the
30th International Joint Conference on Artificial Intelligence (IJCAI-21),
pp. 3286-3292, 2021.

Abstract

*Path representations are critical in a variety of transportation applications, such as estimating path ranking in path recommendation systems and estimating path travel time in navigation systems. Existing studies often learn task-specific path representations in a supervised manner, which require a large amount of labeled training data and generalize poorly to other tasks. We propose an unsupervised learning framework Path InfoMax (*PIM*) to learn generic path representations that work for different downstream tasks. We first propose a curriculum negative sampling method, for each input path, to generate a small amount of negative paths, by following the principles of curriculum learning. Next,* PIM *employs mutual information maximization to learn path representations from both a global and a local view. In the global view,* PIM *distinguishes the representations of the input paths from those of the negative paths. In the local view,* PIM *distinguishes the input path representations from the representations of the nodes that appear only in the negative paths. This enables the learned path representations encode both global and local information at different scales. Extensive experiments on two downstream tasks, ranking score estimation and travel time estimation, using two road network datasets suggest that* PIM *significantly outperforms other unsupervised methods and is also able to be used as a pre-training method to enhance supervised path representation learning.*

1 Introduction

Path representations are crucial for various transportation applications, e.g., travel cost estimation [12, 18], routing [8, 19], path recommendation [7, 28], and traffic analysis [3, 11]. Path representation learning (*PRL*) aims to obtain distinguishable path representations for different paths in a transportation network and hence facilitating various downstream applications. Existing studies on *PRL* often learn path representations in a supervised manner, which has two limitations. First, they require a large amount of labelled training data. Second, the learned path representations are task-specific, e.g., working well for the task with labels, but generalize poorly to other tasks. The two limitations restrict supervised path representation learning from broader usage, thus calling for unsupervised path representation learning.

Although unsupervised graph representation learning methods exist, they are not designed to capture representations of paths. Node representation learning [6, 22] learns representations for individual nodes in a graph but does not consider paths, i.e., sequences of nodes. Simply aggregating the node representations of the nodes in a path fails to capture the sequential information in paths. Whole graph representation learning [21] learns representations for different graphs, while path representation learning considers different paths from the same graph. In addition, unsupervised graph representation

learning often utilize random negative sampling to enable training, which is ineffective for path representation learning.

We propose an unsupervised path representation learning framework Path InfoMax (*PIM*), including a curriculum negative sampling method and a path representation learning method. First, we propose a *curriculum negative sampling strategy* to generate a small number of negative paths for an input path. Instead of randomly select other input paths as negative paths, the strategy follows the principles of curriculum learning [1] to first generate paths that are largely different from the input path and thus are easy to be distinguished from the input path. Then, we gradually generate paths that are increasingly similar to the input path and thus are more difficult to be distinguished from the input path. The proposed curriculum negative sampling facilitates effective learning of distinguishable path representations.

Next, we propose two different discriminators, a *path-path discriminator* and a *path-node discriminator*, to jointly learn path representations. The path-path discriminator captures the representation differences between an input path and its negative paths, which we refer to as a *global* view. The path-node discriminator captures the representation difference between an input path and the representations of the nodes that only appear in its negative paths, which we refer to as a *local* view. The two discriminators ensure the quality of the learned path representations, because they are distinguishable from not only the representations of negative paths from a global view but also the representations of the nodes in negative paths from a local view. To the best of our knowledge, *PIM* is the first work that studies unsupervised path representation learning. We make the following contributions.

1. We propose a curriculum negative sampling strategy for path representation learning.
2. We propose the path-path and path-node discriminators to learn jointly path representations from a global and a local view.
3. We conduct extensive experiments on two data sets with two downstream tasks to demonstrate the effectiveness of *PIM*.

2 Related Work

Path Representation Learning. Existing proposals on path representation learning are all under the supervised learning setting. Such proposals often require large amount of labeled training data and the learned path representations cannot be easily reused for other tasks. For example, *Deepcas* [15], *ProxEmbed* [17], and *PathRank* [27, 28] employ different kinds of RNNs to

combine node representations of the nodes in a path to obtain a path representation. Then, the training is performed in an end-to-end fashion by using the labeled training data. Instead, we propose an unsupervised path representation learning framework *PIM* that does not require labeled training data and it generalizes nicely to multiple downstream tasks (cf. Table C.1 in Section 5.2). In addition, *PIM* can be used as a pre-training method to enhance existing supervised path representation learning (cf. Figure D.7 in Section 5.2). An unsupervised trajectory representation learning method transforms trajectories into images and thus do not apply on paths in graphs [13].

Mutual Information Maximization on Graphs. Motivated by Deep InfoMax [9], mutual information maximization has been applied for unsupervised graph representation learning. Deep Graph Infomax (*DGI*) [24] and Graph Mutual Information (*GMI*) [20] learn node representations and *InfoGraph* [21] learns whole graph representations. Here, negative samples are often randomly drawn from a different graph and the mutual information only considers a local view, e.g., a node representation vs. a graph representation. In *PIM*, we propose a curriculum negative sample strategy to generate negative paths with different overlapping nodes with the input paths from the same graph, which facilitates training. Other advanced negative sampling approaches exist [5, 25], but they are not proposed for graphs and do not follow curriculum learning. In addition, we compute mutual information on both a local view (i.e., the representations of input paths vs. the node representations of negative paths) and a global view (i.e., the representations of input paths vs. negative paths) and use them jointly to train the model, which improves accuracy.

3 Preliminaries

Graph. We consider a directed graph $G = (\mathbb{V}, \mathbb{E})$, where $\mathbb{V}$ is the node set and $\mathbb{E}$ is the edge set and we have $|\mathbb{V}| = N$ and $|\mathbb{E}| = M$. Each node $V_i \in \mathbb{V}$ is associated with a node feature vector $v_i \in \mathbb{R}^D$.

Path. A path $P = \langle V_1, V_2, \ldots, V_Z \rangle$ is a sequence of nodes, where Z is the path length and $P.s = V_1$ and $P.d = V_Z$ are the source and destination of path P, respectively. Each pair of adjacent nodes (V_k, V_{k+1}) is connected by an edge in $\mathbb{E}$, $1 \leq k < Z$. We use $IV(P) \in \mathbb{R}^{Z \times D}$ to represent the concatenation of the node feature vectors of the nodes in path P. We call $IV(P_i)$ the initial view of path P_i.

Problem Definition. Given a set of path $\mathbb{P}$ in graph G, *Path Representation Learning (PRL)* aims at learning a path representation vector $p_i \in \mathbb{R}^{D'}$ for each path $P_i \in \mathbb{P}$. Formally, *PRL* learns a path encoder PE_ψ that takes as input the initial view $IV(P_i)$ of path P_i, i.e., the node features of the nodes in path P_i, and outputs its path representation vector p_i.

$$PE_\psi : \mathbb{R}^{Z \times D} \rightarrow \mathbb{R}^{D'}, \tag{C.1}$$

where ψ indicates the learnable parameters for the path encoder, e.g., weights in a neural network, Z is the length of path P_i, and $D' \ll Z \times D$ is an integer indicating the dimension of the learned path representation vector p_i.

The learned path representation vectors are supposed to support a variety of downstream tasks, e.g., path ranking and path travel time estimation.

4 Path InfoMax

Figure C.1 offers an overview of the proposed framework Path InfoMax (*PIM*).

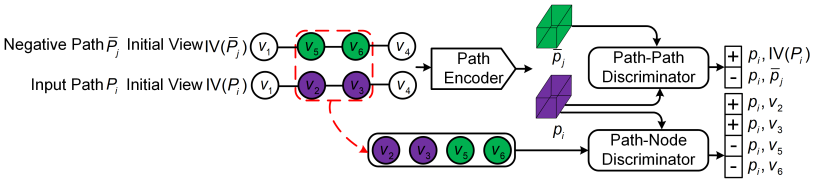


Figure C.1: *PIM* Overview. The Path Encoder takes as input the initial view $IV(P_i)$ of input path P_i and the initial view $IV(\bar{P}_j)$ of negative path $\bar{P}_j$, and returns their representations p_i and $\bar{p}_j$, respectively. The Path-Path Discriminator takes as input a pair of path representations and decides whether they are from the same path. A positive pair, e.g., is$(p_i, IV(P_i))$, refers to two different representation views of the same input path P_i. A negative pair, e.g., $(p_i, \bar{p}_j)$, refers to the path representations of an input path vs. its negative path. The Path-Node Discriminator takes as input a (input path representation, node feature vector) pair and decides whether the node is from the input path. A positive pair, e.g., (p_i, v_2), represents the path representation of P_i and a node feature vector of node v_2 that only appears in P_i. A negative pair, e.g., (p_i, v_5), represents the path representation of the input path and a node feature vector of node v_5 that only appears in the negative path.

PIM employs contrastive learning, specifically mutual information maximization, to train the path encoder to produce path representations without requiring task-specific labels.

The path encoder takes as input the initial view of an input path and outputs its path representation (cf. Sec. 4.1). Training the path encoder is supported by a path-path discriminator and a path-node discriminator using negative samples. To this end, we first introduce the curriculum negative sampling strategy to generate negative paths (cf. Sec. 4.2). Then, the path-path discriminator guides the path encoder to produce path representations such

that the path representations of input paths can be distinguished from the path representations of negative paths (cf. Sec. 4.3). In addition, the path-node discriminator guides the path encoder to produce path representations such that the path representations of input paths can be distinguished from the node features of the nodes that only appear in the negative paths (cf. Sec. 4.4). Finally, we discuss the final training objectives of *PIM*.

4.1 Path Encoder

Since a path consists of a sequence of nodes, we use a model that is able to encode sequential data, e.g., a recurrent neural network [2, 10] or a Transformer [23] as the path encoder PE_ψ, where ψ represents the parameters to be learned for the path encoder.

Given a path $P_i = \langle V_1, V_2, \ldots, V_Z \rangle$, we use its initial view $IV(P_i) \in \mathbb{R}^{Z \times D}$ as the input to the path encoder, which returns its path representation vector $p_i \in \mathbb{R}^{D'}$.

4.2 Curriculum Negative Sampling

Motivated by curriculum learning [1], we propose a curriculum negative sampling method to generate negative samples. The idea behind curriculum learning is that we start to train a model with easier samples first, and then gradually increase the difficulty levels. In our setting, we first generate negative paths that are different from the input path, e.g., paths without any overlapping nodes with the input path. In this case, it can be easy to train a path encoder that returns distinguishable representations of the input path and the negative paths. Then, we gradually generate negative paths that are increasingly similar to the input path, e.g., sharing the same source and destination with the input path and with increasingly overlapping nodes. This makes more difficult for the path encoder to generate distinguishable path representations. Figure C.2 shows three negative paths $\bar{P}_1$, $\bar{P}_2$, and $\bar{P}_3$ with increasingly difficulties for input path P_1, along with the underlying road network graph.

Specifically, for each input path P_1, we first randomly select a path from the path set $\mathbb{P}$ as the first negative path. Next, we use the source and the destination of P_1 as the input to call the top-k diversified shortest path algorithm [16] to generate paths that share the same source and destination of P_1. This algorithm allows us to set different diversity thresholds, enabling us to generate negative paths with different overlapping nodes with the input path.

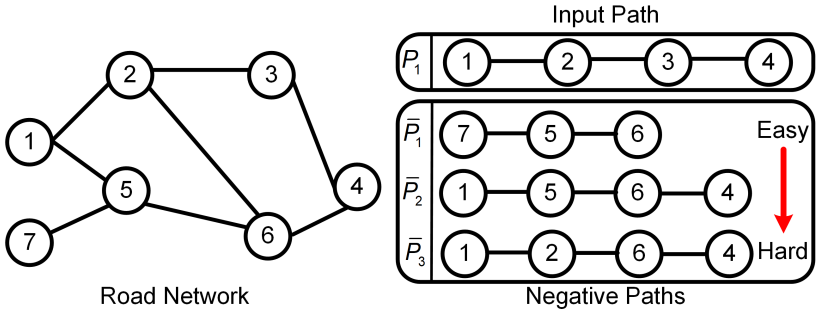


Figure C.2: Curriculum Negative Sampling.

4.3 Global Mutual Information Maximization

We proceed to the learning of the path encoder using the negative paths. We first consider a global view of the path representations. We expect that the learned path representations are distinguishable from the path representations of the negative paths.

To this end, we first construct negative and positive pairs for training a *path-path discriminator* $D^{PP}_{\omega_1}$. In a negative pair $\langle (p_i, \bar{p}_j), - \rangle$, p_i and $\bar{p}_j$ represent the path representations of input path P_i and a negative path $\bar{P}_j$, respectively, which are both returned by the path encoder PE_ψ. In a positive pair $\langle (p_i, IV(P_i), + \rangle$, p_i is still the path representations of input path P_i returned by the path encoder and $IV(P_i)$ is the initial view of path P_i (cf. Section 4.1). Here, p_i and $IV(P_i)$ represent two different views, i.e., a view from the path encoder vs. a view from the node features, of the same input path P_i. Figure C.1 shows examples of a negative and a positive pair.

Next, we use mutual information maximization to train the path-path discriminator $D^{PP}_{\omega_1}$ such that it is able to make a binary classification on the negative vs. positive pairs. Specifically, we aim at maximizing the estimated mutual information (MI) over the positive and negative pairs.

$$\underset{\psi,\omega_1}{\operatorname{argmax}} \sum_{P_i \in \mathbb{P}} I_{\psi,\omega_1}(p_i, \mathbb{NP}_i),$$

where $I_{\psi,\omega_1}(\cdot,\cdot)$ is the MI estimator modeled by the path-path discriminator $D^{PP}_{\omega_1}$ that is parameterized by parameters ω_1 and the path encoder PE_ψ that is parameterized by parameters ψ. Path P_i is an input path from $\mathbb{P}$ and p_i is its path representation returned by the path encoder. $\mathbb{NP}_i$ includes the negative paths of P_i. Following [9, 24], we use a noise-contrastive type objective with a standard binary cross-entropy loss on the positive pairs and the negative

pairs, as shown in Equation C.2.

$$\mathcal{I}_{\psi,\omega_1}(p_i, \mathbb{NP}_i) := \frac{1}{1+|\mathbb{NP}_i|}(\mathbb{E}_{\mathbb{P}}\left[\log D^{PP}_{\omega_1}(p_i, IV(P_i))\right] + \sum_{\bar{p}_j \in \mathbb{NP}_i} \mathbb{E}_{\mathbb{NP}_i}\left[\log\left(1 - D^{PP}_{\omega_1}\left(p_i, \bar{p}_j\right)\right)\right]) \tag{C.2}$$

Here, we use $\mathbb{E}_{\mathbb{P}}$ and $\mathbb{E}_{\mathbb{NP}_i}$ to denote expectations w.r.t. the empirical probability distribution of the input paths and the negative paths. Note that p_i and $\bar{p}_j$ are the path representations returned by the path encoder PE_ψ. Thus, maximizing the MI estimator enables the training of both the path encoder (i.e., parameters ψ) and the path-path discriminator (i.e., parameters ω_1).

4.4 Local Mutual Information Maximization

We proceed to consider a local view of the path representations. We expect that the learned path representations are distinguishable from the node feature vectors of the nodes from input vs. negative paths. This is particularly important when distinguishing two paths with significant overlapping nodes. We introduce a positive node set $\mathbb{X}_i$ that includes nodes appearing only in the input path P_i but not the negative paths and a negative node set $\mathbb{Y}_i$ that includes nodes appearing only in the negative paths but not the input path P_i. We then construct negative and positive pairs for training a *path-node discriminator* $D^{PN}_{\omega_2}$. In a negative pair $\langle (p_i, v_j), - \rangle$, p_i represents the path representations of input path P_i, returned by the path encoder PE_ψ; v_j represents the node feature vector of a negative node $V_j \in \mathbb{Y}_i$. Similarly, in a positive pair $\langle (p_i, v_k), + \rangle$, v_k represents the node feature vector of a positive node $V_k \in \mathbb{X}_i$. Figure C.1 shows examples of two negative and two positive such pairs for the path-node discriminator.

Similar to the path-path discriminator training, we also employ mutual information maximization to train the path-node discriminator $D^{PN}_{\omega_2}$. In particular, we have

$$\underset{\psi,\omega_2}{\operatorname{argmax}} \sum_{P_i \in \mathbb{P}} I_{\psi,\omega_2}(p_i, \mathbb{X}_i \cup \mathbb{Y}_i),$$

where I_{ψ,ω_2} is the MI estimator modeled by the path-node discriminator $D^{PN}_{\omega_2}$ that is parameterized by parameters ω_2 and the path encoder PE_ψ that is parameterized by parameters ψ. We use a noise-contrastive with a BCE loss, similar to Equation C.2, to compute $I_{\psi,\omega_2}(p_i, \mathbb{X} \cup \mathbb{Y})$ as follows.

$$I_{\psi,\omega_2}(p_i, \mathbb{X}_i \cup \mathbb{Y}_i) := \frac{1}{|\mathbb{X}_i \cup \mathbb{Y}_i|}(\sum_{v_k \in \mathbb{X}_i} \mathbb{E}_{\mathbb{X}_i}\left[\log \mathcal{D}^{PN}_{\omega_2}(p_i, v_k)\right] + \sum_{v_j \in \mathbb{Y}_i} \mathbb{E}_{\mathbb{Y}_i}\left[\log\left(1 - D^{PN}_{\omega_2}\left(p_i, v_j\right)\right)\right]) \tag{C.3}$$

4.5 Maximization of *PIM*

We combine both the global and local mutual information maximization when training the final *PIM* model, see below.

$$\underset{\psi,\omega_1,\omega_2}{\operatorname{argmax}} \sum_{P_i \in \mathbb{P}} \left(I_{\psi,\omega_1}(p_i, \mathbb{NP}_i) + I_{\psi,\omega_2}(p_i, \mathbb{X}_i \cup \mathbb{Y}_i) \right).$$

5 Experiments

We conduct experiments to investigate the effectiveness of the proposed unsupervised path representation learning framework *PIM* on two downstream tasks using two data sets. In addition, we also demonstrate that *PIM* is able to use as a pre-training method to enhance supervised path representation learning.

5.1 Experimental Setup

Road Network and Paths

We obtain two road network graphs from OpenStreetMap. The first is from Aalborg, Denmark, consisting of 8,893 nodes and 10,045 edges. The second is from Harbin, China, consisting of 5,796 nodes and 8,498 edges. We also obtain two substantial GPS trajectory data sets on the two road networks. We consider 52,494 paths in the Aalborg network and 37,079 paths in the Harbin network.

Downstream Tasks

Path Travel Time Estimation. Each path is associated with its travel time (seconds) obtained from trajectories. We aim at building a regression model to estimate the travel time of paths. We evaluate the accuracy of the estimations by Mean Absolute Error (**MAE**), Mean Absolute Relative Error (**MARE**) and Mean Absolute Percentage Error (**MAPE**).

Path Ranking. Given a set of paths, which often share the same source and destination, each path is associated with a ranking score in range $[0, 1]$. The ranking scores are obtained with the help of trajectories by following an existing study [27]. In path ranking, we aim at building a regression model to estimate the ranking scores of the paths. To evaluate the accuracy of the estimated ranking scores, we not only report the **MAE** of the estimated ranking scores but also use Kendall rank correlation coefficient (denoted by τ) and Spearman's rank correlation coefficient (denoted by ρ) to measure the

consistency between the ranking derived by the estimated ranking scores vs. the ranking derived by the ground truth ranking scores. Smaller **MAE** and higher τ and ρ values indicate higher accuracy.

Baselines

We compare *PIM* with seven baseline methods.

- **Node2vec** [6], Deep Graph InfoMax (**DGI**) [24], Graphical Mutual Information Maximization (**GMI**) [20] are three unsupervised node representation learning models, which output the node representation for each node in a graph. We use the average of the node representations of the nodes in a path to get the path representation of the path. We also consider using concatenation instead of average, but resulting worse accuracy.
- **Memory Bank (MB)** [26] is an unsupervised learning approach to obtain representations from high-dimensional data. It uses a memory bank to achieve the negative samples from current batch to train an encoder, then gets the representation based on contrastive loss. We re-implement **MB** with an LSTM encoder to better capture the sequential information to get the path representations.
- **InfoGraph** [21] is an unsupervised whole graph representation learning model. Here, we treat a path as a graph to learn the path representation.
- **BERT** [4] is an unsupervised language representation learning model. To enable training, we (1) treat a path as a sentence and mask some nodes in the path; and (2) split a path P into two sub-paths P_1 and P_2, and consider (P_1, P_2) as a valid Q&A pair and (P_2, P_1) as an invalid Q&A pair because the former keeps a meaningful order while the latter does not.
- **PathRank** [27] is a *supervised* path representation learning model based on GRU. **PathRank** takes into account the labels from a specific downstream task to obtain path representations.

Among these baselines, *Node2vec*, *DGI*, *GMI*, *MB*, *InfoGraph*, and *BERT* are *unsupervised learning* approaches, which do not employ labels from specific downstream tasks to produce path representations. In contrast, *PathRank* is a *supervised learning* approach, where it employ labels from specific downstream tasks to produce path representations, meaning that the obtained path representations are different when using labels from different downstream tasks.

Regression Model

For all unsupervised learning approaches, we first obtain a task-independent path representation and then apply a regression model to solve different downstream tasks using task-specific labels. In the experiments, we choose *Gaussian Process Regressor* (*GPR*) to make travel time and ranking score estimation. We randomly choose 85%, 10%, and 5% of the paths as the training, validation, and test sets.

Implementation Details

We use an LSTM as the path encoder. We use *node2vec* [6], an unsupervised node representation learning method, to obtain a 128 dimensional node feature vector for each node, i.e., $D = 128$. We set the path representation size $D' = 128$. In the curriculum negative sampling, for each input path, we generate four negative paths—the first two paths are randomly selected from $\mathbb{P}$ and the third and the fourth paths are two paths returned by the top-k diversified shortest paths with different overlapping nodes with the input path. We use Adam [14] for optimization with learning rate of 0.001. All algorithms are implemented in Pytorch 1.7.1. We conduct experiments on Ubuntu 18.04.5 LTS, with 40 Intel(R) Xeon(R) Gold 5215 CPUs @ 2.50GHz and four Quadro RTX 8000 GPU cards. Code is available at `https://github.com/Sean-Bin-Yang/Path-InfoMax.git`.

5.2 Experimental Results

Overall accuracy on both downstream tasks

Table C.1 shows the results on travel time and ranking score estimation. *PIM* consistently outperforms all baselines on both tasks and on both data sets. *Node2vec*, *DGI*, and *GMI* fail to capture the dependencies among node feature vectors in paths. In contrast, *PIM* considers such dependencies by using the LSTM based path encoder. In addition, the two discriminators further improve the accuracy.

InfoGraph implicitly considers node feature vector sequences. However, the discriminator in *InfoGraph* only considers the local view. In addition, *InfoGraph* considers other paths in the same batch as negative samples, whereas *PIM* employs curriculum negative sampling to generate negative samples. *PIM* outperforms *InfoGraph* suggests that the proposed curriculum negative sampling and jointly consider both local and global views are effective.

Although *MB* and *BERT* also capture dependencies among the node feature vectors in paths, such methods only achieve relatively poor accuracy. This is because *MB* often requires large amount of negative samples (e.g., more than 256), which is not feasible in our setting. Although the unsupervised training

Method	Aalborg						Harbin					
	Travel Time Estimation			Path Ranking			Travel Time Estimation			Path Ranking		
	MAE	MARE	MAPE	MAE	τ	ρ	MAE	MARE	MAPE	MAE	τ	ρ
Node2vec	121.43	0.27	31.04	0.18	0.66	0.70	258.91	0.22	23.17	0.15	0.70	0.72
DGI	192.63	0.42	82.44	0.54	0.49	0.52	528.71	0.39	86.53	0.21	0.59	0.60
GMI	136.58	0.30	50.81	0.23	0.58	0.61	979.68	0.73	192.45	0.24	0.55	0.56
MB	243.97	0.53	84.17	0.35	0.34	0.38	533.41	0.40	86.01	0.27	0.31	0.34
BERT	254.17	0.54	61.61	0.36	0.38	0.39	514.95	0.57	49.80	0.28	0.45	0.46
InfoGraph	132.28	0.29	39.47	0.17	0.69	0.73	391.45	0.44	44.60	0.29	0.68	0.72
PIM	**76.10**	**0.16**	**17.28**	**0.12**	**0.72**	**0.76**	**125.76**	**0.14**	**13.73**	**0.11**	**0.75**	**0.79**

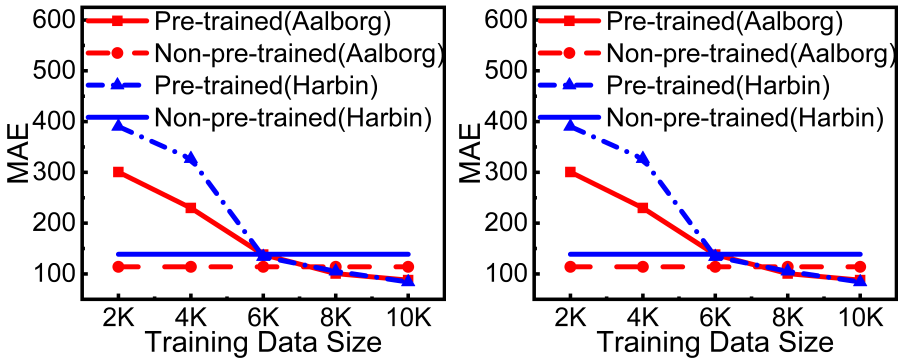
Table C.1: Overall Accuracy on Travel Time Estimation and Ranking Score Estimation.

strategy in *BERT* works well for NLP, it does not fit our problem setting on learning path representations.

Using *PIM* as a Pre-training Method

In this experiment, we consider *PIM* as a pre-training method for the supervised method *PathRank*. *PathRank* employs an GRU that takes as input node feature vectors in a path and predicts travel time or ranking scores. To use *PIM* as a pre-training method for *PathRank*, we use a GRU based path encoder. Then, we first train *PIM* in an unsupervsied manner, and then use the learned parameters in the GRU path encoder to initialize the GRU in *PathRank*. Finally, we use the labelled training paths to fine tune *PathRank*.

Figure D.7 shows the travel time estimation performance of *PathRank* with vs. without pre-training on both data sets. When not using pre-training, we train *PathRank* using 10K labelled training paths. We observe that: (1) when using pre-training, we are able to achieve the same accuracy of the non-pre-training *PathRank* using less labelled training paths, e.g., ca. 7K for Aalborg and 6K for Harbin. (2) when using 10K labelled training paths, the pre-training *PathRank* achieves higher accuracy than the non-pre-training *PathRank*. We observe similar results on the other task of path ranking, suggesting that *PIM* can be used as a pre-training method to enhance supervised methods.



(a) Travel Time Estimation. **(b)** Path Ranking.

Figure C.3: Effects of Pre-training.

Impact of Local and Global MI Maximization

We investigate the impact of jointly using both path-path and path-node discriminators to consider both the local and global MI maximization. We consider two variants of *PIM* where (1) we only use the path-path discriminator to maximize the global MI and (2) we only use the path-node discriminator to

	Travel Time Estimation			Path Ranking		
	MAE	**MARE**	**MAPE**	**MAE**	τ	ρ
Global	237.92	0.51	85.88	0.34	0.22	0.25
Local	118.03	0.25	26.20	0.14	0.70	0.74
Joint	**76.10**	**0.16**	**17.28**	**0.12**	**0.72**	**0.76**

Table C.2: Effects of Local and Global MI Maximization, Aalborg.

	Travel Time Estimation			Path Ranking		
	MAE	**MARE**	**MAPE**	**MAE**	τ	ρ
Rand.	101.16	0.22	23.51	0.14	0.65	0.69
Top-k	100.87	0.22	22.31	0.13	0.72	0.75
Curr.	**76.10**	**0.16**	**17.28**	**0.12**	**0.72**	**0.76**

Table C.3: Effects of Curriculum Negative Sample Strategy, Aalborg.

maximize the local MI. Table C.2 shows that jointly maximizing both the local and global MI achieves the best accuracy, which justifies our design choices of using both the path-path and path-node discriminators.

Impact of Curriculum Negative Sample Strategy

To investigate the effectiveness of the proposed curriculum negative sample strategy, we compare it with the following two strategies.

1. Random only: it randomly selects paths from $\mathbb{P}$.
2. Top-k only: it employs the top-k diversified shortest path algorithms to generate negative paths sharing the same origin and destination with the input path with different overlapping nodes.

To make a fair comparison, we use each strategy to generate the same number of negative paths, i.e., 4. Table C.3 shows that the top-k only strategy is better than random only, suggesting that it is important to distinguish the representations of input paths vs. paths sharing the same origin and destination. The proposed curriculum negative sampling strategy achieves the best accuracy, suggesting that training *PIM* from easy to hard negative paths help further improves accuracy.

Impact of Negative Path Numbers

We investigate the impact of using different numbers of negative paths. We vary the number of negative paths K from 1, 2, 3, to 4. Recall that when using curriculum negative sampling, the first two paths are from $\mathbb{P}$ and the last two

	Travel Time Estimation			Path Ranking		
	MAE	**MARE**	**MAPE**	**MAE**	τ	ρ
K=1	119.77	0.29	32.91	0.19	0.58	0.63
K=2	107.46	0.26	29.22	0.18	0.59	0.63
K=3	87.58	0.19	20.00	0.12	0.71	0.74
K=4	**76.10**	**0.16**	**17.28**	**0.12**	**0.72**	**0.76**

Table C.4: Effects of Negative Path Numbers, Aalborg.

Posi.	**Travel Time Estimation**			**Path Ranking**		
Nods.	**MAE**	**MARE**	**MAPE**	**MAE**	τ	ρ
20%	114.31	0.25	24.92	0.20	0.65	0.70
40%	111.33	0.24	24.08	0.16	0.66	0.70
60%	104.57	0.23	22.94	0.14	0.68	0.71
80%	101.31	0.23	22.56	0.13	0.68	0.72
100%	**76.10**	**0.16**	**17.28**	**0.12**	**0.72**	**0.76**
Neg.	**Travel Time Estimation**			**Path Ranking**		
Nods.	**MAE**	**MARE**	**MAPE**	**MAE**	τ	ρ
20%	130.90	0.29	28.21	0.19	0.60	0.65
40%	110.86	0.24	25.30	0.15	0.67	0.70
60%	105.70	0.23	24.01	0.13	0.67	0.71
80%	102.80	0.22	23.35	0.13	0.68	0.72
100%	**76.10**	**0.16**	**17.28**	**0.12**	**0.72**	**0.76**

Table C.5: Effects of Positive / Negative Nodes, Aalborg.

paths are from the top-k diversified shortest path finding algorithm. Table C.4 suggests that when using more negative paths, the accuracy improves. The accuracy improvements from 2 to 3 is the largest, suggesting that the top-k algorithm is very effective on generating high quality negative paths.

Impact of Positive/Negative Nodes in local MI

To study the impact of positive and negative nodes, we consider cases where we only use 20%, 40%, 60%, 80% of positive or negative nodes. Table C.5 shows that the accuracy increases when using more less positive and negative nodes.

6 Conclusion

We study unsupervised path representation learning without using task-specific labels. We propose a novel contrastive learning framework Path InfoMax (*PIM*), including a curriculum negative sampling strategy to generate

a small number of negative paths and a training mechanism that jointly learns distinguishable path representations from both a global and a local view. Finally, we conduct experiments on two datasets with two downstream tasks. Experimental results show that *PIM* outperforms other unsupervised methods and, as a pre-training method, *PIM* is able to enhance supervised path representation learning.

References

[1] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *ICML*, 2009, pp. 41–48.

[2] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," in *EMNLP*, 2014, pp. 103–111.

[3] R.-G. Cirstea, T. Kieu, C. Guo, B. Yang, and S. J. Pan, "Enhancenet: Plugin neural networks for enhancing correlated time series forecasting." in *ICDE*, 2021, pp. 1739–1750.

[4] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *NAACL-HLT*, 2019, pp. 4171–4186.

[5] J. Ding, Y. Quan, Q. Yao, Y. Li, and D. Jin, "Simplify and robustify negative sampling for implicit collaborative filtering," in *NeurIPS*, 2020.

[6] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *KDD*, 2016, pp. 855–864.

[7] C. Guo, B. Yang, J. Hu, and C. S. Jensen, "Learning to route with sparse trajectory sets," in *ICDE*, 2018, pp. 1073–1084.

[8] C. Guo, B. Yang, J. Hu, C. S. Jensen, and L. Chen, "Context-aware, preference-based vehicle routing," *VLDB J.*, vol. 29, no. 5, pp. 1149–1170, 2020.

[9] R. D. Hjelm, A. Fedorov, S. Lavoie-Marchildon, K. Grewal, P. Bachman, A. Trischler, and Y. Bengio, "Learning deep representations by mutual information estimation and maximization," in *ICLR*, 2019.

[10] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[11] J. Hu, C. Guo, B. Yang, and C. S. Jensen, "Stochastic weight completion for road networks using graph convolutional networks," in *ICDE*, 2019, pp. 1274–1285.

[12] J. Hu, B. Yang, C. Guo, C. S. Jensen, and H. Xiong, "Stochastic origin-destination matrix forecasting using dual-stage graph convolutional, recurrent neural networks," in *ICDE*, 2020, pp. 1417–1428.

[13] T. Kieu, B. Yang, C. Guo, and C. S. Jensen, "Distinguishing trajectories from different drivers using incompletely labeled trajectories," in *CIKM*, 2018, pp. 863–872.

[14] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2015.

[15] C. Li, J. Ma, X. Guo, and Q. Mei, "Deepcas: An end-to-end predictor of information cascades," in *WWW*, 2017, pp. 577–586.

[16] H. Liu, C. Jin, B. Yang, and A. Zhou, "Finding top-k shortest paths with diversity," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 3, pp. 488–502, 2018.

[17] Z. Liu, V. W. Zheng, Z. Zhao, F. Zhu, K. C. Chang, M. Wu, and J. Ying, "Semantic proximity search on heterogeneous graph by proximity embedding," in *AAAI*, 2017, pp. 154–160.

[18] S. A. Pedersen, B. Yang, and C. S. Jensen, "Anytime stochastic routing with hybrid learning," *PVLDB*, vol. 13, no. 9, pp. 1555–1567, 2020.

[19] ——, "Fast stochastic routing under time-varying uncertainty," *VLDB J.*, vol. 29, no. 4, pp. 819–839, 2020.

[20] Z. Peng, W. Huang, M. Luo, Q. Zheng, Y. Rong, T. Xu, and J. Huang, "Graph representation learning via graphical mutual information maximization," in *WWW*, 2020, pp. 259–270.

[21] F. Sun, J. Hoffmann, V. Verma, and J. Tang, "Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization," in *ICLR*, 2020.

[22] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "LINE: large-scale information network embedding," in *WWW*, 2015, pp. 1067–1077.

[23] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *NIPS*, 2017, pp. 5998–6008.

[24] P. Velickovic, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep graph infomax," in *ICLR*, 2019.

[25] P. Wang, S. Li, and R. Pan, "Incorporating GAN for negative sampling in knowledge representation learning," in *AAAI*, 2018, pp. 2005–2012.

[26] Z. Wu, Y. Xiong, S. X. Yu, and D. Lin, "Unsupervised feature learning via non-parametric instance discrimination," in *CVPR*, 2018, pp. 3733–3742.

[27] S. B. Yang, C. Guo, and B. Yang, "Context-aware path ranking in road networks," *IEEE TKDE*, 2020.

[28] S. B. Yang and B. Yang, "Learning to rank paths in spatial networks," in *ICDE*, 2020, pp. 2006–2009.

Paper D

Weakly-supervised Temporal Path Representation Learning with Contrastive Curriculum Learning

Sean Bin Yang, Chenjuan Guo, Jilin Hu, Bin Yang, Jian Tang, and Christian S. Jensen

The paper has been published in the
IEEE 38th International Conference on Data Engineering (ICDE),
pp. 2873-2885, 2022.

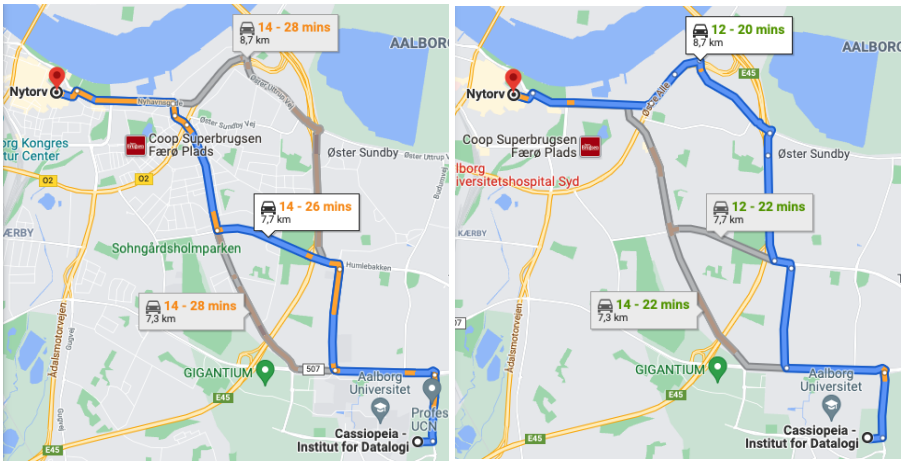
Abstract

In step with the digitalization of transportation, we are witnessing a growing range of path-based smart-city applications, e.g., travel-time estimation and travel path ranking. A temporal path (TP) that includes temporal information, e.g., departure time, into the path is of fundamental to enable such applications. In this setting, it is essential to learn generic temporal path representations (TPRs) that consider spatial and temporal correlations simultaneously and that can be used in different applications, i.e., downstream tasks. Existing methods fail to achieve the goal since (i) supervised methods require large amounts of task-specific labels when training and thus fail to generalize the obtained TPRs to other tasks; (ii) though unsupervised methods can learn generic representations, they disregard the temporal aspect, leading to sub-optimal results.

To contend with the limitations of existing solutions, we propose a Weakly-Supervised Contrastive learning model. We first propose a temporal path encoder that encodes both the spatial and temporal information of a temporal path into a TPR. To train the encoder, we introduce weak labels that are easy and inexpensive to obtain, and are relevant to different tasks, e.g., temporal labels indicating peak vs. off-peak hour from departure times. Based on the weak labels, we construct meaningful positive and negative temporal path samples by considering both spatial and temporal information, which facilities training the encoder using contrastive learning by pulling closer the positive samples' representations while pushing away the negative samples' representations. To better guide the contrastive learning, we propose a learning strategy based on Curriculum Learning such that the learning performs from easy to hard training instances. Experimental studies involving three downstream tasks, i.e., travel time estimation, path ranking, and path recommendation, on three road networks offer strong evidence that the proposal is superior to state-of-the-art unsupervised and supervised methods and that it can be used as a pre-training approach to enhance supervised TPR learning.

1 Introduction

Road-network paths are central in many intelligent transportation system applications, such as path recommendation [10, 11, 16], routing [29, 32, 33, 50], travel cost estimation [15, 17, 22, 44], and traffic analysis [3, 5, 6, 21, 23, 41]. Path representation (PR) learning is the process of learning representations of paths in the form of vectors with a fixed and relatively low dimensionality that is independent of the actual lengths of path. Thus, such representations can render downstream applications that operate on paths much more efficient than what is possible when operating directly on traditional, variable-length representations of paths. This illustrates the potential of path representation learning for improving intelligent transportation applications. Indeed, initial



(a) 8:00 a.m. **(b)** 10:00 a.m.

Figure D.1: Travel Time Estimation. Travel times of paths from *Cassiopeia* to *Nytorv* at (a) 8:00 a.m. and (b) 10:00 a.m.

studies of path representation learning [27, 45] already exist.

In this study, we aim at learning generic temporal path representations (TPRs), meaning that the representations can be utilized in variety of downstream tasks, and we do so without the need for task-specific labeled data. Next the temporal aspect is essential in transportation applications. Consider the travel-time estimation example from Google Maps in Figure D.1. Travel from *"Cassiopeia"* to *"Nytorv"* takes longer at 8:00 a.m. than at 10:00 a.m., due to the traffic congestion during morning peak hours. Further, it can be seen that the path recommendation rankings are also different. It recommends to avoid the highway at 8:00 a.m. due to the heavy congestion there, while recommends the highway again at 10:00 a.m., when the traffic is clear. Learning path representations without considering the temporal aspect results in poor accuracy, which in turn reduces the utility of such representation in downstream tasks. However, it is non-trivial to learn generic TPRs using either supervised or unsupervised learning.

Supervised approaches (Figure D.2(a)) learn TPRs based on task-specific labels [26, 46]. We call these "strong" labels because their use targets specific tasks. For example, for the task of travel-time prediction, the time and path encoders first take as input a departure time t and a path p, respectively. Then, their outputs are aggregated into a travel-time (TT) specific TPR, which is then utilized to predict the travel time of path p when departure at time t. This supervised learning approach has two drawbacks: (1) task specific TPRs do not generalize across tasks. For example, TPRs learned from travel-time labels may perform poorly in path ranking tasks. (2) supervised learning requires a

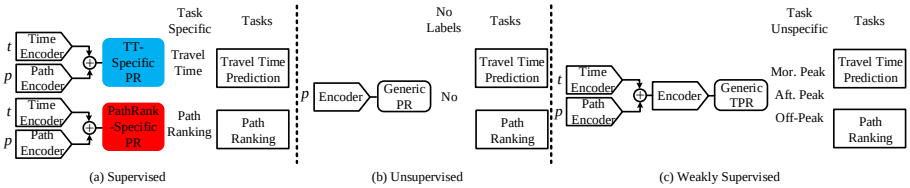


Figure D.2: Supervised, Unsupervised, and Weakly-Supervised methods for learning Temporal Path Representations (TPR): (a) Supervised learning relies on task-specific labels to obtain task-specific path representations (PRs), and thus fails to generalize across tasks; (b) Unsupervised learning produces generic path representations for use in different tasks, but fails to capture temporal traffic aspects of paths; (c) Weakly supervised learning (**Ours**) uses weak labels to learn generic TPRs.

large amount of labeled training data, which may be impossible or expensive to obtain.

Unsupervised approaches do not rely on task specific labels and are thus able to offer generic path representations. Existing unsupervised path representation approaches rely heavily on unsupervised graph representation learning, where the representations of the edges in a path are aggregated into a path representation [36, 45]. Since existing graph representation learning does not consider temporal information, the obtained path representations also lacks the temporal aspect. However, as argued in the context of Figure D.1, disregarding the temporal aspect adversely affects the quality of downstream tasks.

In this paper, we target at a solution that is able to offer generic TPRs that take into account spatial and temporal correlations simultaneously without using task specific labels. To this end, we propose a temporal path encoder, consisting of a temporal and a spatial embedding module, to encode a temporal path into a TPR by considering both spatial and temporal information. Specifically, we construct a temporal graph to learn temporal embeddings for different departure times via graph representation learning; next, we embed various traffic related information from pertinent road networks into spatial embeddings. Finally, the temporal path encoder combines the temporal and spatial embeddings to generate the TPR of the input temporal path.

To enable the training of the temporal path encoder such that the obtained TPRs are generic and include temporal information, we introduce weak labels on the temporal aspect. Such weak labels are easy to obtain and are relevant to different tasks. Example weak labels include labels indicating peak vs. off-peak periods, which only depend on departure times. This way, all temporal paths are associated with weak labels according to their departure times.

Next, we construct meaningful positive and negative temporal path samples to enable contrastive learning such that no task-specific labels are required and thus the learned TPRs are generic across downstream tasks. The positive

samples are those with the same paths and same weak labels and all other temporal paths, i.e., same paths with different weak labels, different paths with both same and different weak labels, are negative samples. To learn meaningful representations, we design an objective function to try to pull together representations of positive samples, while separating representations of negative samples. This enables generic TPRs while capturing the temporal information. Unlike the supervised methods, we therefore do not require strong, task-specific labeled data for our training. Instead, by deriving weak labels for temporal paths we obtain more generic representations. And unlike the unsupervised approaches, we consider the temporal aspect when we learn path representations.

To further enhance the weakly-supervised contrastive (WSC) learning, we integrate curriculum learning to improve the convergence rate and generalization capabilities of the TPR learning. Specifically, we propose an curriculum sample evaluation model that outputs difficulty scores for all training samples, according to which the training samples can be sorted. To achieve this we first split the training data set into non-overlapping meta-sets. Then, we train separate weakly supervised contrastive (WSC) models on each meta-set, respectively. Next, we calculate a TPR similarity score and treat it as a difficulty score for each training sample, based on which we sort all the samples. Finally, we provide a curriculum selection algorithm to perform the curriculum learning according to the difficulty scores.

To the best of our knowledge, this is the first solution that combines advantages of supervised and unsupervised learning to learn generic temporal path representations. In summary, we make the following contributions.

- We formulate the temporal path representation learning problem.
- We propose a weakly-supervised, contrastive model (basic framework) to learn generic path representations that take temporal information into account.
- We integrate curriculum learning into the weakly-supervised contrastive model to further enhance the learned temporal path representations, yields the advanced framework.
- We report on extensive experiments using three real-world data sets on three downstream tasks to assess in detail the effectiveness of the proposed framework.

2 Related Work

2.1 Path Representation Learning

Deep learning is already being used for representation learning, and different studies have proposed a variety of methods to learn useful path representations. To the best of our knowledge, recurrent neural network (RNN) architectures, including long short-term memory (LSTM) [13] and gated recurrent unit (GRU) networks [4], have been established firmly as the state-of-the-art for path representation learning. Deepcas [25] leverages bi-directional GRUs to sequentially process forward and backward node representations of paths, representing a path by the concatenation of resulting forward and backward hidden vectors. ProxEmbed [30] uses LSTMs to process node representations and apply max-pooling on outputs across all time steps to generate a path representation. SPAE [26] proposes self-attentive path embedding. Paths of arbitrary length are first embedded into fixed-length vectors that are then fed to LSTMs to generate path representations. PathRank [46] propose a supervised path representation learning model that takes departure time as additional context information. The above methods all perform end-to-end training and rely on the availability of large amounts of labeled training data. In addition, their path representations are task specific. Most recently, the unsupervised path representation learning framework *PIM* [45] learns path representations. However, it does not include temporal information. In contrast, we propose a temporal path representation learning framework based on weakly-supervised contrastive loss that can learn path representations when given different departure times.

2.2 Contrastive Learning

Recently, the most effective approaches for learning representations with or without labeled data have been supervised or unsupervised contrastive learning [12, 20, 34, 36, 38, 45], which have shown impressive performance in computer vision and graph learning. As a form of metric learning [2], contrastive approaches achieve representations in a discriminating manner through contrasting positive data pairs against negative data pairs. In early work, Hjelm et al. [12] proposed Deep InfoMax (DIM) for learning a generic image representations by maximizing mutual information between local and global features in an unsupervised manner. Inspired by DIM, Velickovic et al. [38] proposed a similar approach, called Deep Graph Informax (DGI), that learns graph-node representations in an unsupervised manner. Recently, Sun et al. [36] proposed InfoGraph for graph representation learning and evaluated the proposal in both unsupervised and semi-supervised settings. Most Recently, Khosla et al. [20] extended the self-supervised batch contrastive method to a fully-supervised setting, making it possible to leverage label information effectively. However, no previous studies have explored the

direction of weakly supervised contrastive learning.

2.3 Curriculum Learning

Inspired by the human learning principle of starting by learning simple tasks before proceeding to learn increasingly hard tasks, curriculum learning (CL) [1] uses nonuniform sampling of mini-batches according to the order of sample difficulty. Due its great potential to improve sample efficiency for different deep learning models, CL has attracted considerable interest and has found application in different research domains, e.g., computer vision [18, 24, 40], and natural language processing (NLP) [35, 39, 43]. However, none of these studies apply CL to path representation learning. *PIM* [45] is the closet to our paper, in that it proposes a curriculum negative sampling method to enhance the path representation learning. However, *PIM* focuses on negative sampling generations, but not on training. Xu et al. [43] propose two-staged curriculum learning for NLP, including difficulty evaluation and curriculum arrangement. Inspired by Xu et al. [43], we propose a curriculum learning framework that can evaluate the difficulty levels of data in a training data set automatically, so that models can be trained on increasingly difficult subsets of the training data set. The new framework features two key novelties. (i) Difficulty score computation: In NLP settings, difficulty scores, e.g., accuracy or F1 score, are computed based on strong labels in a supervised setting. In contrast, our difficulty scores are computed based on representation similarities, which do not rely on strong labels. (ii) How the training data is split into metasets: In NLP settings, training data is often split into metasets at random. In contrast, we split the training data based on the lengths of paths. This facilitates distinguishing the difficulty scores of paths.

3 Preliminaries

We first cover important concepts and then present the problem statement.

3.1 Definitions

Road network. A road network is defined as a directed graph $G = (\mathbb{V}, \mathbb{E})$, where $\mathbb{V}$ is a set of vertices v_i that represent intersections and $\mathbb{E} \subset \mathbb{V} \times \mathbb{V}$ is a set of edges $e_i = (v_j, v_k)$ that represent edges. Figure D.3 shows an example road network.

GPS Trajectory. A GPS trajectory $traj = \langle (l_i, t_i) \rangle_{i=1}^{|traj|}$ of a moving object is defined as a timestamped location sequence, where l_i represents the GPS location at timestamp t_i.

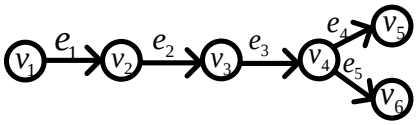


Figure D.3: An Example Road Network

Path. A path $= \langle e_i \rangle_{i=1}^{||}$ is a sequence of adjacent edges, where $e_i \in \mathbb{E}$ is the i-th edge in the path.

Temporal Path. A temporal path is given by $tp = (, t)$, where $tp.$ is a path and $tp.\mathbf{t}$ is a departure time.

Downstream Task. A downstream task is a task that make estimations based on temporal path representations. Specifically, we consider travel time estimation, path ranking score estimation, and path recommendation.

Weak Labels. Weak labels are easy and inexpensive to obtain, which do not dependent on specific tasks but are relevant to different tasks.

Example. An example of weak labels are labels indicating peak v.s. off-peaks periods based on the departure time. For example, it can be Morning peak (7 to 9 a.m., weekdays), Afternoon peak (4 to 7 p.m., weekdays), and Off-peak (all other times). Such labels are easy and inexpensive to obtain, compared to task-specific labels, e.g., labels indicating travel time or path ranking for different paths. Meanwhile, such weak labels are also relevant for three downstream tasks, because the travel time, path ranking, and path recommendation of the same path during peak vs. off-peak periods often differ significantly.

Temporal Path Representation. The temporal path representation TPR_{tp} of a temporal path tp is a vector in R^{d_h}, where d_h is the dimensionality of the vector.

3.2 Problem Statement

Given a set of temporal paths $\mathbb{TP} = \{tp_1, tp_2, \ldots, tp_n\}$ where each temporal path tp_i is with a weak label y_i, temporal path representation learning (*TPRL*) aims at learning a temporal path representation $TPRL(tp_i)$ for each temporal path $tp_i \in \mathbb{TP}$ as formulated in Eq. D.1.

$$TPRL_\psi(tp_i) : R^{d_{tem}} \times \mathbb{R}^{M \times d} \rightarrow \mathbb{R}^{d_h}, \tag{D.1}$$

where ψ represents the learnable parameters for the path encoder, M is the total number of edges in the path, d, d_{tem}, and d_h are the feature dimensions for an edge, a departure time embedding, and a resulted temporal path representation, respectively.

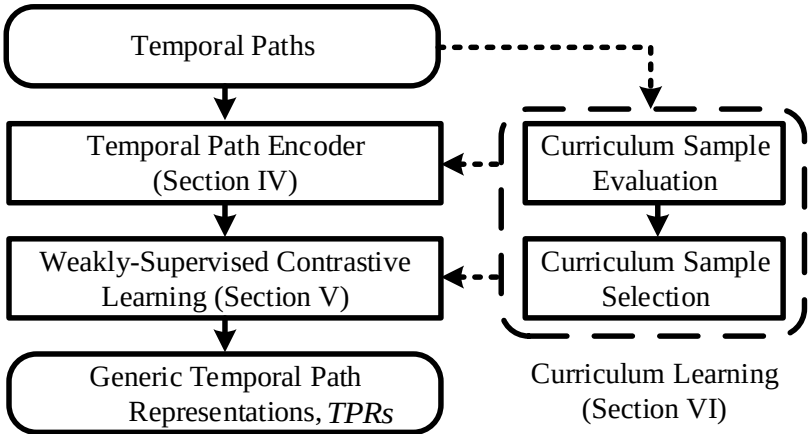


Figure D.4: Solution Overview

3.3 Solution Overview

Figure D.4 shows an overview of the proposed weakly-supervised contrastive curriculum learning (*WSCCL*), which consists of three modules: 1) Temporal path encoder, 2) Weakly-supervised contrastive learning, and 3) Curriculum Learning. The details of those modules are provided in Sections 4, **??**, and 6, respectively.

4 Temporal Path Encoder

Figure D.5 gives an overview of the WSC base framework. We detail the Temporal Path Encoder, which consists of a Spatial Embedding layer, a Temporal Embedding layer, and an LSTM layer. Spatial embedding takes as input a sequence of edges and outputs a sequence of spatial feature representations. Temporal embedding takes temporal information as input and converts input to a temporal feature vector. Next, we concatenate the spatial and temporal vector representations and feed the resulting representation to the LSTM model that extracts coupled spatio-temporal relationships and outputs spatio-temporal edge representations. Finally, we aggregate these edge representations to obtain the desired temporal path representations.

4.1 Temporal Embedding

Motivated by Yuan et al. [48], we construct a temporal graph $G' = (\mathbb{V}', \mathbb{E}')$, where each node $v' \in \mathbb{V}'$ denotes a departure time slot and each edge $e' \in \mathbb{E}'$ denotes a connection between two time slots. We first split the 24 hours

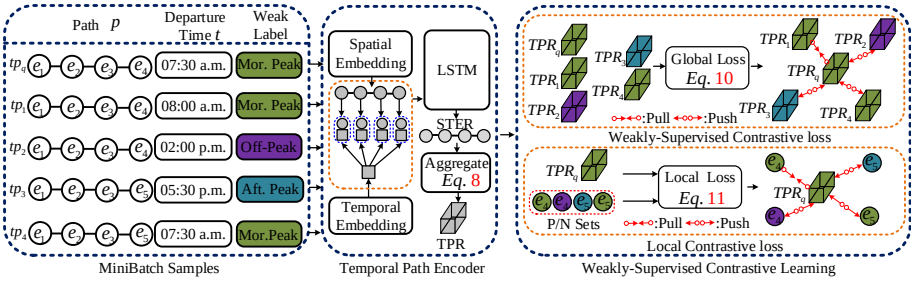


Figure D.5: Illustration of Basic Framework. Given a set of temporal paths in a minibatch, an input temporal path is encoded into a feature map by the temporal path encoder. The global loss framework takes a (query temporal path tp_q, positive or negative temporal path) pair as input and pulls together the TPRs of the query path and positive path, while pushing away the TPRs of the query path and negative paths. The local loss takes as input a TPR of query path and a spatio-temporal edge representation (STER) and brings a TPR of a query path with the positive edge representations closer while pushing apart TPR of a query path with the negative edge representations.

of a day into 5-minute time slots to get 288 time slots. Then, to capture periodicities, we consider the 7 days of a week separately to get a total of 2016 nodes in the temporal graph, where each node represents a time slot and a day of the week. Next, we use two one-hot vectors, $t_s \in R^{288}$ and $t_w \in R^7$, to denote the initial representations for time slots and days of the week, respectively. For example, the departure time 00:06 a.m. on Monday is represented as $t_s = [0, 1, 0, \cdots, 0]$ and $t_w = [1, 0, 0, 0, 0, 0, 0]$.

Therefore, the node representation t_g^{emb} of temporal graph G' can be formulated as the concatenation of the two representations: $t_g^{emb} = [t_s, t_w] \in R^{288+7}$.

To consider the local similarities and weekly periodicities, we draw connections between different nodes in the temporal graph. More specifically, we connect adjacent time slots, indicating that neighboring time ranges should be similar. Further, we connect the adjacent nodes during neighboring days, indicating that time ranges during neighboring days should be similar. Finally, we also connect the time slots between Sunday and Monday.

Next, we apply a graph representation model, specifically, *node2vec* [9], to the temporal graph to further learn node representations in the graph. This is expressed in Eq. D.2.

$$t^{all} = \text{Node2Vec}^{tg}(t_g^{emb}), \tag{D.2}$$

where $t^{all} \in R^{d_{tem}}$ is the finalized temporal representation.

4.2 Spatial Embedding

Recall that a path consists of a sequence of edges, each of which as a number of spatial features, including, e.g., road types, number of lanes.

Capture of Spatial Features The intuitive way to learn TPRs is to encode the spatial features of all edges in a temporal path into TPRs. We consider the following four types of spatial edge features: *Road Type (RT):* a categorical value that includes primary, secondary, residential, etc. *Number of Lanes (NoL):* a real value that represents the number of traffic lanes in the edge. *One Way (OW):* a Boolean that indicates whether the edge is one way or not. *Traffic Signals (TS):* a Boolean that indicates whether the edge has one or more traffic signals on the edge or not.

Next, we represent these different categorical features as one-hot vectors, which can be formulated as $s^{one}_{RT} \in R^{n_{rt}}$, $s^{one}_{NoL} \in R^{n_l}$, $s^{one}_{OW} \in R^{n_o}$, $s^{one}_{TS} \in R^{n_{ts}}$, where n_{rt}, n_l, n_o, n_{ts} represent the number of possible values in the four types of features.

Afterward, we leverage embedding matrices to convert these sparse one-hot vectors into dense vectors, which is formulated in Eq. D.3.

$$\begin{aligned} s^{emb}_{RT} = M^T_{RT} \times s^{one}_{RT} \, , s^{emb}_{NoL} = M^T_{NoL} \times s^{one}_{NoL}, \\ s^{emb}_{OW} = M^T_{OW} \times s^{one}_{OW} \, , s^{emb}_{TS} = M^T_{TS} \times s^{one}_{TS}, \end{aligned} \tag{D.3}$$

where $M_{RT} \in R^{n_{rt} \times d_{rt}}$, $M_{NoL} \in R^{n_l \times d_l}$, $M_{OW} \in R^{n_o \times d_o}$, and $M_{TS} \in R^{n_{ts} \times d_{ts}}$, where d_{rt}, d_l, d_o, and d_{ts} are feature dimensions for dense vectors of RT, NoL, OW, and TS, respectively.

Finally, we concatenate all four dense features as final spatial feature embeddings for edge s_i, which can be formulated as follows.

$$s^{type} = [s^{emb}_{RT}, s^{emb}_{NoL}, s^{emb}_{OW}, s^{emb}_{TS}], \tag{D.4}$$

where $[\cdot, \cdot]$ denotes concatenation vectors.

Road Network Topology Since each edge has effects on its neighboring edges, connected edges should have similar representations. Inspired by graph embedding [9, 34, 38], which aims to learn node representations in a graph by considering the graph topology. Again, we apply *node2vec* [9] to learn graph representation of road network, which can be formulated as $n^{rn}_{vi} = \text{Node2Vec}^{rn}(n^{one}_{vi})$, where n^{one}_{vi} is one-hot vector of node v_i and $n^{rn}_{vi} \in R^{\frac{d_{top}}{2}}$ is the finalized node representation in a road network. The finalized edge representation of $e_k \in \mathbb{E}$ can be formulated as follows.

$$s^{rn}_{e_k} = [n^{rn}_{v_i}, n^{rn}_{v_j}], \tag{D.5}$$

where v_i and v_j are start and end nodes of edge e_k.

Finally, the finalized topology feature with spatial feature of edge e_k can be rewritten as follows.

$$s^{all}_{e_k} = [s^{rn}_{e_k}, s^{type}], \tag{D.6}$$

where $s^{all}_{e_k} \in R^d$ is the finalized spatial embedding for e_k, where $d = d_{rt} + d_l + d_o + d_{ts} + d_{top}$.

4.3 LSTM Encoder

Given an input temporal path tp, we achieve a sequence of spatio-temporal representations $\langle x_{e_1}, x_{e_2}, \cdots x_e \rangle$, where $x_{e_i} = [t^{all}, s_{e_i}^{all}]$, where t^{all} is the temporal embedding for $tp.t$. As recurrent neural networks (RNN) are known to be effective at modelling sequences, we feed this spatio-temporal representation into an RNN to further learn path representations. Specifically, we employ an LSTM model [14] to capture the sequential dependencies by taking each element of the spatio-temporal representations as input, which can be formulated as follows.

$$\hat{\mathbf{p}} = \langle \vec{p}_{e_1}, \vec{p}_{e_2}, \cdots \vec{p}_e \rangle = \text{LSTM}(\langle x_{e_1}, x_{e_2}, \cdots x_e \rangle), \tag{D.7}$$

where $\vec{p}_{e_j} \in R^{d_h}$ is the finalized spatio-temporal representation of edge e_j. $\text{LSTM}(\cdot)$ is the RNN model to capture the sequential dependencies, but it is also possible to use more advanced sequential models, e.g., Transformer [37].

4.4 Aggregate Function

As shown in Figure D.5, the *Aggregate* function takes as input a sequence of spatio-temporal edge feature vectors and returns a TPR. In particular, we aggregate $\hat{\mathbf{p}}$ into a TPR via an aggregate function $\mathcal{P}(\cdot) : \mathbb{R}^{n \times d_h} \longmapsto \mathbb{R}^{d_h}$, where n is the number of edges in the path. We use an average aggregate function that takes the average of the edge representations in $\hat{\mathbf{p}}$ across edges.

$$\vec{h}_p = \frac{\sum_{i=1}^{n} \vec{p}_i}{|\hat{\mathbf{p}}|} \in \mathbb{R}^{d_h}, \tag{D.8}$$

where $\vec{h}_p$ represents the temporal path representation. $\vec{p}_i$ is the latent representation of edge i in the path.

5 Weakly-supervised Contrastive Learning

To ensure that we obtain generic TPRs that apply to different downstream tasks, we employ contrastive learning to construct the learning objectives for the whole framework. Here, we first detail positive and negative sample generation with weak labels. Then, we show how to construct weakly-supervised contrastive global and local losses.

5.1 Generation of Positive and Negative Samples

Positive and negative samples play an essential role in contrastive learning. Contrastive learning does not require strong labels and provides us with

a good way of constructing the learning objectives for our model. In self-supervised contrastive learning, positive samples are always derived from the same object with different views, e.g., cropped parts of an object or generated by different models. Negative samples are simply representations that come from different objects. However, if some negative samples that have properties that are similar to those of positive samples, self-supervised contrastive learning faces difficulties because all negative samples are treated equally.

Suppose we have a set of temporal paths, positive TPs are not only different representations of the same temporal path, but they also include TPs that traverse the same path with the same weak label. In contrast, negative TPs belong to three categories: (1) same paths but different weak labels; (2) different paths but the same weak labels; (3) different paths and different weak labels. Therefore, we can generate multiple positive and negative TPs for a query TP.

The block of MiniBatch Samples in Figure D.5 shows an example, with five TPs, i.e., $tp_q, tp_1, tp_2, tp_3,$ and tp_4, and three weak labels, i.e, morning peak (Mor. Peak), Afternoon peak (Aft. Peak) and Off-Peak. If we take tp_q as the query TP, tp_1 is the corresponding positive sample since the two share the same path (i.e., $\langle e_1, e_2, e_3, e_4 \rangle$) and the same departure weak label (i.e., Mor. Peak), although their exact departure times are different. Next, $tp_2, tp_3,$ and tp_4 are negative samples, where tp_2 has the same path but a different weak label, tp_3 has a different path and a different weak label, and tp_4 has a different path but the same weak label.

5.2 Global Weakly-supervised Contrastive Loss

Given a batch of training samples with batch size B, self-supervised contrastive loss can be formulated as in Eq. D.9.

$$\mathcal{L}^{\text{self}} = \sum_{i=1}^{2B} -\log \frac{\exp\left(< \Psi\left(x_i'\right), \Psi\left(x_p'\right) > / \Psi\left(x_k'\right) > / \hat{\tau} \right)}{\sum_{k=1}^{2B} \mathbf{1}_{i \neq k} \exp < \left(\Psi\left(x_i'\right), \Psi\left(x_k'\right) > / \hat{\tau}\right)}, \tag{D.9}$$

where $\exp(\cdot)$ denotes the exponential operation, $< \cdot, \cdot >$ denotes the inner product of two vectors, $\Psi(\cdot) \in \mathbf{R}^{d_h}$ represents the output from the encoder, $\hat{\tau}$ is a temperature parameter, x_p' is an alternative view of object x_i' which can be generated by using data augmentation, e.g., rotate an image by 90 degree. x_k' denotes the negative samples from the batch, which is all samples from the batch other than x_i', meanwhile $\mathbf{1}_{i \neq k}$ is an indicator vector, where all elements are 1s, except a 0 at the i-th position.

However, the self-supervised contrastive loss in Eq. D.9 is unable to take into account differences among negative samples. Motivated by *SupCon* [20], good generalization requires the ability to capture the similarity between

samples in the same class and contrast them with samples in other classes. In this paper, we propose instead a WSC loss that utilizes positive and negative sample generation, as introduced in Section 5.1. As is shown in Figure D.5, for each query temporal path tp_q, we try to pull closer TPRs with positive temporal path samples, which can be represented as $TPR_q \rightarrow\leftarrow TPR_1$, and push away TPRs from negative temporal path samples, denoted by $TPR_q \leftarrow\rightarrow TPR_2$, $TPR_q \leftarrow\rightarrow TPR_3$, and $TPR_q \leftarrow\rightarrow TPR_4$. This yields global WSC formulated in Eq. D.10.

$$\mathcal{L}^{global} = \sum_{(\boldsymbol{tp}_{i'i})\in} \mathcal{L}^{global}_{(\boldsymbol{tp}_{i'i})} = \sum_{(\boldsymbol{tp}_{i'i})\in} \frac{1}{|\boldsymbol{tp}_i|} \sum_{\boldsymbol{tp}_j \in_{\boldsymbol{tp}_i}} \log \frac{\exp\left(\text{sim}(TPR_i, TPR_j)\right)}{\sum_{\boldsymbol{tp}_k \in_{tp_i}} \exp\left(\text{sim}(TPR_i, TPR_k)\right)}, \tag{D.10}$$

where $\text{sim}(\cdot)$ is cosine similarity function that quantifies the similarity between two TPRs; $= \{(tp_i, y_i)\}_{i=1}^{B}$ is a set of temporal paths in one training batch, where y_i is the departure weak label for tp_i; ${}_{tp_i} = \{tp_j\}_{j=1}^{|_{tp_i}|}$ is the positive sample set for query tp_i, where $y_i = y_j$ and $tp_i.\mathbf{p} = tp_j.\mathbf{p}$; and ${}_{tp_i} = \setminus\{\boldsymbol{tp}_i \cup_{\boldsymbol{tp}_i}\}$ is the negative sample set for query tp_i.

5.3 Local WSC Loss

In addition to the weakly-supervised learning across query temporal path with global positive and negative temporal paths, we also consider local differences between positive and negative temporal edge samples. These acts as a strong regularization that enhances the learning ability of our method. The Local Contrastive Loss element in Figure D.5 illustrates the design of our local contrastive loss, which consists of WSC with TPR_{tp_q}.

The goal of local contrastive learning is to preserve the local similarity between a TPR and the spatio-temporal representation of its edges. In particular, it is expected that a TPR can capture local similarity (edge-level similarity), i.e., TPRs are close to embeddings of positive edge samples and are distant from embeddings of negative edge samples. Similar to global WSC, we formulate the local WSC loss as maximizing the similarity with positive temporal edge samples as well as minimizing the similarity with negative temporal edge samples.

We proceed to describe the construction of the positive and negative edge samples. First, we randomly select edges that appear in positive temporal paths as our positive edge set, which is denoted as $\mathbb{PN}_i$. Then, edges that appear in negative temporal paths are selected as our negative edge set, denoted as $\mathbb{NN}_i$. Next, we set the weak temporal label of each temporal path be the label of the corresponding edge in the temporal path. As is shown in

Figure D.5, for each query path tp_q, we try to pull TPRs with positive edges closer, e.g., (TPR_q, Mor. Peak) $\rightarrow\leftarrow$ ($STER(e_4)$, Mor. Peak), and push away TPRs from negative edges, e.g., (TPR_q, Mor. Peak) $\leftarrow\rightarrow$ ($STER(e_4)$, Off-Peak), (TPR_q, Mor. Peak) $\leftarrow\rightarrow$ ($STER(e_5)$, Aft. Peak), and (TPR_q, Mor. Peak) $\leftarrow\rightarrow$ ($STER(e_5)$, Mor. Peak).

In this phase, the objective of local contrastive learning is to increase the similarity of TPRs with positive edge samples while decreasing the similarity of TPRs with negative edge samples. Using cosine similarity $s(x, y) = x^\top y / \|x\|\|y\|$, we aim to optimize the objective function for local contrastive loss that is formulated in Eq. D.11.

$$\mathcal{L}^{local} = \sum_{(\boldsymbol{tp}_i, n_i) \in} \frac{1}{|\mathbb{PN}_i|} \log \frac{\sum_{(\boldsymbol{e}_j, y_{i)} \in \mathbb{PN}_i} \exp\left(s(TPR_i, e_j)\right)}{\sum_{(\boldsymbol{e}_k, y_j \neq y_i) \in \mathbb{NN}_i} \exp\left(s(TPR_i, e_k)\right)}, \tag{D.11}$$

where $\mathbb{PN}_i$ and $\mathbb{NN}_i$ are the positive and negative edge sets, and y_i denotes the weak label for edge representation, which inherits from the corresponding temporal path.

5.4 Objective for WSC

To train our temporal path encoder in an end-to-end manner, we jointly leverage both the the weakly-supervised global and local contrastive loss. Specifically, the overall objective function to maxmize is defined in Eq. D.12.

$$\mathcal{L} = \arg\max_{\psi} \sum_{i \in I} \lambda \cdot \mathcal{L}_i^{global} + (1 - \lambda) \cdot \mathcal{L}_i^{local}, \tag{D.12}$$

where λ is a balancing factor and I is the set of training batches.

6 Contrastive Curriculum Learning

When training WSC using randomly shuffled training data, the training process is prone to getting stuck in a bad local optimum, which leads to suboptimal TPRs. To alleviate this problem, we build on the intuition that the algorithm should be presented with the training data in a meaningful order that facilitates learning. Specifically, the order of the samples is determined by how easy they are, as this can be expected to enhance weakly-supervised contrastive learning. We proceed to integrate curriculum learning with WSC, thus obtaining the advanced framework called *WSCCL*.

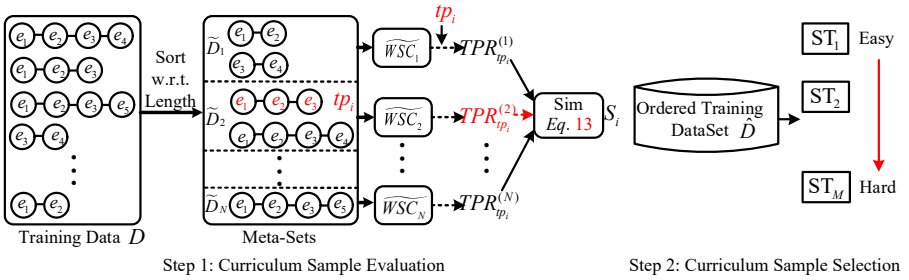


Figure D.6: Illustration of Advanced Framework

6.1 Overview of Curriculum Learning

Motivated by Xu et al. [43], we decompose curriculum sample generation into two stages: *Curriculum Sample Evaluation* and *Curriculum Sample Selection*, as shown in Figure D.6.

1) In curriculum sample evaluation, we assign a difficulty score S_i to path p_i in the training dataset . The score reflects the difficulty of the model to learn a good representation w.r.t. path p_i. 2) In curriculum sample selection, we aim to partition the training data into different difficulty stages. More specifically, we first sort the training data according to the difficulty scores. Then, we split the sorted training data into a sequence of sorted learning stages $\{ST_i | i = 1, 2, \ldots, M\}$ in an easy-to-difficult fashion. Finally, our base model, WSC, is trained according to this curriculum. We detail these two stages in Sections 6.2 and 6.3, respectively.

6.2 Curriculum Sample Evaluation

The difficulty of a given temporal path can be quantified in many different ways. We argue that difficulty scores, like the intrinsic properties of the training dataset, should be decided by the model itself.

We first sort the training data set according to the lengths of the paths. Then, we split into N non-overlapping meta-sets, i.e., $= \{\widetilde{1}, \widetilde{2}, \cdots, \widetilde{N}\}$, where $\widetilde{i}$ is the i-th meta-set, $\widetilde{i} \cap \widetilde{j} = \emptyset, \forall i \neq j, i, j \in [1, N]$. Next, we train N independent WSC models w.r.t. the different meta-sets, i.e., $\widetilde{WSC}_i, i = [1, N]$. More specifically, $\widetilde{WSC}_i$ is trained only on the i-th meta-set, $\widetilde{i}$. After that, we obtain a total of N trained independent WSC models, which we call *Experts*, to evaluate the difficulty of each training sample.

We then use the *Experts* to calculate the difficulty scores for the training data. We take a temporal path $\boldsymbol{tp}_i \in \widetilde{j}$ as input to each *Expert*, $\widetilde{WSC}_j, j \in [1, N]$, and obtain a total of N TPRs, i.e., $TPR^{(1)}_{\boldsymbol{tp}_i}, TPR^{(2)}_{\boldsymbol{tp}_i}, \cdots, TPR^{(N)}_{\boldsymbol{tp}_i}$. Since $\boldsymbol{tp}_i$ comes from $\widetilde{j}$, we take $TPR^{(j)}_{\boldsymbol{tp}_i}$ as the ground truth, and calculate the similarity

between $TPR^{(j)}_{\boldsymbol{tp}_i}$ and $TPR^{(k)}_{\boldsymbol{tp}_i}$, where $\forall k \neq j, k \in [1, N]$. The resulting similarity scores are then summed up to denote as the difficulty score of temporal path $\boldsymbol{tp}_i$, denoted by S_i. This is formulated in Eq. D.13.

$$S_i = \sum_{k=1, k\neq j}^{N} Sim(WSC_j(tp_i), WSC_k(tp_i)), \tag{D.13}$$

where $Sim(\cdot, \cdot)$ denotes the similarity function.

Finally, we repeat the difficulty score calculation for the entire training data. This yields a temporal path dataset $= \{(\boldsymbol{tp}_1, S_1), \cdots, (\boldsymbol{tp}_N, S_N)\}$, where $(\boldsymbol{tp}_i, S_i)$ is one element and S_i is the difficulty score for temporal path $\boldsymbol{tp}_i$. The intuition is that if one element, i.e., $(\boldsymbol{tp}_i, S_i)$ can obtain a similar representations in all other *Experts* compared to the representation from its own *Expert*, then we denote it as an easy sample. Since we calculate the sum of similarities, the higher S_i is, the easier the temporal path $\boldsymbol{tp}_i$.

6.3 Curriculum Sample Selection

To define the curriculum sample selection strategy, we first represent the learning curriculum in a multi-stage manner: $\{ST_i | i = 1, 2, \ldots, M\}$. More specifically, we rank all the training samples according to their difficulty scores and then distribute them evenly among M training stages. This way, the training data is partitioned into M parts with different levels of difficulty, ranging from S_1 (the easiest) to S_M (the hardest). To ensure some local variations, the training samples in different stages are shuffled.

Next, we train our WSC for one epoch at each stage. When the training reaches stage S_M, the WSC should be ready for the original distribution in the whole training dataset . Finally, we add another stage, denoted by S_{M+1}, that covers the entire training data, and WSC is trained on this stage until it converges. For simplicity, we keep $N = M$, as is done elsewhere [43], and we leave the investigation of different combinations of N and M for future work.

7 Experiment Study

7.1 Experimental Setup

Data sets

We use three traffic data sets to study the effectiveness of the proposed framework. Using each of these, we report results for three downstream tasks: travel time prediction, path ranking, and path recommendation.

Aalborg, Denmark [46] We use the road network graph of Aalborg from OpenStreetMap[1] that consists 10,017 nodes and 11,597 edges. We use a substantial GPS data set that captures travel in this road network. Specifically, the data set consists of 180 million GPS records from 183 vehicles sampled at 1 Hz over a two-year period. A well-known map matching method [31] is used to map match the trajectories. This yields a total of 28,370 paths.

Harbin, China [27] The data set was collected from 13,000 taxis in Harbin, China. We extract the corresponding road network from OpenStreetMap. The network contains 8,497 nodes and 14,497 edges. The GPS data was sampled at about 1/30 Hz. After map matching, we obtain 58,977 paths.

Chengdu, China[2] The data set was collected in Chengdu, China during October and November 2016. We extract the corresponding road network from OpenStreetMap. The network contains 6,632 nodes and 17,038 edges. The GPS data was sampled at about 1/4–1/2 Hz. After map matching, we obtain 57,404 paths.

Downstream Tasks

We consider three tasks.

Path Travel Time Estimation Each path is associated with a travel time (in seconds) obtained from the corresponding trajectory. We aim at building a regression model to estimate the travel times of paths based on their TPRs. We evaluate the accuracy of estimations using the Mean Absolute Error (**MAE**), Mean Absolute Relative Error (**MARE**), and Mean Absolute Percentage Error (**MAPE**). Smaller values indicate higher estimation accuracy.

Path Ranking In path ranking, each path is associated with a ranking score in the range $[0, 1]$. The ranking scores are obtained with the help of historical trajectories by following an existing procedure [46]. Given a historical trajectory of a driver, we consider the path used by the trajectory, called the trajectory path, as the top ranked path. Then, we use a path finding algorithm to generate multiple paths connecting the same source and destination. We use the similarity between a generated path and the trajectory path for ranking: the more similar a generated path is to the trajectory path, the higher its similarity score; and the trajectory path itself has score 1 and thus ranks the highest. As for the previous task, we aim at building a regression model to estimate the ranking scores of paths.

[1] https://www.openstreetmap.org
[2] https://outreach.didichuxing.com/research/opendata/en/

To quantify the performance of path ranking, we report the **MAE** of the estimated ranking scores, the Kendall rank correlation coefficient (τ) [19], and the Spearman's rank correlation coefficient (ρ) [49]. The latter two capture the similarity, or consistency, between the ground truth and estimated rankings. The higher the τ and ρ values are, the more consistent the two rankings are, indicating higher accuracy.

Path Recommendation A similar strategy is used in an existing study [28], where a path is associated with a binary label with the help of users' trajectories. A path used by a user's trajectory, say path A, is labeled 1, whereas alternative paths connecting the same source and destination, say paths B and C, are labeled 0. This follows the intuition that given three paths A, B, and C, the user should choose A, not B and C, meaning that path A should be recommended to the user. We conduct the path recommendation task on all three data sets. We evaluate the recommendation effectiveness using the classification **Accuracy** (Acc.) and **Hit Rate** (HR). Higher values indicate better performance.

Baselines

We compare *WSCCL* with 12 baseline methods, which include 7 unsupervised methods and 5 supervised methods. The unsupervised methods are: (1) **Node2vec** [9], Deep Graph InfoMax (**DGI**) [38], Graphical Mutual Information Maximization (**GMI**) [34] are unsupervised graph representation learning frameworks, that give the edge representation for each edge in a graph. We use the average of the edge representations of the edge in a path as the path's representation. (2) **Memory Bank (MB)** [42] is an unsupervised learning approach to learn representations based on contrastive loss, where the representations of negative sample are randomly selected from memory bank. We re-implement **MB** with an LSTM encoder to capture the sequential information in paths. (3) **InfoGraph** [36] is a graph representation learning framework for unsupervised and semi-supervised settings. Here, we consider the unsupervised variant and treat a path as a graph to learn the path's representation. (4) **BERT** [8] is an unsupervised language representation learning model. To enable training, we treat a path as a sentence and mask some edges in the path. Then we split a path P into sub-paths P_1 and P_2, and consider (P_1, P_2) as a valid question-answer (Q&A) pair and (P_2, P_1) as an invalid Q&A pair because the former represents a meaningful ordering while the latter does not. (5) **PIM** [45] is an unsupervised path representation learning model based on global and local mutual information maximization. The supervised methods that take into account the labels from a specific downstream task to obtain path representations, which are: (1) **DeepGTT** [27] is a *supervised* travel time distribution estimation (i.e., to learn the parameters for inverse

Gaussian distribution) framework based on a deep generative model. (2) **HMTRL** [28] enables unified route representation learning and exploits both spatio-temporal dependencies in road networks and the semantic coherence of historical routes. (3) **PathRank** [46] is a *supervised* path representation learning model based on GRUs. (4) **GCN** [7] is a graph covolutional neural network based method that estimates the travel times of all edges in a road network. The travel time of a path is then the sum of the travel times of the edges in the path. (5) **STGCN** [47] is a traffic prediction framework based on spatio-temporal graph convolutional networks. Similar to GCN, the travel time of a path is the sum of the predicted travel times of the edges in the path. Finally, note that for the path ranking task, we cannot simply aggregate the rankings of edges to obtain the rankings of paths. This also applies to the newly included path recommendation task. Thus, GCNs and STGCNs cannot work as baselines for the these two tasks.

Models for Downstream Tasks

For all unsupervised learning approaches, we first obtain a task-independent TPR and then apply a regression model to address different downstream tasks using task-specific labels. In the experiments, we use the ensemble model *Gradient Boosting Regressor* (*GBR*) to estimate travel time and ranking scores for paths as they are regression problems. In addition, we use the ensemble model *Gradient Boosting Classifier* (*GBC*) to make path recommendations, as they are classification problems.

Weak Labels

We consider two different types of weak labels, including peak/off-peak (POP) and traffic congestion indices (TCI). We use POP as default weak labels. We only conduct experiments on the Harbin and Chengdu data sets for TCI since we cannot obtain TCI for Aalborg, Denmark from Baidu Maps[3].

Implementation Settings

We set embedding feature dimensions of RT, NoL, OW, and TS as $d_{rt} = 64$, $d_l = 32$, $d_o = 16$, $d_{ts} = 16$, respectively. The feature dimensions of output from *Node2Vec* on both temporal graph and road networks are set to be 128. Meanwhile, we apply 2 LSTM layers and set the dimensionality of the hidden state h_j to 128. Further, we set the size for temporal path representation dimensionality to 128, i.e., $d_h = 128$. The number of Meta-Set is set to be $N = 10$, and the number of stages in curriculum learning is also set to be $M = 10$. The hyper-parameter λ is set to 0.8. We set the learning rate (lr)

[3]https://jiaotong.baidu.com/congestion/city/urbanrealtime/

Table D.1: Overall Accuracy on Travel Time Estimation

Methods	Aalborg		Harbin		Chengdu	
	MAE	**MAPE**	**MAE**	**MAPE**	**MAE**	**MAPE**
Node2vec	63.82	45.67	269.21	31.41	290.47	34.43
DGI	67.22	49.36	288.09	34.01	312.28	38.46
GMI	70.61	52.40	310.39	36.60	337.06	41.58
MB	57.32	39.37	315.25	35.28	333.73	42.45
BERT	71.96	45.42	217.96	24.52	303.00	36.77
InfoGraph	69.36	41.28	200.81	22.68	291.54	36.07
PIM	57.66	39.34	196.06	21.96	289.10	35.55
DeepGTT	44.78	26.53	214.95	22.76	305.08	35.47
HMTRL	40.59	21.81	228.58	23.60	360.08	37.33
PathRank	37.09	23.89	190.08	20.12	334.94	35.11
GCN	78.04	53.05	368.21	35.62	480.83	42.01
STGCN	58.57	38.97	284.12	23.48	406.09	33.58
WSCCL	**31.66**	**21.39**	**178.89**	**19.43**	**281.20**	**33.30**

to $3e-4$ and the batch size 32. In particular, we train our *WSCCL* using all unlabeled paths shown in data sets section and then we randomly choose 80% and 20% paths in labeled path as training and testing data for GBR. Finally, we evaluate all models on a powerful Linux server with 40 Intel(R) Xeon(R) Gold 5215 CPUs @ 2.50GHz and four Quadro RTX 8000 GPU cards. Finally, all algorithm are implemented in PyTorch 1.9.1. The code is available at https://github.com/Sean-Bin-Yang/TPR.git and the CoRR version is available at https://arxiv.org/abs/2203.16110.

7.2 Experimental Results

Overall accuracy on downstream tasks

Table D.1, Table D.2 and Table D.3 report the overall results on the three downstream tasks. *WSCCL* achieves the best performance on these three tasks for three real-world data sets. The three graph node representation learning methods *Node2vec*, *DGI*, and *GMI* are unable to capture temporal correlation in the temporal path. In contrast, *WSCCL* takes temporal correlation into consideration by virtue of its temporal embedding layer. In addition, the weakly supervised contrastive curriculum learning improves the estimation accuracy.

Although *MB* and *BERT* can capture dependencies among the edge feature vectors in temporal paths, these approaches achieve poor estimation accuracy. This is because *MB* needs large amounts of negative samples to ensure effective training, which is not feasible in our scenario. In addition, *BERT* is not well

Table D.2: Overall Accuracy on Path Rank Estimation

Methods	Aalborg		Harbin		Chengdu	
	MAE	τ	**MAE**	τ	**MAE**	τ
Node2vec	0.23	0.60	0.22	0.37	0.20	0.73
DGI	0.24	0.60	0.21	0.48	0.21	0.52
GMI	0.24	0.59	0.21	0.49	0.21	0.51
MB	0.23	0.62	0.22	0.44	0.20	0.71
BERT	0.26	0.49	0.22	0.46	0.22	0.55
InfoGraph	0.26	0.52	0.21	0.45	0.20	0.73
PIM	0.22	0.60	0.21	0.43	0.19	0.76
DeepGTT	0.39	0.12	0.29	0.04	0.23	0.20
HMTRL	0.17	0.65	0.22	0.51	0.16	0.77
PathRank	0.23	0.64	0.18	0.55	0.17	0.79
WSCCL	**0.15**	**0.68**	**0.14**	**0.68**	**0.13**	**0.84**

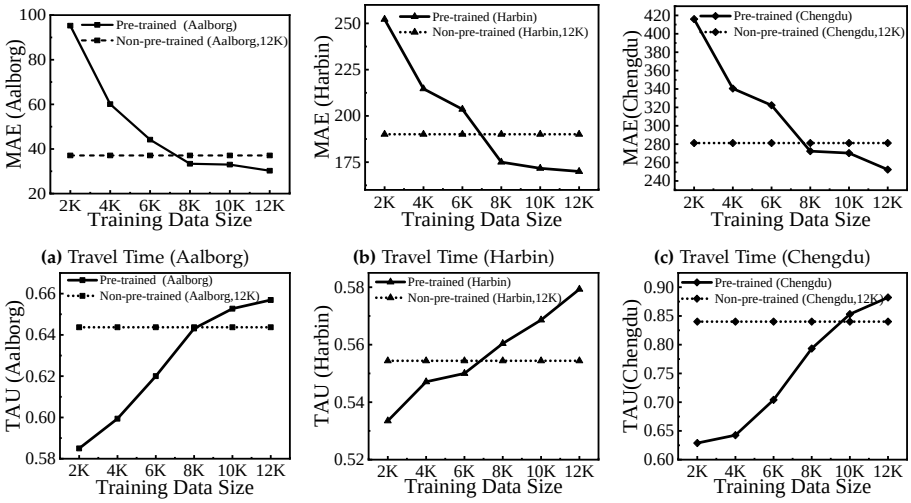
Table D.3: Overall Performance on Path Recommendation

Methods	Aalborg		Harbin		Chengdu	
	Acc.	**HR**	**Acc.**	**HR**	**Acc.**	**HR**
Node2vec	0.79	0.51	0.76	0.51	0.75	0.61
DGI	0.74	0.55	0.70	0.36	0.70	0.57
GMI	0.78	0.53	0.72	0.41	0.68	0.58
MB	0.67	0.48	0.61	0.69	0.73	0.69
BERT	0.60	0.43	0.64	0.53	0.66	0.61
InfoGraph	0.72	0.69	0.79	0.78	0.73	0.65
PIM	0.79	0.82	0.86	0.83	0.76	0.74
HMTRL	0.80	0.86	0.81	0.82	0.78	0.83
PathRank	0.77	0.71	0.79	0.74	0.77	0.73
WSCCL	**0.82**	**0.88**	**0.97**	**0.91**	**0.81**	**0.90**

suited for our setting of learning generic TPRs since BERT cannot support contrastive learning in the setting of multiple positive samples against multiple negative samples.

InfoGraph learns full graph representations. However, it only applies a local view and cannot capture the sequential information of edge in a path. In contrast, *WSCCL* not only uses sequence model (e.g., LSTM) to capture sequential information between edge in a path but considers both the (global) path and (local) edge levels. Although *PIM* is designed for path representation learning, it is unable to learn meaningful TPRs because it ignores temporal information and only has one positive sample. In contrast, *WSCCL* allows multiple positive temporal path samples in each minibatch and also uses a learned curriculum instead of the pre-defined curriculum negative

sampling used by *PIM*. The supervised learning methods *DeepGTT*, *HMTRL*, and *PathRank* achieve relatively poor accuracy due to the small size of labeled training data. Since task-specific labels ("strong labels") are expensive to obtain, we consider a setting where labelled training data is limited. *DeepGTT* exhibits the worst performance on the Path Ranking task because it is designed for travel-time estimation, which is evidence for the poor generalizability of supervised feature representation learning, as discussed in Section **??**. In contrast, the *GCN* and *STGCN* results also are worse than the *WSCCL* results. This is because dependencies among edges are disregarded.



(a) Travel Time (Aalborg) **(b)** Travel Time (Harbin) **(c)** Travel Time (Chengdu)

(d) Path Ranking (Aalborg) **(e)** Path Ranking (Harbin) **(f)** Path Ranking (Chengdu)

Figure D.7: Effects of Pre-training.

Using *WSCCL* as a Pre-training Method

We conduct experiments that treat *WSCCL* as a pre-training method for the supervised method *PathRank*. Here, *PathRank* takes as input a sequence of edge features and estimates the travel time and ranking score. To use *WSCCL* for pre-training method for *PathRank*, we first train *WSCCL* in a weakly supervised manner and then apply the learned parameters in temporal path encoder to initialize the encoder in *PathRank*.

Figure D.7 shows performance of *PathRank* with and without pre-training for the two tasks. Without pre-training, *PathRank* is trained by using 12K labeled training paths. We observe the following: 1) With pre-training we can achieve the same performance as with non-pre-trained *PathRank* while using fewer labeled training paths. For example, when using *WSCCL* for pre-training, *PathRank* only needs 8K, 7K and 10K labeled samples for the Aalborg,

Table D.4: Effect of the CL Design Strategy

Methods	Aalborg					
	Travel Time Estimation			Path Ranking		
	MAE	MARE	MAPE	MAE	τ	ρ
Heuristic	33.58	0.15	22.06	0.19	0.61	0.65
WSCCL	**31.66**	**0.13**	**21.39**	**0.15**	**0.68**	**0.72**

Table D.5: Effects of CL, Global Loss and Local Loss

Methods	Aalborg					
	Travel Time Estimation			Path Ranking		
	MAE	MARE	MAPE	MAE	τ	ρ
w/o CL	32.58	0.14	21.88	0.19	0.62	0.66
w/o Global	51.19	0.22	31.13	0.24	0.54	0.58
w/o Local	32.80	0.14	23.34	0.21	0.57	0.62
WSCCL	**31.66**	**0.14**	**21.39**	**0.15**	**0.68**	**0.72**

Harbin and Chengdu data sets, respectively, to achieve the same performance as *PathRank* with 12K labeled samples, on the path ranking task. 2) When we pre-train *PathRank* with 12K samples, the performance is much better than without pre-training. In both tasks, we obtain similar observations, which indicates that *WSCCL* can be applied advantageously for the pre-training of different downstream tasks.

Ablation Studies

We conduct ablation studies on *WSCCL* to observe 1) the effect of the CL design strategy on TPR learning; 2) the effect of variants of *WSCCL*, specifically *CL*, *global loss*, and local loss; 3) the effect of different weak labels; and 4) the effect of temporal information.

Effect of the CL Design Strategy To observe the effectiveness of the learned CL, we compare with *WSCCL* with a heuristic curriculum design where we simply sort the paths based on the number of edges. The comparison between these two CL variants is reported in Table D.4 that shows that our learned curriculum is better than the heuristic curriculum design on all tasks over on Aalborg data sets. This is because the lengths of edges are varying (from few meters to kilometers), even two paths with the same number of edges, the lengths of two paths may have a big difference. Thus, the difficulties of paths cannot simply represented by the number of edges directly.

Effects of Global Loss, Local Loss, and CL To study the effect of these three modules, we consider three variants of *WSCCL*: 1) *w/o Global*, 2) *w/o*

Table D.6: Effect of Different Weak Labels

Methods	Harbin					
	Travel Time Estimation			Path Ranking		
	MAE	MARE	MAPE	MAE	τ	ρ
WSCCL-TCI	177.07	0.18	19.19	0.13	0.70	0.74
WSCCL-POP	178.89	0.18	19.43	0.14	0.68	0.73
Methods	**Chengdu**					
	Travel Time Estimation			Path Ranking		
	MAE	MARE	MAPE	MAE	τ	ρ
WSCCL-TCI	280.85	0.29	32.97	0.12	0.86	0.87
WSCCL-POP	281.20	0.29	33.30	0.13	0.84	0.86

Local, and 3) *w/o CL*. In *w/o Global*, we remove Global WSC loss from *WSCCL*, in *WSCCL w/o Local*, we remove the Local WSC loss, and in *WSCCL w/o CL*, the curriculum strategy is omitted. The results on Aalborg data sets are reported in Table D.5. We can observe that *WSCCL w/o Global* shows the worst performance and has a clear margin to the other variants. This shows that the proposed Global WSC performs well. We also observe that *WSCCL* achieves the best performance. This indicates that all the proposed modules contribute positively to the final performance, which validates the overall design.

Effect of Different Weak Labels We conduct additional experiments by using traffic congestion indices (TCI), which indicate four congestion levels in a city across time, as weak labels. Table D.6 shows the results on the Harbin and Chengdu data sets. We observe that *WSCCL* works well when using the TCI as weak labels.

Effect of Temporal Information We further conduct experiments on the three data sets using a *WSCCL* variant that disregards temporal information. The results, shown in Table D.7, indicate that the non-temporal *WSCCL-NT* performs worse than *WSCCL* on both downstream tasks, suggesting that our temporal embedding is effective.

Comparison with Temporally Enhanced Unsupervised Method

To compare *WSCCL* with the unsupervised *PIM* method, we first incorporate a temporal representation into the non-temporal path representations learned by *PIM*. Specifically, we use the same temporal embedding to learn temporal representations and then concatenate these with the path representation from PIM to obtain *PIM-Temporal* unsupervised TPRs. The results of comparing this approach with *WSCCL* are reported in Table D.8. We see that *WSCCL*

Table D.7: Effect of Temporal Information

Methods	Aalborg					
	Travel Time Estimation			PathRank		
	MAE	MARE	MAPE	MAE	τ	ρ
WSCCL	**31.66**	**0.14**	**21.39**	**0.15**	**0.68**	**0.72**
WSCCL-NT	41.25	0.18	29.38	0.21	0.55	0.59
Methods	**Harbin**					
	Travel Time Estimation			PathRank		
	MAE	MARE	MAPE	MAE	τ	ρ
WSCCL	**178.89**	**0.18**	**19.43**	**0.14**	**0.68**	**0.73**
WSCCL-NT	199.58	0.20	22.20	0.15	0.64	0.68
Methods	**Chengdu**					
	Travel Time Estimation			PathRank		
	MAE	MARE	MAPE	MAE	τ	ρ
WSCCL	**281.20**	**0.29**	**33.30**	**0.13**	**0.84**	**0.86**
WSCCL-NT	292.76	0.31	35.10	0.18	0.81	0.83

Table D.8: Comparison with Temporally Enhanced Unsupervised *PIM* Method

Methods	Aalborg					
	Travel Time Estimation			Path Ranking		
	MAE	MARE	MAPE	MAE	τ	ρ
PIM-Temporal	42.27	0.19	27.95	0.19	0.65	0.70
WSCCL	**31.66**	**0.13**	**21.39**	**0.15**	**0.68**	**0.72**

Table D.9: Comparison with Supervised Methods

Methods	Aalborg					
	Travel Time Estimation			Path Ranking		
	MAE	MARE	MAPE	MAE	τ	ρ
PathRank-PR	37.09	0.16	23.89	0.24	0.58	0.62
PathRank-TTE	55.08	0.24	36.71	0.23	0.64	0.68
HMTRL-PR	40.59	0.18	21.81	0.25	0.60	0.64
HMTRL-TTE	47.22	0.21	29.97	0.17	0.65	0.68
DeepGTT-PR	44.78	0.20	26.53	0.31	0.56	0.57
DeepGTT-TTE	59.52	0.26	37.80	0.39	0.12	0.12
WSCCL	**31.66**	**0.13**	**21.39**	**0.15**	**0.68**	**0.72**

Table D.10: Effects of λ

λ	Aalborg					
	Travel Time Estimation			Path Ranking		
	MAE	MARE	MAPE	MAE	τ	ρ
0.0	51.19	0.22	31.13	0.24	0.54	0.58
0.2	40.25	0.18	24.67	0.22	0.60	0.64
0.4	34.22	0.15	21.80	0.18	0.64	0.68
0.6	34.76	0.15	22.35	0.17	0.65	0.69
0.8	**31.66**	**0.14**	**21.39**	**0.15**	**0.68**	**0.72**
1.0	32.80	0.14	23.34	0.21	0.57	0.62

Table D.11: Effects of Number of Meta-Set.

N	Aalborg					
	Travel Time Estimation			Path Ranking		
	MAE	MARE	MAPE	MAE	τ	ρ
2	36.64	0.17	25.86	0.20	0.56	0.60
6	36.06	0.16	24.96	0.20	0.56	0.60
10	**31.66**	**0.14**	**21.39**	**0.15**	**0.68**	**0.72**
14	33.16	0.15	21.60	0.19	0.58	0.63
18	33.47	0.15	21.65	0.20	0.56	0.61

outperforms *PIM-Temporal* on both tasks. This indicates that TPRs obtained by adding a temporal representation to the path representation directly is not as good as the TPR learned by *WSCCL*. This is because the added temporal representation can only capture the overall traffic condition on the road network for all the paths, yet not more unique spatio-temporal path representations learned by our *Temporal Path Encoder* for different paths. This experiment shows that it is not feasible to obtain a generic TRP by independently adding a temporal representation to an unsupervised learned generic PR (i.e., spatial representation). Further, it offers evidence of a more correlated and intricate interplay between space and time in TPRs. For example, during the morning peak hours, different paths may have different traffic conditions.

Comparison with Supervised Method

To study the applicability of TPRs from supervised method across tasks, we use the supervised methods *PathRank*, *HMTRL* and *DeepGTT* as baselines. In the supervised methods, we define a primary and a secondary task: A supervised model is trained on the primary task, and then the learned path representation is applied to the secondary task directly. Thus, we have two

experimental settings: 1) *Baseline-PR*, travel-time estimation is the primary task and path ranking is the secondary task; 2) *Baseline-TTE*, where path ranking is the primary task and travel-time estimation is the secondary task. The results are reported in Table D.9. We first observe that *WSCCL* achieves the best performance on both downstream tasks. Further, we observe that the performance of *PathRank* and *HMTLR* are always better on the primary task than on the secondary task. For example, for travel-time estimation on Aalborg, *PathRank-PR* is better than *PathRank-TTE*. This is evidence of the drawbacks of supervised approaches that task-specific TPRs do not generalize well across tasks. Moreover, we also observe that *DeeGTT-PR* is always better than *DeeGTT-TTE* on both data sets. This is because *DeepGTT* is designed to do travel time distribution estimation. Given the task of path ranking, whose distribution may not follow the same inverse-Gaussian distribution like in travel time, so it fails in this case.

Parameter Studies

We study the effects of λ and N.

Effects of λ To study the effect of the balancing factor λ (cf. Eq. D.12), we conduct a parameter study on Aalborg. Based on the results reported in Table D.10, we see that the performance of our model changes when varying λ. We can also see that the optimal λ is 0.8, which means that both global WSC loss and local WSC loss can contribute to the model's performance. When $\lambda = 0.0$, the global WSC loss is ignored, which yields poor performance. When $\lambda = 1.0$, the local contrastive loss is ignored, and the best performance is not obtained, although the performance is quite good. When $\lambda > 0$, meaning that we consider both weakly-supervised and local WSC loss, we observe that the prediction performance is improved. Overall, we conclude that global WSC loss is more important than the local loss.

Effects of N To study the effect of varying the number of *Experts* N, which is also the number of curriculum stages as we always set $N = M$, in the curriculum strategy, we observe the performance for values of N in $\{2, 6, 10, 14, 18\}$. The results in Table D.11 indicate that the best performance is obtained for $N = 10$ on Aalborg data set. We also observe that when N is too small, the curriculum strategy is not effective. This occurs because the number of *Experts* is small, meaning that the difficulty scores have more uncertainty and inaccuracy. This can also be the reason why the number of curriculum stages is small in the curriculum sample selection stage, such that the samples in the beginning are difficult to learn. Next, when the N becomes too large, this may cause problems in the curriculum sample selection stage because the diversity

and number of data points in the meta-sets may be too small, resulting in the model overfitting at each stage.

8 Conclusion and Future Work

We study temporal path representation learning using weak labels. We propose a novel weakly supervised contrastive learning method that uses weakly supervised contrastive learning and local constrative loss. Next, we integrate curriculum learning into the method to further enhance its performance. Finally, we report on experiments on three data sets in the settings of three downstream tasks, finding that our proposal achieves significant performance improvements over unsupervised and supervised baselines. In addition, the proposed method can be utilized as a pre-training method to enhance supervised temporal path representation learning. As future work, it is of interest to study how to incorporate additional weak labels such as drivers and vehicle types.

References

[1] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *ICML*, 2009, pp. 41–48.

[2] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah, "Signature verification using a siamese time delay neural network," in *NIPS*. Morgan Kaufmann, 1993, pp. 737–744.

[3] D. Campos, T. Kieu, C. Guo, F. Huang, K. Zheng, B. Yang, and C. S. Jensen, "Unsupervised time series outlier detection with diversity-driven convolutional ensembles," *Proc. VLDB Endow.*, vol. 15, no. 3, pp. 611–623, 2021.

[4] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *CoRR*, vol. abs/1412.3555, 2014.

[5] R. Cirstea, T. Kieu, C. Guo, B. Yang, and S. J. Pan, "EnhanceNet: Plugin neural networks for enhancing correlated time series forecasting," in *ICDE*, 2021, pp. 1739–1750.

[6] R. Cirstea, B. Yang, C. Guo, T. Kieu, and S. Pan, "Towards spatio-temporal aware traffic time series forecasting," in *ICDE*, 2022.

[7] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *NIPS*, 2016, pp. 3837–3845.

[8] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *NAACL-HLT*, 2019, pp. 4171–4186.

[9] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *KDD*, 2016, pp. 855–864.

[10] C. Guo, B. Yang, J. Hu, and C. S. Jensen, "Learning to route with sparse trajectory sets," in *ICDE*, 2018, pp. 1073–1084.

[11] C. Guo, B. Yang, J. Hu, C. S. Jensen, and L. Chen, "Context-aware, preference-based vehicle routing," *VLDB J.*, vol. 29, no. 5, pp. 1149–1170, 2020.

[12] R. D. Hjelm, A. Fedorov, S. Lavoie-Marchildon, K. Grewal, P. Bachman, A. Trischler, and Y. Bengio, "Learning deep representations by mutual information estimation and maximization," in *ICLR*, 2019.

[13] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[14] ——, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[15] J. Hu, C. Guo, B. Yang, and C. S. Jensen, "Stochastic weight completion for road networks using graph convolutional networks," in *ICDE*, 2019, pp. 1274–1285.

[16] J. Hu, B. Yang, C. Guo, and C. S. Jensen, "Risk-aware path selection with time-varying, uncertain travel costs: a time series approach," *VLDB J.*, vol. 27, no. 2, pp. 179–200, 2018.

[17] J. Hu, B. Yang, C. Guo, C. S. Jensen, and H. Xiong, "Stochastic origin-destination matrix forecasting using dual-stage graph convolutional, recurrent neural networks," in *ICDE*, 2020, pp. 1417–1428.

[18] Y. Huang, Y. Wang, Y. Tai, X. Liu, P. Shen, S. Li, J. Li, and F. Huang, "Curricularface: Adaptive curriculum learning loss for deep face recognition," in *CVPR*, 2020, pp. 5900–5909.

[19] M. G. Kendall, "A new measure of rank correlation," *Biometrika*, vol. 30, no. 1/2, pp. 81–93, 1938.

[20] P. Khosla, P. Teterwak, C. Wang, A. Sarna, Y. Tian, P. Isola, A. Maschinot, C. Liu, and D. Krishnan, "Supervised contrastive learning," in *NeurIPS*, 2020, pp. 18 661–18 673.

[21] T. Kieu, B. Yang, C. Guo, R. Cirstea, Y. Zhao, Y. Song, and C. S. Jensen, "Anomaly detection in time series with robust variational quasi-recurrent autoencoders," in *ICDE*, 2022.

[22] T. Kieu, B. Yang, C. Guo, and C. S. Jensen, "Distinguishing trajectories from different drivers using incompletely labeled trajectories," in *CIKM*, 2018, pp. 863–872.

[23] T. Kieu, B. Yang, C. Guo, C. S. Jensen, Y. Zhao, F. Huang, and K. Zheng, "Robust and explainable autoencoders for time series outlier detection," in *ICDE*, 2022.

[24] Y. Kong, L. Liu, J. Wang, and D. Tao, "Adaptive curriculum learning," in *ICCV*, 2021, pp. 5067–5076.

[25] C. Li, J. Ma, X. Guo, and Q. Mei, "Deepcas: An end-to-end predictor of information cascades," in *WWW*, 2017, pp. 577–586.

[26] J. Li, Z. Han, H. Cheng, J. Su, P. Wang, J. Zhang, and L. Pan, "Predicting path failure in time-evolving graphs," in *KDD*, 2019, pp. 1279–1289.

[27] X. Li, G. Cong, A. Sun, and Y. Cheng, "Learning travel time distributions with deep generative model," in *WWW*. ACM, 2019, pp. 1017–1027.

[28] H. Liu, J. Han, Y. Fu, J. Zhou, X. Lu, and H. Xiong, "Multi-modal transportation recommendation with unified route representation learning," *Proc. VLDB Endow.*, vol. 14, no. 3, pp. 342–350, 2020.

[29] H. Liu, C. Jin, B. Yang, and A. Zhou, "Finding top-k optimal sequenced routes," in *ICDE*, 2018, pp. 569–580.

[30] Z. Liu, V. W. Zheng, Z. Zhao, F. Zhu, K. C. Chang, M. Wu, and J. Ying, "Semantic proximity search on heterogeneous graph by proximity embedding," in *AAAI*, 2017, pp. 154–160.

[31] P. Newson and J. Krumm, "Hidden markov map matching through noise and sparseness," in *SIGSPATIAL*, 2009, pp. 336–343.

[32] S. A. Pedersen, B. Yang, and C. S. Jensen, "Anytime stochastic routing with hybrid learning," *PVLDB*, vol. 13, no. 9, pp. 1555–1567, 2020.

[33] ——, "Fast stochastic routing under time-varying uncertainty," *VLDB J.*, vol. 29, no. 4, pp. 819–839, 2020.

[34] Z. Peng, W. Huang, M. Luo, Q. Zheng, Y. Rong, T. Xu, and J. Huang, "Graph representation learning via graphical mutual information maximization," in *WWW*, 2020, pp. 259–270.

[35] L. Shen and Y. Feng, "CDL: curriculum dual learning for emotion-controllable response generation," in *ACL*, 2020, pp. 556–566.

[36] F. Sun, J. Hoffmann, V. Verma, and J. Tang, "Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization," in *ICLR*, 2020.

[37] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *NIPS*, 2017, pp. 5998–6008.

[38] P. Velickovic, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep graph infomax," in *ICLR*, 2019.

[39] C. Wang, Y. Wu, S. Liu, M. Zhou, and Z. Yang, "Curriculum pre-training for end-to-end speech translation," in *ACL*, 2020, pp. 3728–3738.

[40] Y. Wang, W. Gan, J. Yang, W. Wu, and J. Yan, "Dynamic curriculum learning for imbalanced data classification," in *ICCV*, 2019, pp. 5016–5025.

[41] X. Wu, D. Zhang, C. Guo, C. He, B. Yang, and C. S. Jensen, "AutoCTS: Automated correlated time series forecasting," *Proc. VLDB Endow.*, vol. 15, no. 4, pp. 971–983, 2022.

[42] Z. Wu, Y. Xiong, S. X. Yu, and D. Lin, "Unsupervised feature learning via non-parametric instance discrimination," in *CVPR*, 2018, pp. 3733–3742.

[43] B. Xu, L. Zhang, Z. Mao, Q. Wang, H. Xie, and Y. Zhang, "Curriculum learning for natural language understanding," in *ACL*, 2020, pp. 6095–6104.

[44] B. Yang, J. Dai, C. Guo, C. S. Jensen, and J. Hu, "PACE: a path-centric paradigm for stochastic path finding," *VLDB J.*, vol. 27, no. 2, pp. 153–178, 2018.

[45] S. B. Yang, C. Guo, J. Hu, J. Tang, and B. Yang, "Unsupervised path representation learning with curriculum negative sampling," in *IJCAI*, 2021, pp. 3286–3292.

[46] S. B. Yang, C. Guo, and B. Yang, "Context-aware path ranking in road networks," *IEEE TKDE*, 2020.

[47] B. Yu, H. Yin, and Z. Zhu, "Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting," in *IJCAI*, 2018, pp. 3634–3640.

[48] H. Yuan, G. Li, Z. Bao, and L. Feng, "Effective travel time estimation: When historical trajectories over road networks matter," in *SIGMOD*, 2020, pp. 2135–2149.

[49] J. H. Zar, "Significance testing of the spearman rank correlation coefficient," *Journal of the American Statistical Association*, vol. 67, no. 339, pp. 578–580, 1972.

[50] B. Zheng, Q. Hu, L. Ming, J. Hu, L. Chen, K. Zheng, and C. S. Jensen, "Soup: Spatial-temporal demand forecasting and competitive supply," *TKDE*, 2021.

Paper E

LightPath: Lightweight and Scalable Path Representation Learning

Sean Bin Yang, Jilin Hu, Chenjuan Guo, Bin Yang, and Christian S. Jensen

The paper has been submitted for publication.

The layout has been revised.

Abstract

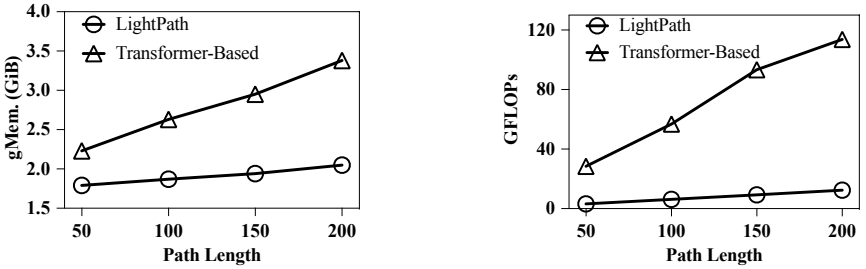
Movement paths are used widely in intelligent transportation and smart city applications. To serve such applications, path representation learning aims to provide compact representations of paths that enable efficient and, still, accurate operations when used for different downstream tasks such as path ranking and travel cost estimation. In many cases, it is attractive that the path representation learning is lightweight and scalable; in resource-constrained environments, it is essential. Yet, existing path representation learning studies focus on accuracy and pay at most secondary attention to resource consumption and scalability.

We propose a lightweight and scalable path representation learning framework, termed **LightPath***, that aims to reduce resource consumption and achieve scalability without affecting accuracy, thus enabling broader applicability. More specifically, we first propose a sparse auto-encoder that ensures that the framework achieves good scalability with respect to path length. Next, we propose a relational reasoning framework to enable faster training of more robust sparse path encoders. We also propose global-local knowledge distillation to further reduce the size and improve the performance of sparse path encoders. Finally, we report extensive experiments on two real-world datasets to offer insight into the efficiency, scalability, and effectiveness of the proposed framework.*

1 Introduction

Motivated in part by an increasing number of intelligent transportation and smart city services that operate on movement paths, path representation learning (PRL) has received remarkable attention [5, 7, 41]. Specifically, a variety of intelligent transportation services involve paths, e.g., travel cost estimation [13, 24, 32, 33, 39, 40], routing [8, 14, 20, 29, 31], trajectory analysis [9, 22, 23, 28, 30] and path ranking [16, 33, 35]. Thus, path representations that are both accurate and compact, thus facilitating efficient operations, are in high demand as they hold the potential to significantly improve the services that use them. Indeed, recent path representation learning methods, in particular deep learning based methods, demonstrate impressive and state-of-the-art performance on a wide variety of downstream tasks.

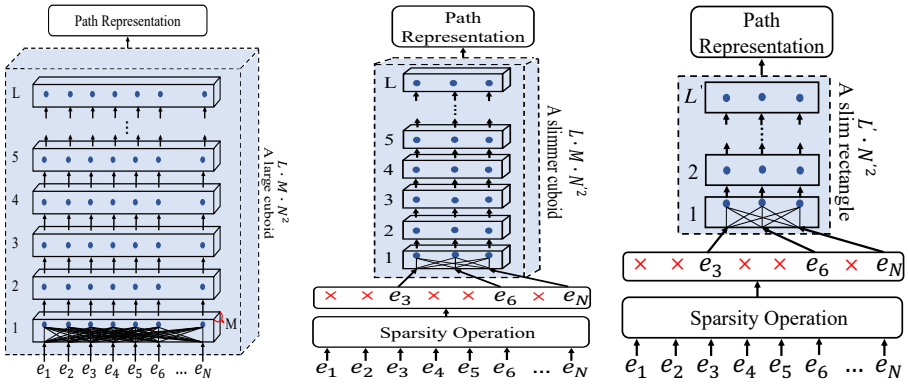
However, existing path representation learning methods focus on accuracy improvement and pay secondary attention at best to scalability and resource usage. The resulting models often include large numbers of layers and parameters, driving up computational costs and memory consumption, especially for long paths. For instance, a model with many parameters may achieve good accuracy, but cannot be used in resource-restricted environments, e.g., on edge devices. On the other hand, increasingly more users access intelligent transportation services through edges devices, such as mobile phones.



(a) GPU Memory **(b)** GFLOPs

Figure E.1: Scalability w.r.t. Path Length.

Existing path representation methods suffer from two limitations.



(a) Traditional Transformer Encoder **(b)** Sparse Transformer Encoder **(c)** LightPath

Figure E.2: Encoder Architectures: (a) A traditional transformer encoder with L layers and M heads, takes as input a path (N-Length) and thus has complexity $\mathcal{O}\left(L \cdot M \cdot N^2\right)$; (b) A sparse transformer encoder takes as input a sparse path (i.e., reducing path length from N to N'), resulting in $\mathcal{O}\left(L \cdot M \cdot N'^2\right)$ complexity; (c) *LightPath* further compresses the traditional transformer in terms of layers and heads, yielding complexity $\mathcal{O}\left(L' \cdot N'^2\right)$, making it more scalable and lightweight than a traditional transformer encoder.

Poor scalability w.r.t. path length Since a path is a sequence of road-network edges, path representation learning benefits from models that are good at capturing sequential relationships, such as the Transformer [26]. However, a Transformer-based method [3] employs a self-attention mechanism, where one edge attends to all other edges in a path in each attention, resulting in quadratic complexity, $\mathcal{O}\left(N^2\right)$ of path length N. As a result, models such as this one exhibits poor scalability with respect to path length, where the path length is the number of edges in a path. Figure E.1 gives an example of

Table E.1: Model Parameter Size with Varying Encoder Layers

Encoder Layers L	12	24	48	96
Parameters (Millions)	29.85	55.07	105.51	206.40

the scalability w.r.t. path length N, covering both memory consumption and computational cost, in terms of GPU memory (gMem.) and Giga floating point operations per second (GFLOPs), respectively. We observe when the path length N increases from 50 to 200, the Transformer-based method performs poorly. A method that scales better w.r.t. N is desirable.

Very large model size. Many existing PRL models have a large number of parameters, which limits their use in resource-constrained environments. For example, in a Transformer-based method [3], where the Transformer stacks L transformer layers, each layer employs multi-head (i.e., M heads) attentions. Thus, the Transformer functions like a large cuboid, with a total complexity of $\mathcal{O}\left(L \cdot M \cdot N^2\right)$, as shown in Figure E.2a. For example, Table E.1 shows the numbers of parameters of Transformer-based path encoders when varying the number of layers among 12, 24, 48, and 96 while fixing the number heads at 8 per layer and the feature dimension of the encoder at 512. We can observe that the model parameters grow dramatically when the number of encoder layers increase, preventing the models from being deployed in resource-constrained environments. Moreover, models with large amounts of parameters also suffer from high storage and computational costs. Such costs are unattractive, particularly in resource-limited environments. More specifically, as shown in Figure E.1a, for path length $N = 200$, the Transformer-based model needs almost 3.4GiB GPU memory. A lightweight path representation learning method that is accurate and efficient is desirable.

Proposed Solution. To tackle the above limitations, we propose *LightPath*, a lightweight and scalable path representation learning approach. To address the first limitation, we first propose a sparse auto-encoder targeting good scalability, w.r.t., path length. In particular, the introduction of sparseness reduces the path length from N to N' by randomly removing edges and returning a sparse path of length N'. The sparse path is fed into a Transformer-based encoder as input, which reduces the complexity from $\mathcal{O}\left(L \cdot M \cdot N^2\right)$ to $\mathcal{O}\left(L \cdot M \cdot N'^2\right)$, as shown in Figure E.2b, reducing a huge cuboid to a slimmer cuboid. To avoid information loss due to the removed edges, we connect the encoder with a decoder, with the aim of reconstructing the whole path. This enables scalable yet effective unsupervised training. To further improve the training of the sparse encoder, we introduce an additional training scheme based on relational reasoning. In particular, for each path p_i, we construct two

distinct sparse path views, denoted as p_i^1 and p_i^2, using different reduction ratios, e.g., removing 40% and 80%, respectively. Then, we propose a dual sparse path encoder, including the original main encoder, and an additional, auxiliary encoder. Then, the dual sparse path encoder encodes the two path views. Thus, we achieve four path presentations PR_i^1, PR_i^2, $\hat{PR}_i^1$, and $\hat{PR}_i^2$ for path p_i in terms of the two path views and the twosparse path encoders, where PR and $\hat{PR}$ denote the representations from the main and the auxiliary encoders, respectively. Finally, given two path representations, we train a relational reasoning network to determine whether the two path representations are from the same "relation" or not. If they are from the same path, we consider them positive relations; otherwise, they belong to negative relations.

To address the second limitation, we propose a global-local knowledge distillation framework that aims to reduce the model size of the main path encoder, which not only enables use in resource-constrained environments but also improves accuracy. To this end, we consider the main path encoder as a teacher, and we create a lightweigth sparse encoder with less layers and one head as a student, further reducing a slimmer cuboid to a slim rectangle (cf. FigureE.2c). The global knowledge distillation tries to push the lightweight student to mimic the teacher from a global semantic level (i.e., path representation level), while the local knowledge distillation can push the lightweight student to mimic the edge correlations from the teacher, thus building a lightweight encoder while maintaining or even improving the accuracy for downstream tasks.

To the best of our knowledge, this is the first study that systematically targets lightweight and scalable path representation learning. This study makes four main contributions.

- *Sparse Auto-encoder.* We propose a sparse auto-encoder framework that takes as input sparse paths with reduced path lengths, achieve good scalability. w.r.t the path length.
- *Relational Reasoning.* We introduce relational reasoning to enable efficient sparse auto-encoder training. Specifically, we propose two types of relational reasoning objective for accurate and efficient path representation learning. These two objectives regularize each other and yield a more effective path encoder.
- *Global-local Knowledge Distillation.* We propose a novel global-local knowledge distillation framework that enables a lightweight student sparse encoder to mimic a larger teacher sparse encoder from global and local perspectives. The resulting lightweight model that can be deployed on edge devices while achieving accurate performance at different downstream tasks.

- *Extensive Experiments.* We report on extensive experiments for two large-scale, real-world datasets with two downstream tasks. The results offer evidence of the efficiency and scalability of the proposed framework as compared with nine baselines.

2 Related Work

2.1 Path Representation Learning

Path Representation Learning (PRL) aims to learn effective and informative path representations in road network that can be applied to various downstream applications, i.e., routing, travel cost estimation, and path recommendation. Existing PRL studies can be categorised as supervised learning based [16, 35, 37], unsupervised learning based [33], and weakly supervised learning based [34] approaches. Supervised learning-based methods aim at learning a task-specific path representation with the availability of large amounts of labeled training data [16, 35, 37], which has a poor generality for other tasks. Unsupervised learning methods are to learn general path representation learning that does not need labeled training data and generalizes well to multiple downstream tasks [33, 34]. Take the cream and discard the dross, weakly supervised learning methods try to learn a generic path representation based on meaningful weak labels, e.g., traffic congestion indices, that are easy and inexpensive to obtain, and are relevant to different tasks [34]. However, these methods are computationally expensive and hard to deploy in resource-limited environments.

2.2 Self-supervised Learning

Self-supervised learning (SSL) tries to learn informative representations without the availability of labels through well-defined pretext tasks. State-of-the-art SSL can be classified into contrastive learning-based and relation reasoning-based methods. Contrastive learning-based methods [2, 12, 25, 27, 33], especially for InfoNCE loss-based, commonly generate different views of same input data through different augmentation strategies, and then discriminate positive and negative samples. However, these methods suffer from their quadratic complexity, w.r.t. the number of data samples, given that it needs a large number of negative samples to guarantee that the mutual information lower bound is tight enough [12]. In contrast, relation reasoning-based methods [4, 19] aim to learn relation reasoning head that discriminates how entities relate to themselves and other entities, which results in linear complexity. However, existing studies construct relation reasoning between different views from the same encoder, ignoring the effect of different views between differ-

ent encoders, i.e., main encoder and auxiliary encoder in Siamese encoder architecture.

3 Preliminaries

We first cover important concepts that underlie the paper's proposal and then state the problem addressed.

3.1 Definitions

Road Network. A road network is defined as a graph $\mathbf{G} = (V, E)$, where V is a set of vertices v_i that represents road intersections, $E \subseteq V \times V$ represents a set of edges $e_i = (v_j, v_k)$ that denotes road segments. **GPS Trajectory.** A GPS trajectory of a moving object is defined as a sequence of spatio-temporal sample points, each of which contains a location (i.e., longitude and latitude) and a timestamp.

Path. A path $p = \langle e_1, e_2, e_3, \cdots, e_N \rangle$ is a sequence of connected edges, where $e_i \in E$ denotes the edge in path p. We denote $p.\Phi = \langle 1, 2, 3, \cdots, N \rangle$ as a sequence of orders for edges in p. **Sparse Path.** A sparse path $p' = \{e_i\}_{i \in p'.\Omega}$ contains a subset of edges in path p, where $p'.\Omega$ is a sub-sequence of $p.\Phi$, which is denoted as the original edge orders of p' in p, where $p'.\Omega \subseteq p.\Phi$. **Example.** Given a path $p = \langle e_1, e_3, e_4, e_6, e_7 \rangle$ and $p.\Phi = \langle 1, 2, 3, 4, 5 \rangle$, then sparse path $p' = \{e_1, e_4, e_7\}$, where $p'.\Omega = \langle 1, 3, 5 \rangle$, is one of the sparse paths for p.

Edge Representation. The edge representation of an edge in a road network graph is a vector in $\mathbb{R}^{d_k}$, where d_k is the dimensionality of the vector. **Transformer Layer.** Given a sequence of edge representations $\mathbf{X} = \langle x_1, x_2, x_3, \cdots, x_N \rangle$ for a path p. Transformer layer takes as input a $\mathbf{X}$ and returns the encoded edge representations $\mathbf{Z} = \langle z_1, z_2, z_3, \cdots, z_N \rangle$ that capture the correlation of different edges. Especially, each Transformer layer consists of multi-head attention and position-wise feed-forward networks.

Multi-Head Attention. Instead of employing a single attention function, we define multi-head attention that linearly projects the queries, keys and values into M subspaces with different, learned linear projections to d_k, d_k and d_v dimensions, respectively. Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions. Then, we formulate it as:

$$\begin{aligned} \mathbf{Z} = \mathbf{MultiHead}(\mathbf{X}) &= \text{Concat}\left(\text{ head }_1, \ldots, \text{ head }_M\right) \cdot \mathbf{W}^O \,, \\ \text{head }_i(\cdot) &= \text{softmax}\left(\left(\mathbf{X}\mathbf{W}_i^Q\right)\left(\mathbf{X}\mathbf{W}_i^K\right)^T / \sqrt{d_k}\right)\left(\mathbf{X}\mathbf{W}_i^V\right) \,, \end{aligned} \tag{E.1}$$

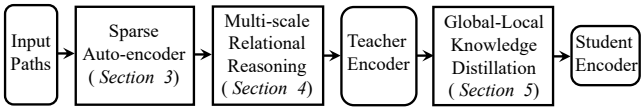


Figure E.3: Solution Overview.

where Concat$(\cdot,\cdot)$ represents concatenation. $\mathbf{W}_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $\mathbf{W}_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $\mathbf{W}_i^V \in \mathbb{R}^{d_{model} \times d_v}$, $\mathbf{W}^O \in \mathbb{R}^{Md_v \times d_{model}}$ are projections parameter matrices for scaled dot-product attention. M denotes number of heads. d_{model} represents the feature dimension of final output.

Position-wise Feed-Forward Networks. Except the attention sub-layers, each of the layers in Transformer (encoder/decoder) contains a fully connected feed-forward network (FFN), which is used to each position separately and identically. This FFN consists of two linear transformations with ReLU activation in between. Specifically, we have

$$\text{FFN}(\mathbf{Z}) = \max\left(0, \mathbf{Z}\mathbf{W}_\mathbf{1}^{\mathbf{FFN}} + \mathbf{b}_\mathbf{1}^{\mathbf{FFN}}\right)\mathbf{W}_\mathbf{2}^{\mathbf{FFN}} + \mathbf{b}_\mathbf{2}^{\mathbf{FFN}}, \tag{E.2}$$

where $\mathbf{W}_\mathbf{1}^{\mathbf{FFN}}$, $\mathbf{W}_\mathbf{2}^{\mathbf{FFN}}$, $\mathbf{b}_\mathbf{1}^{\mathbf{FFN}}$, and $\mathbf{b}_\mathbf{2}^{\mathbf{FFN}}$ are learnable parameters of feed-forward network.

3.2 Problem Definition

Given a set of paths $\mathbb{P} = \{p_i\}_{i=1}^{|\mathbb{P}|}$ in a road network G, scalable and efficient path representation learning aims to learn a function $SEPRL_\theta(\cdot)$ which can generate a generic path representation for each path $p_i \in \mathbb{P}$ without relying on the labeling information, which can be formulated as follows.

$$PR = SEPRL_\theta(p_i) : \mathbb{R}^{N \times d_k} \rightarrow \mathbb{R}^d, \tag{E.3}$$

where PR is learned path representation. θ represents the learnable parameters for the sparse path encoder. N is path length, d_k and d are the feature dimensions for an edge, and a final path representation, respectively.

3.3 Solution Overview

Figure E.3 shows an overview of the proposed *LightPath*, which consists of the following three modules: 1) Sparse Auto-encoder, 2) Relational reasoning, and 3) Global-local knowledge distillation. The details of those modules are provided in Section 3, 4 and 5, respectively. To create *LightPath*, we first train a large teacher encoder that has multiple layers and heads based on sparse auto-encoder and relational reasoning. Then, we employ global-local

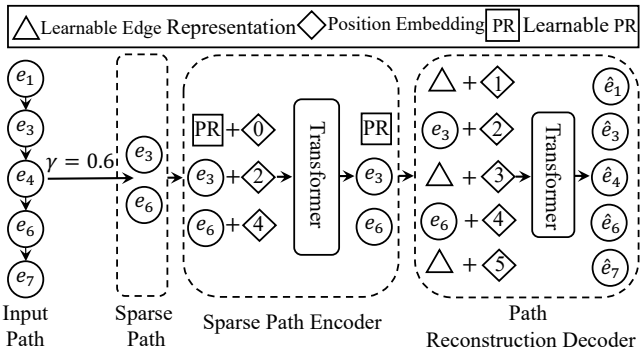


Figure E.4: Sparse Auto-encoder. We remove a subset of edges from a path based on a reduction ratio γ to obtain a sparse path. We introduce a learnable path representation in front of the sparse path. And then, we fed the resulting sparse path vectors with position embeddings to a Transformer based encoder. We then introduce a learnable edge representation, denoted as a triangle, to represent the removed edges. The encoded edges in the sparse path and the removed edge representations with position embeddings are processed by a decoder that reconstructs the edges in the original path.

knowledge distillation to compress the large teacher encoder to achieve a small student encoder that has much less layers and heads.

4 Sparse Path Encoder

4.1 Overview

Figure E.4 illustrates the sparse path encoder framework, which includes a sparsity operation, a sparse path encoder, and a path reconstruction decoder. The sparsity operation takes as input a full path and returns a sparse path with respect to reduction ratio γ. Sparse path encoder takes as input a sparse path and learnable path representation and outputs learned path representations. Next, we introduce a path reconstruction decoder to reconstruct the path, thus ensuring the learned path representation captures the entire path information.

4.2 Sparsity Operation

Path consists of a sequence of edges $p = \langle e_1, e_2, e_3, \cdots, e_N \rangle$, which are the basic processing units of different sequential models. The processing times of sequential models become longer when the path gets longer. Thus, we propose a sparsity operation, which is an approach to reduce the path length from N to N', where N' is much less than N. Specifically, we conduct path reduction by randomly removing a subset of edges in a path based on a high reduction ratio γ (e.g., $\gamma = 0.6$). A high reduction ratio γ (the ratio for edge

removal) can significantly reduce the length of each input path, thus enabling the scalability of the path. Specifically, we construct the sparsity operation as:

$$p' = f(p, \gamma) = \{e_j\}_{j \in \Omega}, \tag{E.4}$$

where p is input path. p' denotes the sparse path. For example, as shown in Figure E.4, if we have a path $p = \langle e_1, e_3, e_4, e_6, e_7 \rangle$, then we conduct sparsity operation, which randomly removes a subset of edges in p, i.e., e_1, e_4, e_7 based on reduction ratio $\gamma = 0.6$ and achieve the sparse path $p' = \{e_3, e_6\}$ and $p'.\Omega = [2, 4]$. Thus, we can reduce path from N to N', i.e., from 5 to 2 in this example.

4.3 Learnable Path Representation

We use Transformer as our path encoder since it processes the input edges parallel using the self-attention mechanism. In contrast, the recurrent neural network (RNN) family is inefficient due to its recurrent nature. To avoid achieving path representation through extra aggregation function [33], we add a super extra learnable path representation in front of each sparse path. Moreover, PR is attached to position 0 for every path, thus enabling it to capture global information of the path during the training procedure. Thus, we update the p' as:

$$p' = \{PR\} + \{e_j\}_{j \in \Omega} = \{e_k\}_{k \in \Omega'}, \tag{E.5}$$

where $e_0 = PR$ denotes a virtual edge and $\Omega' = [0, \Omega]$.

To preserve the sequential information of the path, we add learnable position embedding into the sparse path representations based on order information in Ω'. Specifically, we have:

$$\mathbf{X} = \text{Concat}\{x_k\}_{k \in \Omega'}, \text{ where } x_k = e_k + pos_k, \tag{E.6}$$

where pos_k represents the learnable position embedding for edges in the sparse path. $\mathbf{X}$ represents the sparse path edge representation concatenation.

Take Figure E.4 as an example, we first construct $p' = \{PR, e_3, e_6\}$. Then, we add corresponding position embedding to the edge vectors of p', i.e., positions 0, 2, and 4, where the added position embeddings can help the Transformer encoder to be aware of the input order instead of treating them as a set of unordered path edges. Meanwhile, they enable the learned path representation PR to capture global-level semantics in the sense that edges might play a different role in a road network. The intuition is that the super learnable path representation can attend attention with other edges, which captures global-level semantic. In contrast, the learnable edge representation aims to construct a full path set and reconstruct the specific edge in input path.

4.4 Transformer Path Encoder

To achieve better performance, we usually stack multiple Transformer layers, each consisting of two sub-layers: multi-head attention and position-wise feed-forward network mentioned above (ref. as to Definition 3.1 in Section **??**). Motivated by [11], we employ a residual connection around each sub-layers, followed by layer normalization [1]. The stacked transformer model can be formulated as:

$$\begin{aligned} Z &= \text{LayerNorm}(\mathbf{X} + \text{ MultiHead } (\mathbf{X})) \,, \\ PR &= \text{LayerNorm}\left(Z + \text{FFN}\left(Z\right)\right) \,, \end{aligned} \tag{E.7}$$

where LayerNorm represents layer normalization and PR is learned path representation.

Remarkably, our path encoder only takes as input a small subset of edges (e.g., 60%) of the full path edges, which means we ignore the removed edges and just consider unremoved edges during the encoder stage to enable the path scalability. Path scalability enables us to train our path encoders concerning different lengths of path effectively and reduce the corresponding computational cost and memory usage.

4.5 Path Reconstruction Decoder

To capture the global information of the full path, we further introduce a lightweight path decoder to reconstruct the removed edges in a path. As shown in Figure E.4, we first complement the encoded path edges and path representation with a shared, learnable vector that represents the presence of a removed edge based on the original index of each edge in a path. Then, we add the position embedding vectors to all edge representation, which enables the learnable path representation vector to capture the global information of the entire path. Next, the path decoder takes as input the full set of representations, including (1) path representation, (2) encoded unremoved edges, and (3) removed edges. We select a more lightweight decoder structure, which has less number of Transformer layers. Since the path decoder is only used to perform path reconstruction, the architecture of our path decoder can be more flexible and independent of the path encoder. Thus, the decoder is much shallower than the encoder, e.g., one layer for the decoder and 12 layers for the encoder, which significantly reduces training time. We reconstruct the input path by predicting the removed edges to ensure the learned path representation contains complete information about the entire path. We employ mean squared error (MSE) as our reconstruction loss function and compute MSE between the reconstructed and initial edge representations in the edge level. We only employ MSE loss on removed edges, which can be

formulated as follows:

$$\mathcal{L}_{rec} = \frac{1}{N}\sum_{i=1}^{N}(e_i - \hat{e}_i)^2 \ , \tag{E.8}$$

where e_i and $\hat{e}_i$ are the initial and predicted removed edge representation, respectively. N represents the number of edges for each input path.

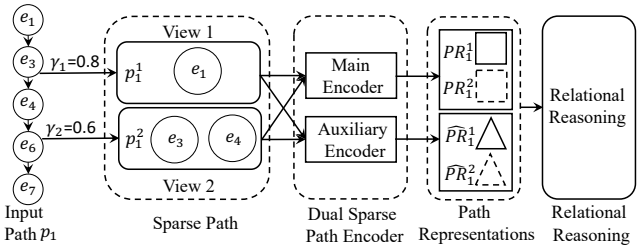
5 Relational Reasoning Path Representation Learning

5.1 Overview

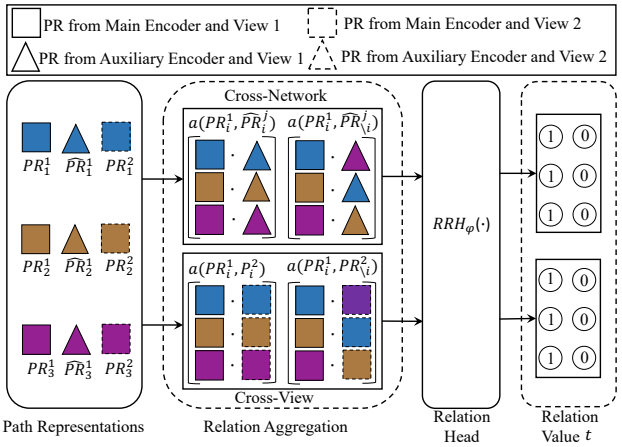
To further enhance sparse auto-encoder (cf. Section **??**) training, we propose a novel self-supervised relational reasoning (RR) framework, as shown in Figure E.5. The intuition behind this is that we train a relation head $RRH_{\varphi}(\cdot)$ to discriminate how path representations relate to themselves (same class) and other paths (different class). In particular, this framework consists of path representation construction (cf. Figure E.5a) and relational reasoning (cf. Figure E.5b), which includes cross-network relational reasoning and cross-view relational reasoning. To train our dual sparse auto-encoder, we first generate two path views, denotes as p_1^1 and p_1^2, based on two different reduction ratios γ_1 and γ_2. After this, by processing these two path views via the main encoder and the auxiliary encoder of the sparse path encoder, we construct different paths on multiple views in the representation space. Finally, we employ relational reasoning to enable efficient path representation learning.

5.2 Dual Sparse Path Encoder

In this section, we introduce our dual sparse path encoder (SPE) that is employed to generate different path representations based on different path views. As shown in Figure E.5a, given a path p_1, we first generate sparse paths in terms of two different reduction ratios γ_1 and γ_2. We consider them as different path views, i.e., path view 1 and path view 2. Then, our dual sparse path encoder, including a main encoder and an auxiliary encoder, takes as input two different path views (i.e., p_1^1 and p_1^2) and returns different path representations. Specifically, each encoder takes as input two different path views and returns two different path representations, where solid and dotted $\square$ denote the path representations returned from main encoder based on path view 1 and path view 2, respectively, i.e., PR_1^1 and PR_1^2. In contrast, solid and dotted $\triangle$ represent the path representations achieved from auxiliary encoder



(a) Path Representation Construction Using Dual Sparse Path Encoders



(b) Relational Reasoning

Figure E.5: Illustration of RR Training: (a) Given an input path p_1, we construct two path views (i.e., p_1^1 and p_1^2) through two reduction ratios γ_1 and γ_2, based on which a main encoder and an auxiliary encoder are employed to generate path representations for each view (i.e., PR_1^1, PR_1^2, $\hat{PR}_1^1$, and $\hat{PR}_1^2$). (b) After getting corresponding path representations for paths in a minibatch, a relational reasoning path representation learning scheme, which utilizes both cross-network and cross-view relational reasoning modules, is deployed. In particular, for both modules, an aggregation function a joins positives (representations of the same paths, e.g., $a(PR_1^1, \hat{PR}_1^1)$, $a(PR_1^1, PR_1^2)$) and negatives (randomly paired representations, e.g., $a(PR_1^1, \hat{PR}_3^1)$, $a(PR_1^1, PR_3^2)$) through a commutative operator. Then relation head module $RRH_\varphi(\cdot)$ estimates the relation score y, which must be 1 for positive and 0 for negatives. Both cross-network and cross-view objectives are optimized minimizing the binary cross-entropy (BCE) between prediction and target relation value t. In this example, $i \in [1, 2, 3]$ denotes the number of paths in the minibatch and $j \in [1, 2]$ represents the number of views.

based on both path views, respectively, i.e,. $\hat{PR}_1^1$ and $\hat{PR}_1^2$. To this end, we construct four different path representations for a given path, which promote our design of cross-network relational reasoning and cross-view relational reasoning in turn. Finally, we formulate it as:

$$PR_i^j = SPE_\theta(p_i^j, \gamma) \ , \ \hat{PR}_i^j = SPE_{\hat{\theta}}(p_i^j, \gamma) \ , \tag{E.9}$$

where PR_i^j and $\hat{PR}_i^j$ are path representations obtained from the main encoder and the auxiliary encoder, respectively. p_i denotes the i-th path in the path set. $j \in [1, 2]$ denotes the path views. θ and $\hat{\theta}$ are the parameters for the main encoder and auxiliary encoder.

5.3 Relational Reasoning

Cross-Network Relational Reasoning

In *LightPath*, we employ a dual sparse path encoder, which includes main and auxilary encoder, as shown in Figure E.5a. We first construct path representations through sparsity operation based on different reduction ratios γ_1 and γ_2. Given a set of path $\{p_1, p_2, \cdots, p_K\}$, we can have a set of path representations $\{PR_1^1, PR_2^1, \cdots, PR_K^1\}$ from main encoder and $\{\hat{PR}_1^1, \hat{PR}_2^1, \cdots, \hat{PR}_K^1\}$ or $\{\hat{PR}_1^2, \hat{PR}_2^2, \cdots, \hat{PR}_K^2\}$ from auxiliary encoder by using path representation construction. Then we employ relation aggregation $a(\cdot)$ that joins the positive path representation relations $\langle PR_i^1, \hat{PR}_i^1 \rangle$ or $\langle PR_i^1, \hat{PR}_i^2 \rangle$ and the negative path representation relations $\langle PR_i^1, \hat{PR}_{\setminus i}^1 \rangle$, where i denotes the i-th path sample and $\setminus i \neq i$ represents randomly selected path representations in a minibatch. Take Figure E.5b as an example, where $K = 3$. we join $\langle PR_1^1, \hat{PR}_1^1 \rangle$ as a positive relation pair (representation from same path), and $\langle PR_1^1, \hat{PR}_2^1 \rangle$ as a negative relation pair (representation from different paths) through aggregation function a. Next, the relational head $RRH_\varphi(\cdot)$, which is non-linear function approximator parameterized by φ, takes as input representation relation pairs of cross-network and returns a relation score y. Finally, we formulate the cross-network relational reasoning task as a binary classification task, where we use binary cross-entropy loss to train our sparse path encoder, which is given as follows.

$$\begin{aligned} \mathcal{L}_{cn} = \underset{\boldsymbol{\theta},\varphi}{\operatorname{argmin}} \sum_{i=1}^{K} \sum_{j=1}^{2} \mathcal{L}\left(RRH_\varphi\left(a\left(PR_i^j, \hat{PR}_i^j\right)\right), t = 1\right) \\ + \mathcal{L}\left(RRH_\varphi\left(a\left(PR_i^j, \hat{PR}_{\setminus i}^j\right)\right), t = 0\right) \ , \end{aligned} \tag{E.10}$$

where K is the the number of path samples in the minibatch. $a(\cdot, \cdot)$ is an aggregation function. $\mathcal{L}$ is a loss function between relation score and a target relation value. t is a target relation values.

The intuition behind this is to discriminate path presentations of same path and different paths, which are from different views across dual sparse path encoder and are able to distill the knowledge from historical observations, as well as stabilizing the main encoder training. To realize this, we adopt

Siamese architecture for our dual sparse path encoder, where the auxiliary encoder does not directly receive the gradient during the training procedure. In contrast, we update its parameters by leveraging the momentum updating principle:

$$\hat{\theta}^t = m \times \hat{\theta}^{(t-1)} + (1-m) \times \theta^t, \tag{E.11}$$

where m is momentum parameter. θ and $\hat{\theta}$ are the parameters of the main encoder and the auxiliary encoder.

Cross-View Relational Reasoning

To enhance the learning ability of our *LightPath*, we further consider the ties between two views within main encoder, which acts as a strong requarization to enhance the learning ability of our methods. We do not have to include such relational reasoning within the auxiliary encoder because it will not directly compute gradient during training, and our goal is to train main encoder. Figure E.5b shows the design of our cross-view relational reasoning, which contains two similar representations from two views based on γ_1 and γ_2. The intuition of cross-view relational reasoning is to preserve the relation between two views of the same path and discriminate them from the view of other paths.

Similar with cross-network, given a set of paths $\left\{p_1, p_2, \cdots, p_K\right\}$. We first achieve two set of path representations in terms of two path views based on main encoder, i.e., $\left\{PR_1^1, PR_2^1, \cdots, PR_k^1\right\}$ and $\left\{PR_1^2, PR_2^2, \cdots, PR_K^2\right\}$. Then, we join the positive relation pairs (e.g., $\langle PR_i^1, PR_i^2 \rangle$) and negative relation pairs (e.g., $\langle PR_i^1, PR_{\setminus i}^2 \rangle$) through aggregation function. For example, as shown in Figure E.5b, there are 3 paths in the set. Thus, we can denote $\langle PR_1^1, PR_1^2 \rangle$ as a positive pair and $\langle PR_1^1, PR_3^2 \rangle$ as a negative pair. Then, we further employ relational head $RRH_\varphi(\cdot)$, which takes as input a positive pair and a negative pairs from different views, to achieve the corresponding relation score y for the cross-view relational reasoning. Last, we formulate the cross-view relational reasoning loss to discriminate how different views of a path is related to themselves and other paths. In this phase, the complete learning objective can be specified as:

$$\begin{aligned} \mathcal{L}_{cv} = \underset{\boldsymbol{\theta},\varphi}{\operatorname{argmin}} \sum_{i=1}^{K} \mathcal{L}\left(RRH_\varphi\left(a\left(PR_i^1, PR_i^2\right)\right), t=1\right) \\ + \mathcal{L}\left(RRH_\varphi\left(a\left(PR_i^1, PR_{\setminus i}^2\right)\right), t=0\right) \ , \end{aligned} \tag{E.12}$$

where K is the the number of path samples in the minibatch.

Objective for *RR*

To train our dual path encoder end-to-end and efficient learn path representations for downstream tasks, we jointly leverage both the cross-network and cross-view relation reasoning loss. Specifically, the overall objective function is formulated as Eq. E.13.

$$\min_{\theta,\varphi} \mathcal{L}_{RR} = \mathcal{L}_{cn} + \mathcal{L}_{cv} \tag{E.13}$$

5.4 LightPath Training

To train our sparse path encoder and learn path representations for downstream tasks, we jointly minimize the reconstruction and RR loss. Specifically, the overall objective function is defined as:

$$\mathcal{L} = \mathcal{L}_{rec} + \mathcal{L}_{RR} \tag{E.14}$$

6 Global Local Knowledge Distillation (GLKD)

So far, we realize our *LightPath* through sparse auto-encoder and relational reasoning and transform it from large cuboid (cf. Figure E.2a to a slim cuboid (cf. Figure E.2b). To enable the *LightPath* that can be deployed on resource-constrained mobile devices, we introduce our global-local knowledge distillation (GLKD) to further reduce the size of the sparse auto-encoder, as shown in Figure E.6. We first train a large cuboid teacher encoder with multiple transformer layers and heads (cf. Figure E.2b) based on path reconstruction and relational reasoning. Then, we employ a small rectangle student encoder (cf. Figure E.2c), which has less layers and heads, to mimic a large teacher model and use the teacher's knowledge to obtain similar or superior performance based on GLKD. Specifically, GLKD constructs a local knowledge distillation by matching the representations of correlated edges. On such a basis, the global term distills the knowledge from teacher to student that enabling the informative and powerful path representation for the student model.

6.1 Global-path Representation Distillation

Given a path $p_i = \langle e_1, e_2, e_3, \cdots, e_n \rangle$, where n is the number of edges in a path. We define $PR^{\mathcal{T}}(p_i)$ and $PR^{\mathcal{S}}(p_i)$ represent the path representations achieved from teacher encoder $\mathcal{T}_\theta$ and student encoder $\mathcal{S}_\theta$. The intuition of global path representation knowledge distillation is to let the student encoder mimic the global properties captured by a large cuboid teacher encoder. And

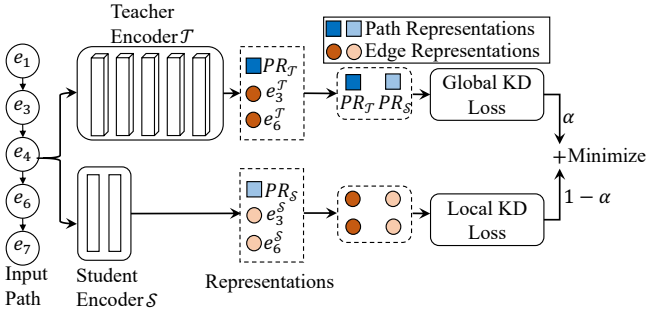


Figure E.6: Illustration of GLKD. Given an input path, we formulate our GLKD as a weighted sum of global path representation knowledge distillation (GPRKD) loss and local edge representation knowledge distillation (LERKD) loss.

thus, the goal of global path representation knowledge distillation is to put closer the path representation from teacher encoder and student encoder in the latent space. We formalize this problem as minimizing a latent space distance representation pairs in terms of the large cuboid teacher encoder and the rectangle student encoder. The formulation of the objective function is given as follows:

$$\min_{\theta} \mathcal{L}_{global}\left(\mathbf{PR}^{\mathcal{T}}(p_i), \mathbf{PR}^{\mathcal{S}}(p_i)\right) = \left\| sp(\mathbf{PR}^{\mathcal{T}}(p_i)/t) - sp(\mathbf{PR}^{\mathcal{S}}(p_i)/t) \right\|^2, \quad \text{(E.15)}$$

where $sp(\cdot)$ is exponential function. t denotes the temperature. Using a higher value for t produces a softer probability distributions over path representations.

6.2 Local-edge Correlation Distillation

The goal of local-edge structure distillation is to preserve the local similarity of the edge correlations in a path. In particular, it is expected that the representation of the same edge in a path represented by the teacher encoder and the student encoder should be close to each other. The intuition is that a rectangle student encoder mimics the edge correlations in a path captured by a large cuboid teacher encoder. Using a similarity measurement, we formulate the local-edge structure distillation problem as minimizing the latent space distance of edge representations from the teacher encoder and then student encoder.

In specific, given a path $p = \langle e_1, e_2, e_3, \cdots e_N \rangle$ in a road network, where N is the number of edges in a path. Through applying an L-layers Transformer encoder (i.e., teacher encoder $\mathcal{T}_\theta$) and L'-layers Transformer encoder (i.e., student encoder $\mathcal{S}_\theta$) upon sparse path p', where $L \ll L'$, the edge representation

that captures spatial dependencies are derived as follows.

$$F^{\mathcal{T}}(e_i)_{i=1}^{N'} = \mathcal{T}_\theta(p) \ , F^{\mathcal{S}}(e_i)_{i=1}^{N'} = \mathcal{S}_\theta(p) \ , \tag{E.16}$$

where $F^{\mathcal{T}}(e_i)_{i=1}^{N'}$ and $F^{\mathcal{S}}(e_i)_{i=1}^{N'}$ represent the edge representation with respect to the teacher and student encoder, respectively.

In this phase, the goal of learning is to reduce the latent space distance between same edge pair from the teacher and student encoder, respectively. To this end, we aim to minimize the following objective functions between edge representation pairs in terms of the parameters of the student encoder.

$$\min_\theta \mathcal{L}_{local}\left(\mathbf{F}^{\mathcal{T}}(e_i), \mathbf{F}^{\mathcal{S}}(e_i)\right) = \frac{1}{n}\sum_{i=1}^{n}\left\| sp(\mathbf{F}^{\mathcal{T}}(e_i)/t) - sp(\mathbf{F}^{\mathcal{S}}(e_i)/t)\right\|^2 , \tag{E.17}$$

where $sp(\cdot)$ represents exponential function. t denotes the temperature. Using a higher value for t produces a softer probability distributions over edges.

6.3 Objective for *GLKD*

To train our global and local knowledge distillation in an end-to-end fashion, we jointly leverage both the global and local knowledge distillation loss. Specifically, the overall objective function to minimize is defined in Eq. E.18.

$$\min_\theta \mathcal{L}_{GLKD} = \alpha * \mathcal{L}_{global} + (1-\alpha) * \mathcal{L}_{local} \ , \tag{E.18}$$

where α is balancing factor.

7 Experiments

7.1 Experimental Setup

Datasets

We conduct experiments on two real-world datasets and one synthetic dataset to enable fair comparisons with existing studies. Based on two real-world datasets, we report results for two downstream tasks: travel time estimation [34, 39], and path ranking [16, 35, 37]. Due to the lack of large amounts of long paths in the real-world datasets, we construct one synthetic dataset that contains paths with lengths of 100, 150, and 200 to verify the efficiency and scalability of *LightPath*.

Aalborg, Europe: We collect the road network of Aalborg from OpenStreetMap[1] that contains 10,017 nodes and 11,597 edges. Specifically, this dataset contains

[1] https://www.openstreetmap.org

180 million GPS records from 183 vehicles sampled at 1 Hz over a two-year period from 2017 to 2018. After map matching [18], we obtain 39,160 paths with length 50.
Chengdu, China: This dataset was collected from Chengdu, China, on Oct. and Nov. 2016. We obtain the corresponding road network from OpenStreetMap. The network contains 6,632 nodes and 17,038 edges. The GPS data was sampled at about 1/4-1/2 Hz. We obtain 50,000 paths through map matching with lengths 50.
Synthetic: We generate paths with lengths of 100, 150, and 200 from the Aalborg Road Network to study the training efficiency and scalability of the *LightPath*.

Downstream Tasks

We report the results on two downstream tasks:
Path Travel Time Estimation: We obtain travel time (in seconds) for each path from the trajectory. We aim to utilize a regression model to predict the travel time based on the learned path representations. We employ Mean Absolute Error(**MAE**), Mean Absolute Relative Error(**MARE**), and Mean Absolute Percentage Error(**MAPE**) to evaluate the performance of travel time estimations. The smaller values of these metrics, the better performance we achieve.
Path Ranking: Each path is assigned a ranking score in the range $[0, 1]$, which is obtained from historical trajectories by following the existing studies [35, 37]. More specifically, we take the path that is used by a driver in the historical trajectories as the trajectory path, which is denoted as the top ranked path. Then, we generate multiple paths connecting the same source and destination via path finding algorithms [17]. Finally, we calculate the similarity between a generated path and the trajectory path as a ranking score. The higher ranking score indicates a generated path is more similar to the trajectory path, and the trajectory path itself has a score of 1 and ranks the highest. To measure the path ranking, we apply **MAE**, the Kendall rank correlation coefficient (τ), and the Spearman's rank correlation coefficient (ρ), which are widely used metrics in path ranking, to evaluate the effectiveness of path ranking.

Models for Downstream Tasks

For all unsupervised learning methods, we aim to build a regression model to estimate the travel time and path ranking. In particular, we select ensemble model *Gradient Boosting Regressor*(GBR) [21] as our prediction model since they are regression problems.

Baselines

We compare *LightPath* with 9 baselines, including 6 unsupervised learning-based methods and 3 supervised learning-based methods. The details of these baseline methods are summarized as follows:

- **Node2vec** [6] is an unsupervised node representation model that learn node representation in a graph. We achieve the path representation by aggregating the node representations of the nodes in a path.
- **MoCo** [10] is a momentum contrast for unsupervised visual representation learning. Here we use momentum contrast to learn path representations.
- **Toast** [3] first uses auxiliary traffic context information to learn road segment representation based on the skip-gram model and then utilizes a stacked transformer encoder layer to train trajectory representation through route recovery and trajectory discrimination tasks. We use the same schema to learn path representations.
- **t2vec** [15] is a trajectory representation learning method for similarity computation based on the encoder-decoder framework, which is trained to reconstruct the original trajectory. We use a sequence of edges in a path to represent a trajectory.
- **NeuTraj** [38] is a method that revised the structure of LSTM to learn representations of the grid in the process of training their framework. To support our task with it, we replace the grid with edges in their framework.
- **PIM** [33] is an unsupervised path representation learning approach that first generates negative samples using curriculum learning and then employs global and local mutual information maximization to learn path representations.
- **HMTRL** [16] is a supervised path representation learning framework for multi-modal transportation recommendation.
- **PathRank** [35] is a supervised path representation learning model based on GRUs, which treats departure time and driver ID as additional information.
- **LightPath-Sup** is a supervised version of our *LightPath*, where we train it in a supervised manner.

Implementation Details

We employ an asymmetrical sparse auto-encoder architecture and randomly initialize all learnable parameters with uniform distributions. In particular, we adopt Siamese architecture, where we update the parameters of the auxiliary encoder based on the momentum updating principle based on the main encoder and we set the momentum parameter $m = 0.99$. We employ node2vec [6] to embed each edge to 128-dimensional vectors and set the dimension for path representation to 128. We select concatenate as the relation aggregation function $a(\cdot, \cdot)$. We use the AdamW optimizer with a cosine decay learning rate schedule over 400 epochs, with a warm-up period of 40 epochs. We set the base learning rate to 1e-3 and betas as $(0.9, 0.95)$. We vary γ from 0.1,0.3,0.5,0.7,0.9 to study the effect of path scalability and efficiency for the *LightPath*. In addition, we consider four different path lengths, i.e., 50, 100, 150, and 200, to study the effectiveness, efficiency, and scalability of the *LightPath*. We then evaluate our *LightPath* as well as all baselines on a powerful Linux server with 40 Intel(R) Xeon(R) W-2155 CPU @ 3.30GHz and two TITAN RTX GPU cards. Finally, all algorithms are implemented in PyTorch 1.11.0.

7.2 Experimental Results

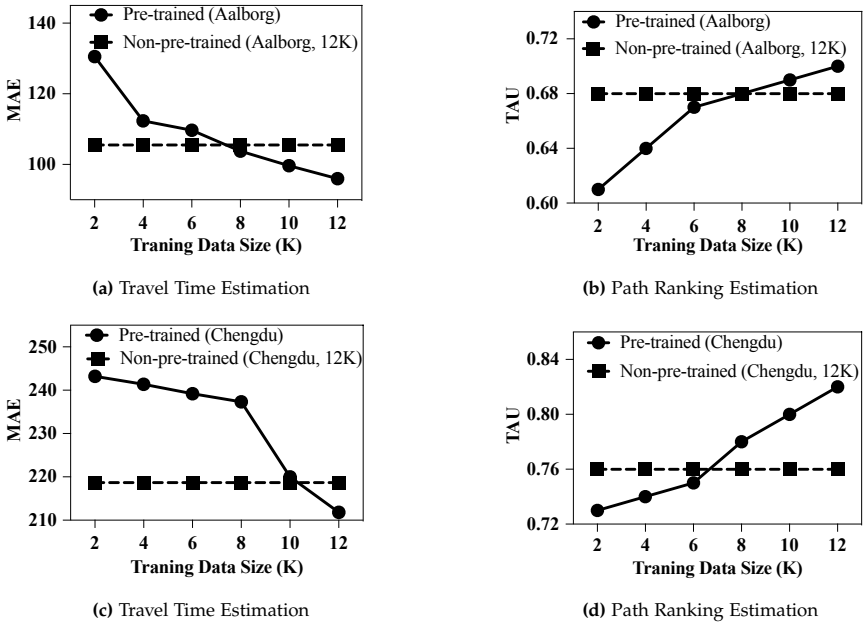
Overall Performance

Table E.3 shows the overall performance of our *LightPath* and all the compared baselines on both datasets in terms of different evaluation metrics. Especially, we select 30K unlabeled paths on Aalborg and Chengdu, respectively, but we only have 12K labeled paths for both datasets. Thus, we use 30K unlabeled paths to train path encoder for unsupervised-based methods. However, supervised approaches can only use the 12K labeled paths. Overall, *LightPath* outperforms all the baselines on these two tasks for both datasets, which demonstrates the advance of our model. Specifically, we can make the following observations. Graph node representation learning approach based *Node2vec* is much worse than *LightPath*. This is because *Node2vec* fails to capture spatial dependencies in a path. In contrast, *LightPath* considers the spatial dependencies through the self-attention mechanism, thus achieving better performance.

Although *MoCo* considers the dependencies among edges in a path, this method still performs worse. The main reason is that *MoCo* can leverage the spatial dependencies, but it converges very slow since it needs large amounts of negative samples to enable training. *LightPath* also outperform *t2vec* and *NeuTraj*, which both are first design to learn trajectory representation for trajectory similarity computation. This suggests that random drops on some edges and not reconstruct these edges in a path resulting in spatial information missing, thus achieving the worse performance on downstream

Table E.3: Overall Accuracy on Travel Time Estimation and Ranking Score Estimation [36].

Method	Aalborg						Chengdu					
	Travel Time Estimation			Path Ranking			Travel Time Estimation			Path Ranking		
	MAE	MARE	MAPE	MAE	τ	ρ	MAE	MARE	MAPE	MAE	τ	ρ
Node2vec	154.07	0.20	25.22	0.24	0.59	0.64	267.28	0.23	26.30	0.15	0.74	0.77
MoCo	146.29	0.19	21.60	0.25	0.53	0.57	237.14	0.20	23.13	0.15	0.77	0.81
Toast	137.27	0.17	20.43	0.24	0.59	0.63	240.57	0.21	23.50	0.11	0.65	0.68
t2vec	147.24	0.19	22.13	0.25	0.52	0.56	242.96	0.21	23.65	0.14	0.77	0.82
NeuTraj	117.06	0.15	18.09	0.25	0.60	0.64	232.96	0.20	22.73	0.12	0.79	0.83
PIM	102.09	0.14	14.92	0.20	0.63	0.67	223.34	0.19	21.69	0.12	0.80	0.84
HMTRL	101.81	0.13	14.51	0.17	0.68	0.72	218.94	0.19	21.22	0.09	0.83	0.84
PathRank	115.37	0.15	16.41	0.21	0.64	0.68	229.85	0.20	22.53	0.11	0.81	0.82
LightPath-Sup	105.51	0.15	16.35	0.14	0.68	0.72	218.67	0.19	21.36	0.13	0.76	0.79
LightPath	**85.76**	**0.11**	**12.12**	**0.13**	**0.73**	**0.77**	**212.61**	**0.18**	**20.75**	**0.07**	**0.87**	**0.88**



(a) Travel Time Estimation

(b) Path Ranking Estimation

(c) Travel Time Estimation

(d) Path Ranking Estimation

Figure E.7: Effects of Pre-training.

tasks. *PIM* consistently outperforms all other unsupervised baselines, which demonstrates the effectiveness of representation learning. The main reason is that *PIM* is designed for path representation learning. However, *PIM* is InfoNCE based method and has high computation complexity, making it hard to deploy on resource-limited edge devices.

HMTRL, *PathRank* and *LightPath-Sup* are three supervised learning methods that achieve relatively worse performance due to the lack of labeled training data. Since labeling data is very time-consuming and expensive. We consider a scenario where labeled data is limited in this paper.

Using *LightPath* as Pre-training Methods

In this experiment, we evaluate the effect of Pre-training. We employ *LightPath* as a pre-training method for the supervised method *LightPath-Sup*. Specifically, we first train *LightPath* in an unsupervised fashion, and then we use the learned transformer path encoder to initialize the transformer in *LightPath-Sup*. Here, it takes as input a sequence of edge representations and predicts the travel time and path ranking score. Figure E.7 illustrates the performance of *LightPath-Sup* w. and w/o pre-training over two downstream tasks on both datasets. When employing non-pre-trained *LightPath-Sup*, we train it using 12K labeled training paths. We notice that (1) when employing pre-training, we can obtain

Table E.4: Effect of Variants of *LightPath*

Method	Aalborg					
	Travel Time Estimation			Path Ranking		
	MAE	MARE	MAPE	MAE	τ	ρ
w/o RR	94.90	0.12	13.85	0.17	0.66	0.70
w/o Rec.	103.45	0.14	15.76	0.15	0.65	0.69
w/o ME	91.57	0.12	13.09	0.16	0.68	0.72
w/o CN	93.17	0.12	13.35	0.15	0.68	0.73
w/o CV	89.84	0.12	13.51	0.15	0.68	0.72
LightPath	**85.76**	**0.11**	**12.12**	**0.13**	**0.73**	**0.77**
Method	**Chengdu**					
	Travel Time Estimation			**Path Ranking**		
	MAE	**MARE**	**MAPE**	**MAE**	τ	ρ
w/o RR	224.31	0.19	21.90	0.14	0.76	0.79
w/o Rec.	229.24	0.20	22.36	0.16	0.69	0.73
w/o ME	223.81	0.19	21.86	0.13	0.78	0.81
w/o CN	217.14	0.18	21.29	0.08	0.80	0.83
w/o CV	215.59	0.18	21.20	0.09	0.81	0.83
LightPath	**212.61**	**0.18**	**20.75**	**0.07**	**0.87**	**0.88**

the same performance with no-pre-trained *LightPath-Sup* using less labeled data. For example, *LightPath-Sup* w. pre-training only needs 8K, and 10K labeled training paths for the Aalborg and Chengdu, respectively, to achieve the same performance of *LightPath-Sup* w/o pre-training with 12k labeled samples on the task of travel time estimation. (2) *LightPath-Sup* w. pre-training achieves higher performance than *LightPath-Sup* w/o pre-training. We observe similar results on the task of path ranking, demonstrating that *LightPath* can be used as a pre-training method to enhance supervised methods.

Ablation Studies

To verify the effectiveness of different components in *LightPath*, we conduct ablation studies on *LightPath*: a) effect of variants of *LightPath*, specifically reconstruction (Rec) loss, relational reasoning (RR) loss, cross-network loss and cross-view loss; b) effect of global-local knowledge distillation.

a) *Effect of variants of* LightPath, we consider five variants of *LightPath*: 1) *w/o RR*; 2) *w/o Rec.*; 3) *w/o ME*; 4) *w/o CN*; 5) *w/o CV*. In *w/o RR*, we only consider path reconstruction loss and use main encoder; In *w/o Rec.*, we only consider relational reasoning loss; In *w/o ME*, we consider both path reconstruction and relational reasoning losses, but we do not consider Siamese architectures in dual path encoder; In *w/o CN*, we remove the cross-network

Table E.5: Effect of KD, Global Loss and Local Loss

Method	Aalborg					
	Travel Time Estimation			Path Ranking		
	MAE	MARE	MAPE	MAE	τ	ρ
w/o KD	87.77	0.11	12.94	0.14	0.70	0.74
w/o Global	90.24	0.12	13.31	0.18	0.67	0.71
w/o Local	89.23	0.12	12.78	0.16	0.69	0.73
LightPath	**85.76**	**0.11**	**12.12**	**0.13**	**0.73**	**0.77**
Method	**Chengdu**					
	Travel Time Estimation			**Path Ranking**		
	MAE	**MARE**	**MAPE**	**MAE**	τ	ρ
w/o KD	213.26	0.18	20.97	0.08	0.84	0.86
w/o Global	220.32	0.19	21.52	0.09	0.79	0.81
w/o Local	215.03	0.19	21.02	0.08	0.82	0.84
LightPath	**212.61**	**0.18**	**20.75**	**0.07**	**0.87**	**0.88**

loss in RR; And in *w/o CV*, we remove cross-view loss in RR. Table E.4 report the results on both dataset. We can observe that (1) *LightPath w/o Rec.* achieves the worst performance because the learned *PR* only capture information from sparse path while ignoring the removed edges, which verifies the importance of path reconstruction decoder; (2) *LightPath w/o RR* also achieves the poor performance, which implies the effectiveness of self-supervised relational reasoning. (3) We observe that the performance of *LightPath* degrades without cross-network and cross-view loss on both datasets, which further demonstrates the effectiveness of our relational reasoning loss. (4) We notice that *LightPath* achieves the best performance. This result implies that all the proposed modules contribute positively to the final performance, which validates that *LightPath* takes advantage of all designed components.

b) *Effect of KD, global KD loss, local KD loss*: We further study the effect of global-local knowledge distillation. We compared our framework with three variants: 1) *w/o KD*, which denotes the performance of the teacher model; 2) *w/o global KD loss*, which removes global loss from global-local knowledge distillation; and 3) *w/o local KD loss*, which removes local loss from global-local knowledge distillation. As shown in Table E.5, compared with *KD*, *LightPath* achieves a better performance, which verifies that the teacher model can improve the performance of the student model. Both global and local loss can improve the performance of the learned path representation of the student model. In specific, global loss makes more contributions to the learned path representations. As a result, removing global loss degrades performance significantly.

Table E.7: Effect of Reduction Ratio γ

γ	Aalborg					
	Travel Time Estimation			Path Ranking		
	MAE	MARE	MAPE	MAE	τ	ρ
0.1	82.79	0.11	11.95	0.12	0.74	0.77
0.3	84.75	0.11	12.14	0.13	0.73	0.77
0.5	84.81	0.11	11.86	0.14	0.72	0.76
0.7	85.91	0.11	12.49	0.14	0.71	0.75
0.9	85.76	0.11	12.12	0.13	0.73	0.77

Parameter Sensitivity Analysis

We proceed to study three important hyper-parameters, including 1) model scalability w.r.t. reduction ratio and path length, 2) model scalability comparison, 3) Effect of Reduction Ratio γ, 4) the parameter of temperature for global-local knowledge distillation, and 5) effect of balancing factor α.

Model Scalability In the sequel, we explore the model scalability in terms of reduction ratio and path length based on the synthetic dataset. Table **??** depicts the results for both *LightPath* and its teacher model, with varying $\gamma = 0, 0.1, 0.3, 0.5, 0.7, 0.9$. $\gamma = 0$ denotes we do not conduct sparsity operation for the input path, i.e., using a classic Transformer based encoder. We can observe that the GFLOPs and gMem. (GiB) decrease with the increase in the reduction ratio. It is because the higher value of γ is, the more edges we can remove. Second, *LightPath* has significantly reduced model complexity, w.r.t., GFLOPs and gMem.. For example, we can reduce the training GFLOPs by 2.54$\times$ for the *LightPath* by increasing the reduction ratio γ from 0 to 0.9 in terms of path length 200. Moreover, *LightPath* also shows better performance (i.e., GFLOPs and gMem.) over teacher model, e.g., 1.79$\times$ GFLOPs speedup with reduction ratio $\gamma = 0.9$. Third, the parameters (Para. (Millions)) of teacher model is at least 3.5$\times$ of *LightPath*, which implies the effectiveness of our proposed framework. Overall, *LightPath* shows potential of scalability to support path representation learning for long paths.

Model Scalability Comparison In this section, we further explore the scalability performance (i.e., GFLOPs, gMem.(GiB), and Para. (Millions)) of *LightPath* compared with *PIM* and *Toast* with respect to different path lengths. We select *PIM* since (1) it achieves the better performance compared with other unsupervised baselines (cf. Table E.3); (2) it is a InforNCE based contrastive path representation learning method. In contrast, we choose *Toast* since it is Transformer-based method. Table E.9 report the experiment results. We

Table E.8: Effect of Temperature t in KD

t	**Aalborg**					
	Travel Time Estimation			**Path Ranking**		
	MAE	**MARE**	**MAPE**	**MAE**	τ	ρ
1	92.01	0.12	13.30	0.16	0.65	0.69
3	94.11	0.12	13.54	0.15	0.68	0.72
5	90.39	0.12	12.85	0.15	0.66	0.70
7	89.64	0.12	12.76	0.15	0.70	0.74
9	**85.76**	**0.11**	**12.12**	**0.13**	**0.73**	**0.77**
11	87.15	0.12	12.43	0.14	0.70	0.74
t	**Chengdu**					
	Travel Time Estimation			**Path Ranking**		
	MAE	**MARE**	**MAPE**	**MAE**	τ	ρ
1	225.70	0.20	22.02	0.09	0.79	0.82
3	217.07	0.19	20.77	0.09	0.81	0.84
5	216.93	0.19	21.24	0.08	0.83	0.86
7	214.88	0.19	20.98	0.08	0.86	0.87
9	**212.61**	**0.18**	**20.75**	**0.07**	**0.87**	**0.88**
11	214.17	0.19	21.00	0.08	0.83	0.85

observe that, (1) although the *LightPath*'s model parameters is at least 2× compared with *PIM*, the GFLOPs and gMem. of *LightPath* is much less than *PIM*. In particular, *LightPath* has much less GFLOPs and gMem. compared with *PIM* when path length is 200. i.e., 42.47 v.s. 12.31 and 2.80 v.s. 1.65; (2) *LightPath* has significant computation performance improvement over *Toast*, where the GFLOPs becomes much larger than *LigthPath* when path length increases, i.e., at least 9×. In addition, *Toast* also needs larger memory compared with *LightPath* when path length gets long, i.e. 3.38 v.s. 1.65 when path length is 200. Overall, our *LightPath* shows good potential scalability to support deploying it on resource-limited environments.

Effect of Reduction Ratio γ To study the impact of reduction ratio γ in the final performance, we conduct an experiment by varying the γ from 0.1 to 0.9 on Aalborg, which is shown in Table E.7. We can observe that the overall performance in both downstream tasks degrades a little when γ increases, which is reasonable as the the model has more input information. However, we can also observe the performance differences are not so significant, which suggests the effectiveness of our proposed framework. Even when a high reduction ratio is applied, the performance does not does not go down too much. Therefore, our proposed method can achieve good scalability while ensuring accuracy.

Table E.9: Model Scallability Comparison

N	**LightPath**	**PIM**	**Toast**
	GFLOPs/gMem./Para.	GFLOPs/gMem./Para.	GFLOPs/gMem./Para.
50	3.18/1.33/1.57	10.62/2.02/0.66	28.43/2.23/1.81
100	6.23/1.43/1.57	21.23/2.28/0.66	56.87/2.63/1.81
150	9.27/1.52/1.57	31.85/2.55/0.66	93.30/2.95/1.81
200	12.31/1.65/1.57	42.47/2.80/0.66	113.74/3.38/1.81

Table E.10: Effect of Balancing Factor α

α	**Aalborg**					
	Travel Time Estimation			**Path Ranking**		
	MAE	**MARE**	**MAPE**	**MAE**	τ	ρ
0	90.24	0.12	12.78	0.16	0.69	0.73
0.2	89.35	0.12	12.85	0.14	0.69	0.73
0.4	91.57	0.12	13.17	0.15	0.69	0.73
0.6	**85.76**	**0.11**	**12.12**	**0.13**	**0.73**	**0.77**
0.8	87.44	0.12	12.76	0.14	0.70	0.75
1	89.23	0.12	12.78	0.16	0.69	0.73
Method	**Chengdu**					
	Travel Time Estimation			**Path Ranking**		
	MAE	**MARE**	**MAPE**	**MAE**	τ	ρ
0	220.32	0.19	21.52	0.09	0.79	0.81
0.2	217.10	0.19	21.34	0.09	0.78	0.80
0.4	217.33	0.19	21.21	0.08	0.85	0.87
0.6	**212.61**	**0.18**	**20.75**	**0.07**	**0.87**	**0.88**
0.8	214.34	0.19	20.93	0.08	0.84	0.86
1	215.03	0.19	21.02	0.08	0.82	0.84

Effect of Temperature t of Knowledge Distillation To study the effect of the temperature t, we conduct a parameter study on both datasets, which is reported in Table E.8. We can observe that the performance of *LightPath* varies with different temperatures. It can be figured out that the best temperature t is 9, which indicates warm temperature can mitigate the peakiness of the teacher model and results in better performance.

Effect of Balancing Factor α . To study the effect of the balancing factor of global-local knowledge distillation, we conduct a parameter study on both datasets. Based on the results reported in Table E.10, we observe that the performance of our model changes when varying α. We can observe that the optimal α is 0.6, which means that global and local knowledge distillation loss can contribute to the *LightPath*'s performance. When $\alpha = 0$, the global

knowledge distillation loss is ignored, which yields poor performance. When $\alpha = 1.0$, the local knowledge distillation loss is ignored, and the performance also performs poorly. This confirms our conjecture that the two proposed global-local knowledge distillation losses can regularize each other and achieve better results than only optimizing one of them (i.e., $\alpha = 0.0$ or $\alpha = 1.0$).

Model Efficiency

We finally evaluate the model efficiency, including training and inference phases. Figure **??** illustrates the corresponding results. The first observation is that *LightPath* outperforms *PIM* and *Toast* in both training and inference phases. In the training phase, *LightPath* is more than $3\times$ faster than *PIM* and almost $5\times$ faster than *Toast* when path length is 200. In the testing phase, we measure the running time for each path sample. As observed, *LightPath* achieves up to at least 100% and almost 200% performance improvement compared with *PIM* and *Toast* when path length is 200.

8 Conclusion

We design a lightweight and scalable framework called *LightPath* for unsupervised path representation learning. In this framework, we first propose sparse auto-encoder that is able to reduce path length N to N', where N is much larger than N', which in turn reduces the computation complexity of the model. Then, we use path reconstruction decoder to reconstruct the input path to ensure no edges information missing. Next, we propose a novel self-supervised relational reasoning approach, which contains cross-network relational reasoning and cross-view relational reasoning loss, to enable efficient unsupervised training. After that, we introduce global-local knowledge distillation to further reduce the size of sparse path encoder and improve the performance. Finally, extensive experiments on two real-world datasets verify the efficiency, scalability, and effectiveness of *LightPath*.

References

[1] L. J. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *CoRR*, vol. abs/1607.06450, 2016.

[2] D. Babaev, N. Ovsov, I. Kireev, M. Ivanova, G. Gusev, I. Nazarov, and A. Tuzhilin, "Coles: Contrastive learning for event sequences with self-supervision," in *SIGMOD*, 2022, pp. 1190–1199.

[3] Y. Chen, X. Li, G. Cong, Z. Bao, C. Long, Y. Liu, A. K. Chandran, and R. Ellison, "Robust road network representation learning: When traffic patterns meet traveling semantics," in *CIKM*, 2021, pp. 211–220.

[4] H. Fan, F. Zhang, and Y. Gao, "Self-supervised time series representation learning by inter-intra relational reasoning," *CoRR*, vol. abs/2011.13548, 2020.

[5] Z. Fang, L. Pan, L. Chen, Y. Du, and Y. Gao, "MDTP: A multi-source deep traffic prediction framework over spatio-temporal trajectory data," *Proc. VLDB Endow.*, vol. 14, no. 8, pp. 1289–1297, 2021.

[6] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *KDD*, 2016, pp. 855–864.

[7] C. Guo, B. Yang, J. Hu, and C. S. Jensen, "Learning to route with sparse trajectory sets," in *ICDE*, 2018, pp. 1073–1084.

[8] C. Guo, B. Yang, J. Hu, C. S. Jensen, and L. Chen, "Context-aware, preference-based vehicle routing," *VLDB J.*, vol. 29, no. 5, pp. 1149–1170, 2020.

[9] X. Han, R. Cheng, C. Ma, and T. Grubenmann, "Deeptea: Effective and efficient online time-dependent trajectory outlier detection," *Proc. VLDB Endow.*, vol. 15, no. 7, pp. 1493–1505, 2022.

[10] K. He, H. Fan, Y. Wu, S. Xie, and R. B. Girshick, "Momentum contrast for unsupervised visual representation learning," in *CVPR*, 2020, pp. 9726–9735.

[11] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.

[12] R. D. Hjelm, A. Fedorov, S. Lavoie-Marchildon, K. Grewal, A. Trischler, and Y. Bengio, "Learning deep representations by mutual information estimation and maximization," *CoRR*, vol. abs/1808.06670, 2018.

[13] J. Hu, C. Guo, B. Yang, and C. S. Jensen, "Stochastic weight completion for road networks using graph convolutional networks," in *ICDE*, 2019, pp. 1274–1285.

[14] S. Huang, Y. Wang, T. Zhao, and G. Li, "A learning-based method for computing shortest path distances on road networks," in *ICDE*, 2021, pp. 360–371.

[15] X. Li, K. Zhao, G. Cong, C. S. Jensen, and W. Wei, "Deep representation learning for trajectory similarity computation," in *ICDE*, 2018, pp. 617–628.

[16] H. Liu, J. Han, Y. Fu, J. Zhou, X. Lu, and H. Xiong, "Multi-modal transportation recommendation with unified route representation learning," *Proc. VLDB Endow.*, vol. 14, no. 3, pp. 342–350, 2020.

[17] H. Liu, C. Jin, B. Yang, and A. Zhou, "Finding top-k shortest paths with diversity," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 3, pp. 488–502, 2018.

[18] P. Newson and J. Krumm, "Hidden markov map matching through noise and sparseness," in *SIGSPATIAL*, 2009, pp. 336–343.

[19] M. Patacchiola and A. J. Storkey, "Self-supervised relational reasoning for representation learning," in *NeurIPS*, 2020.

[20] S. A. Pedersen, B. Yang, and C. S. Jensen, "Anytime stochastic routing with hybrid learning," *PVLDB*, vol. 13, no. 9, pp. 1555–1567, 2020.

[21] S. Peter, F. Diego, F. A. Hamprecht, and B. Nadler, "Cost efficient gradient boosting," in *NIPS*, 2017, pp. 1551–1561.

[22] Z. Shang, G. Li, and Z. Bao, "DITA: A distributed in-memory trajectory analytics system," in *SIGMOD*, 2018, pp. 1681–1684.

[23] ——, "DITA: distributed in-memory trajectory analytics," in *SIGMOD*, 2018, pp. 725–740.

[24] L. V. Tran, M. Mun, M. Lim, J. Yamato, N. Huh, and C. Shahabi, "Deeptrans: A deep learning system for public bus travel time estimation using traffic forecasting," *Proc. VLDB Endow.*, vol. 13, no. 12, pp. 2957–2960, 2020.

[25] A. van den Oord, Y. Li, and O. Vinyals, "Representation learning with contrastive predictive coding," *CoRR*, vol. abs/1807.03748, 2018.

[26] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *NIPS*, 2017, pp. 5998–6008.

[27] P. Velickovic, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep graph infomax," in *ICLR*, 2019.

[28] S. Wang, Z. Bao, J. S. Culpepper, T. Sellis, and X. Qin, "Fast large-scale trajectory clustering," *Proc. VLDB Endow.*, vol. 13, no. 1, pp. 29–42, 2019.

[29] Y. Wang, Q. Wang, H. Koehler, and Y. Lin, "Query-by-sketch: Scaling shortest path graph queries on very large networks," in *SIGMOD*, 2021, pp. 1946–1958.

[30] Z. Wang, C. Long, G. Cong, and Y. Liu, "Efficient and effective similar subtrajectory search with deep reinforcement learning," *Proc. VLDB Endow.*, vol. 13, no. 11, pp. 2312–2325, 2020.

[31] V. J. Wei, R. C. Wong, and C. Long, "Architecture-intact oracle for fastest path and time queries on dynamic spatial networks," in *SIGMOD*, 2020, pp. 1841–1856.

[32] X. Wu, D. Zhang, C. Guo, C. He, B. Yang, and C. S. Jensen, "Autocts: Automated correlated time series forecasting," *Proc. VLDB Endow.*, vol. 15, no. 4, pp. 971–983, 2021.

[33] S. B. Yang, C. Guo, J. Hu, J. Tang, and B. Yang, "Unsupervised path representation learning with curriculum negative sampling," in *IJCAI*, 2021, pp. 3286–3292.

[34] S. B. Yang, C. Guo, J. Hu, B. Yang, J. Tang, and C. S. Jensen, "Weakly-supervised temporal path representation learning with contrastive curriculum learning - extended version," *CoRR*, vol. abs/2203.16110, 2022.

[35] S. B. Yang, C. Guo, and B. Yang, "Context-aware path ranking in road networks," *IEEE TKDE*, 2020.

[36] S. B. Yang, J. Hu, C. Guo, B. Yang, and C. S. Jensen, "Lightpath: Lightweight and scalable path representation learning," in *In Submission*, 2022.

[37] S. B. Yang and B. Yang, "Learning to rank paths in spatial networks," in *ICDE*, 2020, pp. 2006–2009.

[38] D. Yao, G. Cong, C. Zhang, and J. Bi, "Computing trajectory similarity in linear time: A generic seed-guided neural metric learning approach," in *ICDE*, 2019, pp. 1358–1369.

[39] H. Yuan, G. Li, Z. Bao, and L. Feng, "Effective travel time estimation: When historical trajectories over road networks matter," in *SIGMOD*, 2020, pp. 2135–2149.

[40] ——, "An effective joint prediction model for travel demands and traffic flows," in *ICDE*, 2021, pp. 348–359.

[41] J. Zhang, Y. Zheng, J. Sun, and D. Qi, "Flow prediction in spatio-temporal networks based on multitask deep learning," *IEEE Trans. Knowl. Data Eng.*, vol. 32, no. 3, pp. 468–478, 2020.

ISSN (online): 2446-1628
ISBN (online): 978-87-7573-796-3

AALBORG UNIVERSITY PRESS