PathOracle: A Deep Learning Based Trip Planner for Daily Commuters

Md. Tareq Mahmood [✉][1][0000−0002−1248−6939], Mohammed Eunus Ali[1][0000−0002−0384−7616], Muhammad Aamir Cheema[2][0000−0003−2139−9121], Syed Md. Mukit Rashid[1], and Timos Sellis[3][0000−0002−9067−5639]

[1] Bangladesh University of Engineering and Technology, Dhaka, Bangladesh
{tareqmahmood, eunus, mukitrashid}@cse.buet.ac.bd
[2] Monash University, Melbourne, Australia
aamir.cheema@monash.edu
[3] Archimedes/Athena RC, Greece
timossellisg@gmail.com

Abstract. In this paper, we propose a novel data-driven approach for a trip planner, that finds the most popular multi-modal trip using public transport from historical trips, given a source, a destination, and user-defined constraints such as time, minimum switches, or preferred modes of transport. To solve the most popular trip and its variants, we propose a multi-stage deep learning architecture, PathOracle, that consists of two major components: KSNet to generate key stops, and MPTNet to generate popular path trips from a source to a destination passing through the key stops. We also introduce a unique representation of stops using Stop2Vec that considers both the neighborhood and trip popularity between stops to facilitate accurate path planning. We present an extensive experimental study with a large real-world public transport based commuting Myki dataset of Melbourne city, and demonstrate the effectiveness of our proposed approaches.

Keywords: Public Transport · Path Recommendation · Trip Planning · Learning Popular Trips

1 Introduction

Almost every modern city offers a must-have trip planner (or a journey planner) [1] for the smooth and convenient daily commuting of its dwellers. A trip planner is a web or mobile search engine application to find an optimal means (e.g., fastest, shortest, or cheapest) of traveling between two locations in the city using public transport, where a single trip may use a sequence of several modes of transport. The service has become so ubiquitous that major map services such as Google Maps integrate such trip planners with their system. These search-based trip planners rely on the available transport networks and the timetables of the public transport services of a city and find one or more trip options from a source to a destination by optimizing different criteria [2], e.g., minimum travel

time or a minimum number of switches. These existing planners have the following limitations: (i) they do not support returning the preferred trip (i.e., popular one) taken by the past users, which might be of interest for many users, especially tourists; and (ii) these systems rely on the transport network and fixed timetables of the transport service and, thus, do not work when timetables are not available which is the case for a large number of developing mega cities like Dhaka, Karachi, Delhi etc. To mitigate the above problems, in this paper, we take an orthogonal and a completely new data-driven approach for a trip planner that finds the most popular trip from historical trips given a source s, a destination d, and user-defined constraints such as minimum switches or preferred modes of transport.

Popular paths between a source and a destination may vary at different times of the day (or days of the week). Moreover, for the case of daily commuting, the user may want to know the details of the popular path, if the path consists of a combination of different transport modes such as bus, train, and walk. Also, the path preferences of individuals may change, e.g., some may prefer bus over tram, others may prefer a single transport mode rather than taking a combination of bus, train, tram, etc. Answering popular paths tailored for individuals based on different contexts is a challenging research problem. In this paper, we propose a deep learning framework that learns from historical trips to generate popular paths between a given pair of source and destination and user preferences for a trip using public transport.

There have been few efforts to solve the popular path problems using trajectory data [6, 9, 12], which largely falls under route planning. A route planning problem typically deals with finding a route using a single private mode of transportation such as taxis or cars. In contrast, in our trip planner, we are interested in a path that uses one or more public transport modes to reach the destination. Chen *et al.* [6] find the popular route between two locations using HMM. Guo *et al.* [9] proposed a learning to route (L2R) approach that learns the routing behavior of trajectories in a region and transfers this learning to another region where enough user trajectories are not available for answering paths. In the most recent work, Li *et al.* [12] use a deep probabilistic learning based framework, called DeepST, to find the popular path from historical taxi trips.

Though the above works make important contributions for finding popular routes by learning from historical taxi trajectories, they have the following major limitations in addressing our problem of interest. (i) *Transport-Mode Oblivious:* They are oblivious to the transport mode of the route, i.e., they assume that users will use a single mode of transport in their entire path, which is not the case for most of the journeys on public transport. This limits the applicability of such systems in many cities, especially in developing cities where no journey planner is available for public transport network; also, although most of the modern cities provide a journey planner for commuting from one place to another, there are a number of ways to reach from one place to another, and there is no way for the users of these planners to know which path is generally used by most of the commuters. (ii) *Context Oblivious:* Existing works assume that popular

paths will remain fixed for the entire day or may only change in peak and off-peak hours. For example, a simple additional time context in the popular path queries requires both approaches [6, 9] to construct separate graphs for each time range, which is costly. We argue that the model should also learn useful contexts such as time, preferences, etc., while learning the user trajectories and also reflect this learning while answering path queries in a particular context. (iii) *Fixed Preference:* Existing works do not allow users to set their preferences while generating the path from source to destination. However, users may have some personal choices for the preferred trip, e.g., an older person may prefer a bus over a train as it is more accessible to her, or a disabled person may prefer a path with a minimum number of switches. Thus, incorporating user preferences in the popular path construction will facilitate more flexibility for the user.

The key challenges of solving the problem include how to incorporate the complex multi-modal nature of user trips and other preferences, such as minimum switches or preferred mode of transport, of the users in the learning process. A straightforward way to build such a system by adapting the methodologies of existing works (e.g., [6] or [9]) may need to build a separate model for each transport mode and every conditional constraint such as time range and then answer popular paths by combining these models. Building such a large number of models, and more importantly, combining them in answering path queries tailored for individuals is infeasible. While the existing DeepST [12] model can be adapted to provide popular paths considering multi-modal transport and different departure times (which we consider as a baseline in our experimental study), none of the existing approaches can handle user preferences such as preferred mode or a minimum number of switches.

In this paper, we propose a multi-stage neural network framework, called PathOracle, that essentially learns the travel patterns of users while commuting in a city using public transport. The key intuition of PathOracle is to learn key intermediate stops to reach from a source to a destination and use these intermediate stops to generate a mostly preferred trip from historical trips. To achieve this, we build two networks, Key Stops Net (KSNet) to generate key stops on the most probable trip for a given source to a destination and Most Probable Trip Net (MPTNet) to generate trips by progressively connecting key stops to reach from the source to the destination. One of the most important features of PathOracle is the flexibility of incorporating constraints such as time, preferred mode, or trip with the minimum number of switches during query time. PathOracle achieves this flexibility by decoupling key stop generation with the popular trip generation. In short, our contributions are as follows:

- We are the first to formulate the problem of answering the most popular path query from historical trips in the context of multi-modal public transports based city commuting, which allows users to find the popular path for a given source-destination, and preferences such as time, a preferred mode of transport, and minimum switches.
- We propose a deep learning architecture, PathOracle, that consists of two major components: KSNet to generate key stops and MPTNet to generate

popular path trips from a source to a destination passing through the key stops. We also introduce a unique representation of stops using Stop2Vec that considers both the neighborhood and trip popularity between stops.
– We present an extensive experimental study with a large real-world public transport based commuting dataset of Melbourne city and test the effectiveness of our proposed approaches.

2 Definitions and Problem Statement

Stop: A stop x is a location in a map where a person can get on or get off a vehicle. A stop is represented as a tuple $x = (id, lat, lon)$, where id is the stop-id, lat is the latitude and lon is the longitude of the stop. $\mathcal{S}$ is the set of all stops.

Mode: A mode m represents the type of a transport mode. $\mathcal{M}$ is the set of all modes. In our case, $\mathcal{M} = \{bus, train, tram, walk\}$ as we consider city based public transport in Melbourne city.

Hop: A hop $h = (u, v, m, t)$ is represented as a tuple, where a person starts from stop $u \in \mathcal{S}$ at time t and goes to stop $v \in \mathcal{S}$ using a transport mode $m \in \mathcal{M}$.

Trip: A trip $T = [h_i]_{i=1}^{n}$ is a sequence of n hops, where $h_1.u$ is the source, $h_n.v$ is the destination and $h_i.v = h_{i+1}.u$ for $i > 0$.

Stop Sequence: The stop sequence of a trip T includes starting stops of the hops and the destination. It is represented as $x(T) = \langle h_1.u, h_2.u, \ldots, h_n.u, h_n.v \rangle$.

Mode Sequence: The mode sequence of a trip T is the sequence of types of transport modes of the trip. It is represented as $m(T) = \langle h_1.m, h_2.m, \ldots, h_n.m \rangle$.

Trip Length: The length of a trip T is defined as the length of its node sequence, which is $l(T) = |x(T)|$

Mode Coverage: Mode coverage $mc(T, m)$ of a mode m in a trip T is the fraction of the distance of T travelled by m. For simplicity, in the calculation of mode coverage, we exclude walking distances as distances travelled by walk is significantly smaller compared to the distance travelled on vehicles.

Query: A query $q = (s, d, t)$ is a tuple of source $s \in \mathcal{S}$, destination $d \in \mathcal{S}$ and starting time t.

Most Popular Trip (MPT): Given a list of historical trips $\mathcal{H}$ and a query $q = (s, d, t)$, the MPT query predicts the most popular trip T^* that starts from stop s at time t and ends at stop d.

Most Popular Trip with Preferred Mode (MPTPM): Given a list of historical trips $\mathcal{H}$, a query $q = (s, d, t)$, a preferred mode m and mode coverage c, the MPTPM query predicts the most popular trip T^* that starts from stop s at time t, ends at stop d and the mode coverage $mc(T, m) \geq c$.

Most Popular Trip with Minimum Switch (MPTMS): Given a list of historical trips $\mathcal{H}$ and a query $q = (s, d, t)$, the MPTMS query predicts the most popular trip T^* that starts from stop s at time t, ends at stop d and has a trip length of $l(T^*) \leq l_q$ where l_q is the minimum length of all trips $T \in \mathcal{H}$ for query q, i.e., $l_q = \text{argmin}_{T \in \mathcal{H}} l(T)$.

Note that, in our deep learning based approach we do not require to define any explicit popularity metrics; rather, our approach learns from the historical

trips and returns the most likely path as the most popular path. This is also recommended as, in many cases, there may not be any direct trip from s to d in the historical trips, and the proposed algorithms learn to connect s with d using parts of other existing trips to return the preferred path. Thus, we predict the most probable trip with respect to the historical trips, and also design our evaluation metrics accordingly. Please see Section 4.2 for the details of the evaluation metrics and how predicted trips are compared with real observations.

3 Methodology

To answer the MPT query and its variants, we propose a multi-stage deep learning architecture, namely PathOracle. PathOracle consists of two major components: the key stop generation network (KSNet), and the popular trip generation network (MPTNet). Given a source and a destination, and the preferred time of the trip, the KSNet generates a number of key stops through which the popular trips from the source to the destination may pass through. The key intuition of KSNet comes from the observation that most of the trips pass through key stops such as central stations or transportation hubs, and thus identifying key stops play a vital role to the popular trip generation. Based on the identified key stops, we use another deep learning network, MPTNet, that constructs the popular paths by connecting the source and the destination via the key stops.

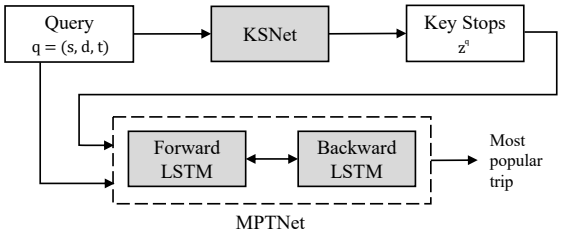


Fig. 1. The block diagram of PathOracle

Moreover, to generalize among stops in the same neighborhood and historical trip frequency among stops, we coin a concept called, Stop2Vec, for learning the vector representation of stops. Figure 1 shows an overview of PathOracle.

3.1 Stop Representation using Stop2Vec

Inspired by Node2Vec [8], we propose a new representation of stops, namely Stop2Vec, which learns low-dimensional features of stops based on historical trips. Learning representations directly from trajectories may be challenging due to the data sparsity issue of rarely-visited nodes. Also, a simple application of Node2Vec will not capture underlying popularity in the historical trips. Stop2Vec addresses both of these issues. The construction of Stop2Vec works as follows. First, we build a weighted graph G from historical trips. The weight of an edge (u, v) is the frequency of hops from u to v. Then, we sample R random walks per node from G. Finally, we adopt the Skip-Gram approach of word embedding as used in Node2Vec to learn node representation $e_s(x) \in \mathbb{R}^{n_s}$ for each stop $x \in \mathcal{S}$.

3.2 Time Representation

The travel patterns of users in a city largely vary at different times of the day (peak vs. off-peak) or on different days of the weeks (e.g., weekdays vs. weekends). Also, some transports may only be available for a particular period of a day. Thus, the popular trip using multi-modal transports between two stops may change with the time of the day. Thus, to capture the impact of time in the learning of popular trips, we split a day into σ_t time windows each having an interval of $\frac{\sigma_t}{24}$ hours. We represent each of the time windows as a fixed-sized vector $e_t(t) \in \mathbb{R}^{n_t}$ of size n_t. For this, both in KSNet and MPTNet, we add embedding layers that learn the representation of time as $e_t(t)$ while training.

3.3 Key Stop Generation (KSNet)

The key intuition for KSNet is that if we can identify the key stops such as transportation hubs of a trip that comes along reaching the destination, we can predict accurate paths. Thus, KSNet finds the most probable key stops between a source and a destination for a given time. Formally, given a query $q = (s, d, t)$, KSNet estimates the probability distribution $P_z(x|q)$, which represents the probability of any stop $x \in \mathcal{S}$ being an intermediate stop of a trip from s to d at time t. To achieve this, we assign KSNet a task to predict every intermediate stop of a trip from the source, destination, and starting time of the trip. The learning objective of KSNet is to maximize the sum of log-probability of all intermediate stops of all historical trips for the respective source, destination, and starting time. This is represented by the following objective function:

$$J_\theta = \frac{1}{|\mathcal{H}|} \sum_{T \in \mathcal{H}} \left(\frac{1}{l(T) - 2} \sum_{i=2}^{l(T)-1} \log p(x_i | T.q, \theta) \right)$$

Here, $p(x_i|T.q, \theta)$ is a probability estimation of $P_z(x_i|T.q)$ based on the learnable parameter θ of KSNet. The learned probability function will produce the likelihood of a stop being an intermediate stop of a query. For KSNet, first, we generate training samples from historical trips and then use a neural network to learn the probability function.

Sample Generation. Suppose, a trip T starts at time t and has a stop sequence $\langle x_1, x_2, \ldots x_n \rangle$. Here, the source is x_1 and the destination is x_n. We generate $n-2$ training samples from T for KSNet, where the inputs are the source, destination, and starting time, and outputs are the intermediate stops. For example, the generated samples from T will be $[(x_1, x_n, t), x_2], [(x_1, x_n, t), x_3], \ldots, [(x_1, x_n, t), x_{n-1}]$. Similarly, we generate samples for all the trips in $\mathcal{H}$.

KSNet Architecture. For a generated sample $[(s, d, t), x]$, we first obtain the embeddings of the source, destination and time. The embedding of the source and destination $(e_s(s), e_s(d))$ is obtained from the learned representation of

Stop2Vec. The embedding layer e_t for time is randomly initialized and is tuned gradually while training KSNet. The core part of KSNet is a Multi-Layer Perceptron (MLP). The MLP takes the concatenation of $e_s(s), e_s(d)$, and $e_t(t)$ as input. The input is passed through hidden layers of this feed-forward network. The output layer is comprised of $|\mathcal{S}|$ neurons with LogSoftmax activation function. LogSoftmax produces the log probabilities of all stops being an intermediate stop for query q. To maximize the objective function J_θ, we use Negative Log-Likelihood Loss (NLL Loss).

Prediction. After the training phase, KSNet is able to infer the probability distribution $P_z(x|q)$ for a query q. We select a list of top K key stops (Z^q) for the trip generation task, where K is a hyperparameter.

3.4 Most Popular Trip Generation (MPTNet)

At the last step of the PathOracle, we develop a separate neural network model, MPTNet. This part of our solution is motivated by [17] that finds alternate paths. MPTNet generates the most popular trip from a source to a destination that passes through the selected key stop. MPTNet consists of two RNN units, a forward-LSTM, and a backward-LSTM, to capture the forward and backward influence, respectively, from historical trips. The output of each LSTM unit is passed through two separate MLPs to predict the stop and the mode.

To generate the most popular trip for a query $q = (s, d, t)$, we obtain the key stops from KSNet in order of their likelihood, and we use MPTNet to perform the following procedure for finding a popular trip from source to destination via the selected key stop. Let Z_i^q be a key stop. We generate two candidate trips using MPTNet for the key stop Z_i^q. First, we generate a sub-trip from Z_i^q to s using the backward-LSTM and then consider the generated sub-trip as the given past sequence to generate another sub-trip from Z_i^q to d using the forward-LSTM. Second, we generate a sub-trip from Z_i^q to d using the forward-LSTM and then treat the generated sub-trip as the future sequence (of the to be generated trip) to generate another sub-trip from Z_i^q to s using the backward-LSTM. While selecting the next stop for a sub-trip in forward-LSTM, we only consider those stops that have a hop from the current stop according to historical trips. A similar strategy is implemented for backward-LSTM too. Finally, of the two candidate trips generated above, we pick the most probable trip as the most popular trip. We use the forward LSTM to compute the probability of a trip.

3.5 Preferred Mode Constraint

PathOracle is flexible enough to incorporate the mode preference of the user. In such a case, we find the most popular trip that mostly uses the preferred mode of choice. Specifically, we allow a user to give two constraint parameters, a transport mode m and target mode coverage c. Based on this, we extend the

KSNet (of Section 3.3) in such a way that the choice of preferred mode can influence the choice of key stops. For this, KSNet learns another probability function $P_m(x|q, m)$, which indicates the probability of x being a key stop of the popular trip from s to d at time t that mostly uses the transport mode m. The preferred mode m is passed to the MLP of KSNet through a learnable embedding layer e_m. The MLP of KSNet takes the concatenation of $(e_s(s), e_s(d), e_t(t), e_m(m))$, and gives the key stops with associated probabilities.

The sample generation for the training phase is also similar to Section 3.3. Suppose a trip T starts at time t, has stop sequence $\langle x_1, x_2, \ldots, x_n \rangle$ and mode sequence $\langle m_1, m_2, \ldots, m_{n-1} \rangle$. We find the mode m_p that has the most mode coverage in that trip,

$$m_p = \underset{m \in \mathcal{M}}{\operatorname{argmax}}\ mc(T, m)$$

where $mc(T, m)$ is the coverage of m in T. Then, we generate $n - 2$ samples: $[(x_1, x_n, t, m_p), x_2]$, $[(x_1, x_n, t, m_p), x_3]$, ..., $[(x_1, x_n, t, m_p), x_{n-1}]$. From the generated samples, KSNet learns to capture the pattern of key stops depending on preferred modes.

During the trip generation phase, we obtain K key stops from KSNet. In order of their likelihood, each of the key stops is passed to MPTNet to complete the trip by connecting the source and destination to the key stop. Once we find a trip that satisfies the mode coverage constraint, we consider that trip as the most popular trip with the preferred mode.

3.6 Minimum Switch Constraint

MPTNet allows us to fix the maximum length (L) of a generated trip. Suppose, MPTNet is generating a candidate trip and it has generated the first sub-trip of length L_1 from a key stop to the source. Then, MPTNet tries to connect the key stop to the destination within a sub-trip of length $L - L_1$. Thus, MPTNet can find the most probable trip within length L. This capability facilitates MPTNet to generate trips with the minimum switch constraint. Specifically, we set L to the minimum trip length value (i.e., 2) and enforce MPTNet to generate a trip. If MPTNet fails, L is incremented gradually up to the maximum value until MPTNet generates a trip.

4 Experiments and Results

In this section, we present the experimental evaluation for our solution, PathOracle, to answer the MPT query and its variants. As there is no prior work that directly answers these problems, we compare our solution with a number of baselines that we adapted by appropriately modifying state-of-art deep learning techniques suitable for these tasks.

4.1 Experimental Setup

Dataset. We use **Myki**[4] dataset which contains real-world public transport data of Victoria, Australia. The dataset consists of *touch-on* (getting on a vehicle) and *touch-off* (getting off a vehicle) events of the first 10 weeks of 2017. The dataset has 27620 stops, 2357 routes and on average 10 million events per week. Among the 10 weeks of 2017 datasets - we consider the first 8 weeks as the training dataset, the 9th week as the validation dataset, and the 10th week as the testing dataset. Each event has the following information: *Mode, Date and time, User ID, Vehicle ID, Route ID, and Stop ID*. We extract the trips taken by various users by connecting the discrete user events. Finally, we are able to reconstruct about 18 million trips. The number of trips we use for training, validation, and testing are 14 million (first eight weeks), 2 million (9th week), and 2 million (10th week), respectively.

We define trip length as the number of stops in a trip, including source and destination. A trip length includes walking events in between changes of vehicles. For example, if a person takes a train from Clayton to Melbourne Central, then walks to a nearby tram stop and reaches the University of Melbourne using a tram, the trip length is four, whereas the number of vehicles involved is 2 (train and then tram) and the number of vehicle switches is 1 (train to tram). We exclude the trips with a trip length of more than six or a vehicle count of more than three because such trips are extremely rare in the dataset.

Baselines. As no prior works focus on finding the popular trip using multi-modal public transports, we developed three baselines by adopting popular deep learning frameworks and state-of-the-art techniques.

- **LSTM:** We adapt vanilla LSTM [10] for popular trip generation using only forward influence. The output of LSTM is passed to two MLPs: one predicts the next stop, and the other predicts the next mode of transport.
- **FB-LSTM:** As a second baseline, we use both forward and backward influence with FB-LSTM (Forward-Backward LSTM) to answer our popular path queries. We separately train the two LSTM models, namely forward-LSTM for modeling forward influence and backward-LSTM for modeling backward influence. In this approach, we predict two trips from these two models and take the most probable one.
- **DeepST:** As the final baseline, we extend DeepST [12], which is the state-of-art model for finding the most probable trip for a given source, destination, traffic condition, and historical trajectories of taxi trips. We modify DeepST to predict sequences of stops and sequences of modes.

For consistency, each baseline is implemented to predict only those stops that have a hop from the current stop according to historical trips. On top of that, as there is no way to incorporate constraints during learning for the baselines, for

[4] `https://www.ptv.vic.gov.au/tickets/myki/`

preferred mode and minimum switch constraints, we implement Beam Search for each of the baselines. We generate 20 alternate trips using Beam Search and select the most optimal one according to the constraints. We observed that these two strategies improve the performance of the baselines.

Implementation Details. In Stop2Vec, we generate $R = 80$ random walks per stop. Random walks are generated up to a length of 10. The window size of the Skip-Gram model is set to 5. For PathOracle, the sizes of stop, time, and mode embeddings are set to 256, 36, and 36, respectively. We split 24 hours of a day into $\sigma_t = 4$ windows, each with a 6-hour interval. The MLP in KSNet has four layers where two hidden layers of size 128 and 32, respectively, are used. Forward and backward LSTMs of MPTNet are two single layer LSTM units with hidden size 512. The maximum allowable trip length L is set to 6 (as trips of more than six stops are extremely rare and thereby discarded from our dataset). We select top $K = 20$ key stops from KSNet. Models including baselines are implemented in PyTorch and are trained with one NVIDIA GeForce GTX 1080 GPU. Models are trained up to 30 epochs with Adam optimizer [11] and batch size 128.

4.2 Evaluation of MPT Query

In this section, we evaluate PathOracle against the baselines on the performance of generating the most popular trip for MPT queries.

Evaluation Metrics for MPT. The performance of an MPT query is evaluated based on five metrics: *stop accuracy, stop recall, mode accuracy, mode recall, and reachability.* The measurement of accuracy and recall is based on the metrics used in DeepST [12]. For a given ground-truth sequence y and a prediction sequence y^*, we can define the metrics as follows.

- *Accuracy* is the ratio of the count of correctly predicted stops/modes to the maximum length between y and y^*. Accuracy $= \frac{|y \cap y^*|}{\max(|y|,|y^*|)}$
- *Recall* is the ratio of the count of correctly predicted stops/modes from the first $|y|$ stops of y^* to the length of y. Recall is measured for both stop sequence and mode sequence. Recall $= \frac{|y \cap y^*_{1:|y|}|}{|y|}$
- *Reachability* is a boolean metric. If a predicted trip is able to reach the destination, the reachability of the trip is one. Otherwise, it is zero.

Performance Comparison. Table 1 and 2 show the performances of different approaches on predicting stop sequence and Table 3 and 4 show the performances on predicting mode sequence. Lastly, we show the performance on finding a trip to the destination. Here, we vary the trip length as 2, 3, 4, 5, and 6.

For shorter length of trips (i.e., 2-4), all models perform similarly well because it is relatively straightforward to capture short-range dependencies. As trip length becomes longer (i.e. 5-6), the performances of all methods drop. This

Table 1. The stop sequence accuracy versus trip length

	Trip Length				
Models	2	3	4	5	6
PathOracle	**0.98**	**0.97**	**0.83**	**0.80**	**0.73**
DeepST	0.97	0.96	0.82	0.76	0.63
FB-LSTM	0.96	0.94	0.75	0.69	0.51
LSTM	0.95	0.91	0.79	0.69	0.48

Table 2. The stop sequence recall versus trip length

	Trip Length				
Models	2	3	4	5	6
PathOracle	**0.98**	**0.97**	**0.83**	**0.80**	**0.73**
DeepST	0.97	0.96	0.82	0.76	0.63
FB-LSTM	0.96	0.94	0.75	0.69	0.51
LSTM	0.95	0.91	0.79	0.69	0.48

Table 3. The mode sequence accuracy versus trip length

	Trip Length				
Models	2	3	4	5	6
PathOracle	**0.97**	0.96	**0.90**	**0.88**	**0.84**
DeepST	0.96	0.97	0.62	0.67	0.51
FB-LSTM	0.95	0.94	0.84	0.82	0.70
LSTM	0.95	0.90	0.87	0.79	0.71

Table 4. The mode sequence recall versus trip length

	Trip Length				
Models	2	3	4	5	6
PathOracle	**1.00**	**0.97**	**0.92**	**0.88**	**0.84**
DeepST	1.00	0.97	0.65	0.69	0.54
FB-LSTM	0.99	0.95	0.86	0.82	0.73
LSTM	1.00	0.91	0.91	0.81	0.75

is because (i) the number of possible trips from source to destination increases rapidly with the increase of trip length, and (ii) capturing the long-range dependencies among stops in long sequences is challenging. We observe that, for lengths 5 and 6, PathOracle outperforms every other model significantly in accuracy and recall metrics. This is because PathOracle significantly reduces the number of possible trips by fixing a key stop between source and destination. The inclusion of the key stop also reduces the length of the sequence to be generated, thus, tackling the challenge of modeling long-range dependencies.

Moreover, PathOracle and FB-LSTM show significantly better reachability than others in long trips. In shorter trips, models perform similarly. However, especially for lengths 6, PathOracle and FB-LSTM are able to generate a valid trip to destination 94% times, whereas DeepST is able to generate such a trip only 83% times. Both PathOracle and FB-LSTM employ forward and backward influences together and thus have a better chance of reaching to destination compared to others, which is also evident from the results.

4.3 Evaluation of MPTPM Query

In this section, we compare the performance of models in generating the most popular trip with a preferred mode constraint. The constraint consists of a preferred mode m and a target mode coverage c.

Metric for MPTPM. The performance of a predicted trip of an MPTPM query is measured by the metric, Coverage Score. Coverage score is the ratio of the coverage of the preferred mode c_p in the predicted trip and the target coverage c. Coverage Score = $\min(1, c_p/c)$. The maximum value of a Coverage

Score can be 1. Also, if the predicted trip does not reach the destination, the coverage score will be 0.

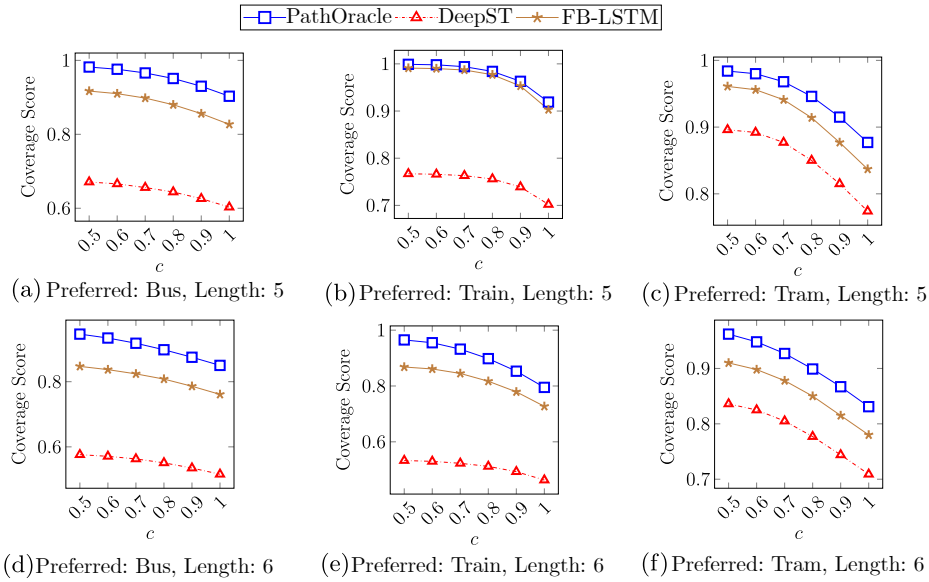
Performance Comparison. We evaluate performance for different preferred modes (bus, train, tram) with different mode coverages. Evaluating performance under this constraint is challenging because (i) there is no information about mode preferences in the dataset and (ii) a trip with random preferred transport mode and desired coverage may not be possible. So, for fair evaluation, we generate queries from the test dataset in such a way as to increase the chance of the existence of trips for those queries. The generation of queries is done in two approaches.

Firstly, we consider only the test dataset. For a transport mode m, we find the trips in the test dataset where the coverage of m is higher than other transport modes. From each of these query trips T, we query each method six times with the same source $T.s$, destination $T.d$, time $T.t$, and preferred mode m, while varying target coverage c from 0.5 to 1.0 with a 0.1 interval. We also keep track of the lengths of the trips we generated the queries from. We show the mean coverage score of MPTPM queries in Figure 2 for different preferred modes and lengths of the query trip. As the performance of the methods largely varies for long sequence trips, we only show the results for lengths 5 and 6. Here, PathOracle consistently shows better performance than DeepST and FB-LSTM in all cases. This is because KSNet can effectively select preferred mode-specific key stops that increase the mode coverage while MPTNet generates the popular trips through them. The coverage score of each method decreases with the increase of c.

Secondly, we generate only those queries from the test dataset such that each query has a trip in the historical dataset that satisfies the constraint. Let, T is the trip in the test dataset we are currently considering and m is our preferred mode. We find the maximum coverage (c_m) of m for all trips between $T.s$ and $T.d$ at time $T.t$ in the historical dataset. If $c_m < 0.5$, we discard the trip. Otherwise, we set target coverage to c, where value of c is 0.5, 0.7 or 0.9 when c_m is within $[0.5, 0.6)$, $[0.7, 0.8)$ or $[0.9, 1.0)$, respectively. Then, we generate an MPTMP query with source $T.s$, destination $T.d$, time $T.t$, preferred mode m, and target coverage c. We keep track of the length l of the historical trip where m has the maximum coverage c_m. We generate such queries for bus, train, and tram, for length $l = 5, 6$. Figure 3 shows the mean coverage score of models for different preferred modes and coverages. We observe that the PathOracle outperforms the baselines in all cases except one. We observe a lack of pattern with the change of c. This is because in each case of preferred mode, length, and target coverage, results are coming from a particular set of queries tailored for that case.

4.4 Evaluation of MPTMS Query

In this section, we compare the capabilities of the methods in finding the most popular trip with minimum change of vehicles. In other words, we want to gen-



(a) Preferred: Bus, Length: 5 (b) Preferred: Train, Length: 5 (c) Preferred: Tram, Length: 5

(d) Preferred: Bus, Length: 6 (e) Preferred: Train, Length: 6 (f) Preferred: Tram, Length: 6

Fig. 2. Performance under preferred mode constraint on test dataset queries

erate the most popular trip with a minimum trip length. Models are compared based on two experiments.

In the first set of experiments, we evaluate models based on historical ground truth of minimum length. For each query $q = (s, d, t)$ in the test dataset, we find the minimum length l_m of all historical trips that start from s at t and end in d. Then each of the methods predicts its most popular minimum trip for q. Say, the length of a predicted trip is l_p. We define a metric ML Score (Minimum Length Score), that measures the ratio of l_m and l_p. ML Score $= \min(1, l_m/l_p)$. If the length of the predicted trip is less than or equal to l_m, the score is 1.

In the first four columns of Table 5, we show the mean ML scores of different methods against different l_m. For the lack of variation in the score and for space constraints, we exclude the results for shorter paths. We observe that PathOracle consistently surpasses the competing methods, especially for length 6. This is because PathOracle can effectively impose restrictions on the trip generation process by reducing L. Whereas, FB-LSTM and DeepST are producing alternate trips using Beam Search without any consideration of the constraint.

For the second set of experiments, we compare the methods in the test dataset. We define a metric, Comparative Score, which is the fraction of cases where a model generated the shortest trip compared to other methods. Comparative scores of PathOracle, DeepST and FB-LSTM in this experiment are shown in the last column of Table 5. PathOracle outperforms DeepST and FB-LSTM by a significant margin.

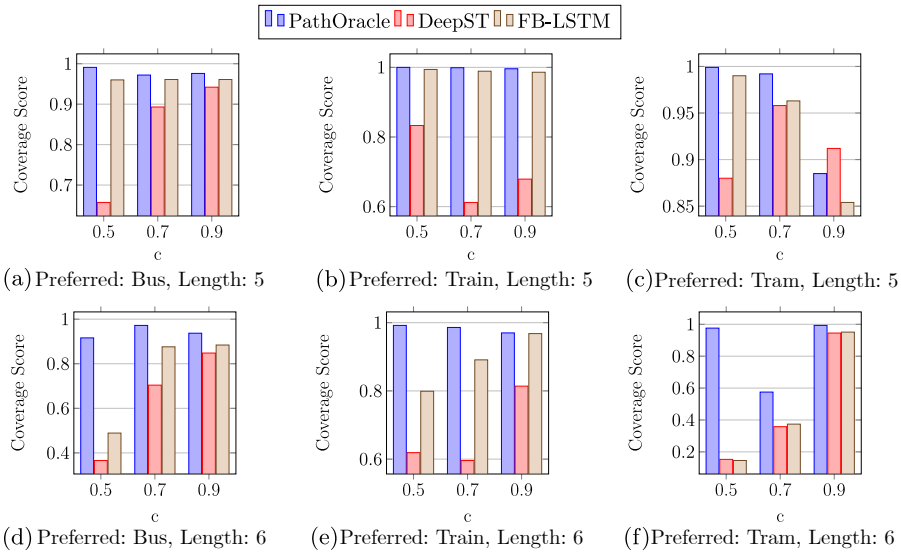


Fig. 3. Performance under preferred mode constraint on historical queries

Table 5. Performance comparison of different methods in MPTSM query

	Historical Minimum Length			Comparative
Models	4	5	6	Score
PathOracle	**0.999**	**0.999**	**0.975**	**0.958**
DeepST	0.918	0.886	0.715	0.802
FB-LSTM	0.995	0.991	0.859	0.906

5 Related Works

Trip planners like Google maps, PTV Journey Planner [1], OpenTripPlanner [2], etc., are widely used by millions every day commuting in a city using public transport. These planners use shortest path algorithms to find the fastest (or shortest/cheapest) route on the public transport network, given the schedule of the services run on the transport network. To facilitate these planners with real-time transit data, a number of works (e.g., [3, 4, 20]) proposed multi-modal trip planning algorithms. All these systems and algorithms work on the provided public transport network and public transport schedules of the services to recommend a route.

Though no work exists for learning based multi-modal popular trip planning, automated route planning (that considers trips with a single vehicle, such as taxis or cars) between two locations based on historical trajectories has been studied extensively in recent years. These studies tackle a variety of route planning related tasks such as the most popular trip planning [6, 12], personalized route recommendation [5,7,14], route pattern modeling [19], route cost estimation [15],

travel time estimation [13, 18], etc. Next, we will discuss the works on route planning and path recommendation from historical trajectories.

MPR [6] finds the most popular route between a source and a destination from historical trajectories. It works without road network data by first creating a transfer model to estimate the transfer probability of nodes. The most popular path is inferred by finding a path that maximizes the probability according to the transfer model. L2R [9] solves the problem of route planning in sparse trajectories. They learn the routing patterns between frequent regions and then transfer those patterns to regions with sparse trajectories. However, to answer time-based queries, both L2R and MPR have to create multiple models for different time ranges. MFP [16] is a search-based technique that considers the temporal context in finding the popular path. It searches the most frequent path from a source to a destination for a time frame in history. MFP processes a query by instantly building a sub-graph that captures the historical routing information for that time frame, then finds the most frequent path from the sub-graph. Being a search-based algorithm, query processing in MFP is expensive in computation and memory.

Recently, Li *et al.* [12] propose DeepST, a deep probabilistic model to learn spatial transition patterns from taxi trajectories. DeepST incorporates the impact of the past traveled route, destination, and real-time traffic condition for route generation, which shows the best performance in generating the most probable route. We have considered DeepST as one of our baselines by modifying it for the multi-modal popular path problem. To the best of our knowledge, none of the previous works can learn to generate popular multi-modal trips considering the impact of the time of the day and user preferences in a unified model.

6 Conclusion

In this paper, we have introduced the problem of answering the most popular path query by learning historical trips in the context of multi-modal public transports based city commuting. To solve this problem, we have developed a multi-stage deep learning architecture, PathOracle, that enables users to find the popular path for a given source-destination pair and the time of the travel. The PathOracle can gracefully accommodate user preferences constraints such as preferred mode of transports, and minimum number of switches in the trip. We have conducted an extensive experimental study with a large real-world public transport based commuting Myki dataset of Melbourne city. The results show that PathOracle outperforms all the baselines significantly, especially while answering longer trips involving multiple modes of public transport.

Acknowledgments. This work is done at DataLab, BUET. Muhammad Aamir Cheema is supported by ARC FT180100140.

References

1. Journey Planner - Public Transport Victoria, `https://www.ptv.vic.gov.au/journey`, Last accessed: 30/06/2022

2. Opentripplanner - Multimodal trip planning, http://opentripplanner.org/, Last accessed: 30/06/2022
3. Benchimol, P., Amrani, A., Khouadjia, M.: A multi-criteria multi-modal predictive trip planner: Application on paris metropolitan network. In: ISC2 (2021)
4. Borole, N., Rout, D., Goel, N., Vedagiri, P., Mathew, T.V.: Multimodal public transit trip planner with real-time transit data. In: Procedia-Social and Behavioral Sciences (2013)
5. Chang, K.P., Wei, L.Y., Yeh, M.Y., Peng, W.C.: Discovering personalized routes from trajectories. In: ACM SIGSPATIAL LBSN (2011)
6. Chen, Z., Shen, H.T., Zhou, X.: Discovering popular routes from trajectories. In: ICDE (2011)
7. Dai, J., Yang, B., Guo, C., Ding, Z.: Personalized route recommendation using big trajectory data. In: ICDE (2015)
8. Grover, A., Leskovec, J.: node2vec: Scalable feature learning for networks. In: ACM SIGKDD (2016)
9. Guo, C., Yang, B., Hu, J., Jensen, C.: Learning to route with sparse trajectory sets. In: ICDE (2018)
10. Hochreiter, S., Schmidhuber, J.: Long short-term memory. In: Neural Computation (1997)
11. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: ICLR (2015)
12. Li, X., Cong, G., Cheng, Y.: Spatial transition learning on road networks with deep probabilistic models. In: ICDE (2020)
13. Li, X., Cong, G., Sun, A., Cheng, Y.: Learning travel time distributions with deep generative model. In: WWW (2019)
14. Liu, H., Tong, Y., Zhang, P., Lu, X., Duan, J., Xiong, H.: Hydra: A personalized and context-aware multi-modal transportation recommendation system. In: ACM SIGKDD (2019)
15. Liu, H., Jin, C., Zhou, A.: Popular route planning with travel cost estimation from trajectories. In: Frontiers of Computer Science (2020)
16. Luo, W., Tan, H., Chen, L., Ni, L.M.: Finding time period-based most frequent path in big trajectory data. In: ACM SIGMOD (2013)
17. Rashid, S.M., Ali, M.E., Cheema, M.A.: DeepAltTrip: Top-k alternative itineraries for trip recommendation. arXiv (2021)
18. Wang, D., Zhang, J., Cao, W., Li, J., Zheng, Y.: When will you arrive? Estimating travel time based on deep neural networks. In: AAAI (2018)
19. Wu, H., Chen, Z., Sun, W., Zheng, B., Wang, W.: Modeling trajectories with recurrent neural networks. In: IJCAI (2017)
20. Yu, L., Shao, D., Wu, H.: Next generation of journey planner in a smart city. In: ICDMW (2015)