The VLDB Journal (2020) 29:1149–1170
https://doi.org/10.1007/s00778-020-00608-7

REGULAR PAPER

Context-aware, preference-based vehicle routing

Chenjuan Guo[1] · Bin Yang[1] · Jilin Hu[1] · Christian S. Jensen[1] · Lu Chen[1]

Received: 21 March 2019 / Revised: 23 November 2019 / Accepted: 24 February 2020 / Published online: 11 March 2020

Abstract

Vehicle routing is an important service that is used by both private individuals and commercial enterprises. Drivers may have different *contexts* that are characterized by different *routing preferences*. For example, during different times of day or weather conditions, drivers may make different routing decisions such as preferring or avoiding highways. The increasing availability of vehicle trajectory data yields an increasingly rich data foundation for *context-aware, preference-based* vehicle routing. We aim to improve routing quality by providing new, efficient routing techniques that identify and take contexts and their preferences into account. In particular, we first provide means of learning contexts and their preferences, and we apply these to enhance routing quality while ensuring efficiency. Our solution encompasses an *off-line* phase that exploits a contextual preference tensor to learn the relationships between contexts and routing preferences. Given a particular context for which trajectories exist, we learn a routing preference. Then, we transfer learned preferences from contexts with trajectories to similar contexts without trajectories. In the *on-line* phase, given a context, we identify the corresponding routing preference and use it for routing. To achieve efficiency, we propose preference-based contraction hierarchies that are capable of speeding up both off-line learning and on-line routing. Empirical studies with vehicle trajectory data offer insight into the properties of proposed solution, indicating that it is capable of improving quality and is efficient.

Keywords Trajectories · Contexts · Preference learning · Multi-task learning · Routing · Contraction hierarchies

1 Introduction

Vehicular transportation plays an important role not only in people's daily lives, but also in many businesses [1,2]. Due to a combination of recent, interrelated developments in autonomous vehicles, mobility-as-a-service, and big data, transportation will undergo profound changes in the years to come. It is a safe bet that vehicle routing decisions will increasingly be made algorithmically rather than by humans [3,4].

✉ Bin Yang
byang@cs.aau.dk

Chenjuan Guo
cguo@cs.aau.dk

Jilin Hu
hujilin@cs.aau.dk

Christian S. Jensen
csj@cs.aau.dk

Lu Chen
luchen@cs.aau.dk

[1] Department of Computer Science, Aalborg University, Aalborg, Denmark

Existing routing algorithms, ranging from Dijkstra's algorithm to contraction hierarchies, aim to identify shortest or fastest paths. However, recent studies [5,6] show that local drivers follow paths that are often neither fastest nor shortest and also often differ from what is offered by existing navigation services. When driving from a source to a destination, paths chosen by local drivers are based on their knowledge of local driving conditions and traffic. Thus, in different *contexts*, e.g., at different times of the day or in different weather conditions, drivers may have different *routing preferences* such as preferring or avoiding highways. Knowledge of contexts and associated preferences holds the potential to enhance the quality of routing for human drivers, and to enable autonomous vehicles to make routing decisions that mimic the behaviors of professional drivers. This motivates us to study *context-aware, preference-based routing (CPR)*.

As part of the continued digitalization of societal processes, more and more data that captures the movements of vehicles becomes available, notably in the form of GPS trajectories [7,8]. This data not only captures contextual information of trips, e.g., departure time and driver identifiers, but captures also the routing preferences made by

drivers, e.g., whether or not they are taking highways. Such data offers a foundation for understanding the relationship between contexts and routing preferences, which in turn offers a foundation for enabling context-aware, preference-based routing.

We propose to exploit routing behavior from local drivers by learning routing preferences in different contexts using trajectories from local drivers, and we then propose to utilize these preferences to enable context-aware, preference-based routing.

A *context* describes the setting at the time a trip starts. In particular, we categorize contexts into *spatial*, *temporal*, and *other* contexts. First, we consider spatial context that captures the source and destination of a trip at a given spatial granularity. Second, since traffic is time dependent [9,10], temporal contexts, such as departure time, the day of the week, and whether it is a holiday, are accommodated. Third, other contexts include, e.g., weather conditions and driver identifiers. Weather conditions, e.g., snow or rain, may affect traffic significantly, and different drivers may have different driving behaviors. For example, if we have $10 \cdot 10$ source–destination pairs, two time intervals (e.g., peak vs. off-peak intervals), and two weather conditions (e.g., snow vs. non-snow), we obtain $10 \cdot 10 \cdot 2 \cdot 2 = 400$ distinct contexts.

A *routing preference* encompasses a travel cost that a routing algorithm is able to minimize and a number of route properties. Thus, a routing preference has the format $\langle master, slaves \rangle$. The benefits of this format are covered in Sect. 3, where we define routing preferences formally. Intuitively, *master* refers to a travel cost feature, e.g., travel time or fuel consumption, that a driver aims to minimize, while *slaves* refer to routing preferences, e.g., using versus avoiding highways or using versus avoiding toll roads. For example, a routing preference $\langle$travel_time, no_highways$\rangle$ indicates that a driver aims to minimize travel time and prefers to avoid highways.

Drivers may have different routing preferences in different contexts. Thus, we aim at identifying the most appropriate routing preference for each distinct context. To achieve this, two non-trivial challenges must be addressed.

Learning with large amounts of data Given a popular context, e.g., from a downtown source s_i to a destination d_j in a large residential region during peak hours and in non-snow condition, a large volume of historical trajectories may exist that conform to that context. We aim to employ these trajectories as training data for learning a routing preference for the context. However, the learning can be time-consuming due to the data volume. Using simply some of the data may yield inaccurate or even incorrect results. Thus, how to efficiently utilize all available trajectories and maintain high accuracy is challenging.

To contend with this, we propose an efficient stochastic coordinate descent learning approach and an efficient routing algorithm based on preference-based contraction hierarchies (Sect. 4).

Learning with no data Next, we must contend with the opposite scenario where no trajectories exist for a specific context. Even with large volumes of trajectory data, it is near impossible to have trajectories that cover every single context, especially when considering contexts at a fine granularity. A promising approach to addressing this problem is to transfer routing preferences learned for some contexts to similar contexts, where no trajectories exist and thus where no routing preferences can be learned directly from trajectories. When considering contexts at a fine granularity, the context space becomes very large and sparse, thus making routing preference transfer with high accuracy challenging.

Representing the relationships between contexts and routing preferences by a tensor, we propose a multi-task learning framework based on the tensor. The framework transfers routing preferences from contexts with trajectories to similar contexts without trajectories (Sect. 5).

Finally, for each distinct context, we obtain a routing preference. This in turn enables the on-line query phase of context-aware, preference-based routing. Using the query input, e.g., source, destination, departure time, and driver identifier, we are able to identify a context and then the corresponding routing preference. Next, we find a path according to the routing preference using the preference-based contraction hierarchies.

To the best of our knowledge, this is the first comprehensive solution to context-aware, preference-based routing, and we make five specific contributions. First, we define a contextual preference tensor to model the relationships between contexts and routing preferences. Second, we propose techniques to learn routing preferences effectively and efficiently given large volumes of trajectories. Third, we present a novel multi-task learning approach to transfer routing preferences to contexts where no trajectories are available. Fourth, we propose a novel routing algorithm that uses preference-based contraction hierarchies. Fifth, we provide an in-depth empirical study of the effectiveness and efficiency of the proposed techniques using real-world GPS trajectories.

2 Preliminaries

We define key concepts, state the problem, and give a solution overview.

A **road network** is a directed, weighted graph $\mathcal{G} = (\mathbb{V}, \mathbb{E}, \mathbb{W}_m, \mathbb{W}_s)$, where vertex set $\mathbb{V}$ consists of vertices representing road intersections, edge set $\mathbb{E} \subseteq \mathbb{V} \times \mathbb{V}$ consists of edges representing road segments, and $\mathbb{W}_m$ and $\mathbb{W}_s$ are sets of weight functions, where each function has signature $\mathbb{E} \rightarrow \mathbb{R}^+$.

In particular, $\mathbb{W}_m$ maintains travel costs of edges, which are used later as *master features* in routing preferences; and $\mathbb{W}_s$ maintains road condition attributes (e.g., road types and toll fees) of edges, which are used later as *slave features* in routing preferences.

Specifically, we maintain three commonly used functions in $\mathbb{W}_m$ for road networks [11,12]. Functions $w_{DI}(\cdot)$, $w_{TT}(\cdot)$, and $w_{FC}(\cdot)$ return the distance (DI), travel time (TT), and fuel consumption (FC) of the argument edge, respectively.

To distinguish peak versus off-peak traffic, we maintain two versions of $w_{TT}(\cdot)$ and $w_{FC}(\cdot)$ for peak versus off-peak intervals, respectively. We maintain function $w_{RT}(\cdot)$ in $\mathbb{W}_s$, which returns the road type (RT) of the argument edge.

A **path** $P = \langle u_1, u_2, \ldots, u_i \rangle$ is a sequence of vertices where two consecutive vertices are connected by an edge.

A **trajectory** T is a time-ordered sequence of GPS records capturing the movement of an object, where a GPS record captures the location of the object at a time point. In addition, a trajectory is often associated with a vehicle or driver identifier. The time gap between two consecutive GPS records in trajectories varies, from a few seconds (a.k.a., high-frequency trajectories) to tens of seconds or a few minutes (a.k.a., low-frequency trajectories). In the experiments, we test the proposed method on both kinds of GPS data. Map matching [13] is used to align a trajectory with the road network path that the trajectory traversed.

Problem definition A context-aware, preference-based routing query is defined as $CPR(v_s, v_d, OCI)$. Given an arbitrary source and destination pair (v_s, v_d) and optional contextual information OCI such as a departure time and a driver identifier, the problem is to identify and use an appropriate routing preference for routing, such that the identified paths are similar to the paths chosen by local drivers.

Solution overview The solution to the CPR problem is outlined in Fig. 1.

The framework consists of an *off-line* phase and an *on-line* phase. In the off-line phase, we aim at identifying a routing preference for each context. We use a contextual preference tensor to organize contexts and their routing preferences. For each context, if trajectories exist for the context, the *Routing Preference Learning* module learns a routing preference based on the available trajectories and enters the routing preference into the corresponding entry in the tensor. For example, given a context (R_1, R_2, [8:00, 9:00], D_1), if trajectories exist that start in region R_1, end in region R_2 during 8:00 to 9:00, and belong to driver D_1, we learn a routing preference from such trajectories and associate the preference with the context in the tensor. Since trajectories cannot cover all contexts, the module output is a *sparse tensor*, meaning that some contexts have no routing preferences. Next, the *Routing Preference Transfer* module transfers routing preferences from contexts with trajectories to similar contexts without trajectories, which then yields a *full tensor*, where every context has a routing preference.

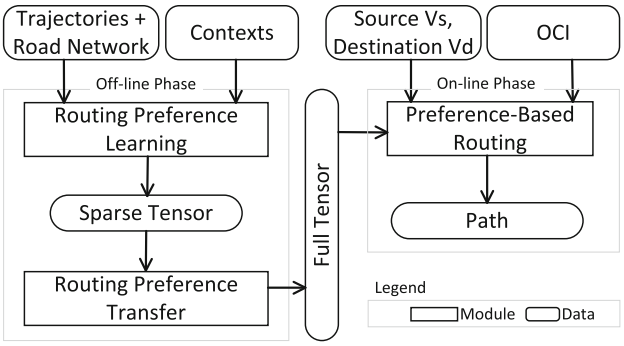


Fig. 1 Solution overview

In the *on-line* phase, a user provides a source vertex v_s and a destination vertex v_d and may provide optional contextual information OCI such as a departure time. The *Preference-Based Routing* module identifies an appropriate routing preference from the full contextual preference tensor and uses the preference to obtain a path from v_s to v_d.

Algorithm 1 captures the off-line phase, and the on-line phase is presented in Algorithm 5.

Algorithm 1: OfflineTraining

Input: Road network $\mathcal{G}$; Contextual preference Tensor **T**;
Output: Contextual preference Tensor **T**;
$\mathcal{G}_{PCH} \leftarrow BuildingPCH(\mathcal{G})$; /*call Alg. 4*/
Initialize an empty tensor **T**; /*Sec. 3*/
for *each context c in tensor* ***T*** **do**
  /*Sec. 4*/
  if *Trajectories are available in context c* **then**
    Use Alg. 3 to identify a routing preference P; /*Alg. 3 calls Alg. 2 which in turn calls Alg. 5 that uses $\mathcal{G}_{PCH}$*/
    $\mathbf{T}(c) \leftarrow P$;
Transfer routing preferences from contexts with trajectories to similar contexts without trajectories; /*Sec. 5*/
return T;

3 Contextual preference tensor

3.1 Contexts

A ***context*** describes the setting at the time a trip starts that affects drivers' routing preferences. We consider three different types of contexts—*spatial*, *temporal*, and *other contexts*.

Spatial context The spatial context includes information about where trips start and end, i.e., source–destination pairs, which have a significant influence on how drivers choose paths. For example, a driver may prefer to use highways when

traveling from downtown to a far-away suburb, but may prefer to not use highways when traveling between downtown locations.

Each vertex in a road network can be a source and a destination, resulting in $|\mathbb{V}|^2$ possible source–destination pairs. Considering all such pairs in real-world road network settings yields a very large spatial context space, and it is very unlikely that even a very large trajectory set is able to cover all $|\mathbb{V}|^2$ pairs. Therefore, we instead partition a road network into regions and use the set of region pairs as the spatial context. The cardinality of the region set, denoted as N, is much smaller than the cardinality of the vertex set $|\mathbb{V}|$. Thus, the spatial context has $N \times N$ region pairs as its instances.

A wide variety of partitioning methods, ranging from simple uniform grids to complex semantic-based methods [14–16], are applicable here. Our method is generic and does not assume any specific region definition. Thus, we simply partition a road network into regions of uniform sizes. Then, when users provide source and destination vertices v_s and v_d, we obtain and use the regions R_s and R_d that cover v_s and v_d as the spatial instance.

The spatial context is mandatory—when a user issues a *CPR* query, the query always includes source and destination vertices v_s and v_d, which are then used for deriving a spatial context.

Temporal contexts Since traffic is time dependent [9,10], we consider *temporal contexts* such as departure time, the day of the week, and whether it is a holiday. In the *CPR* query, such temporal information is provided optionally through the *OCI* argument.

Other contexts Other contexts include, for example, weather conditions and driver identifiers. Some weather conditions, e.g., snow and heavy rain, affect traffic significantly and thus may also affect drivers' choices of paths; and different drivers choose different paths. Such additional context information is also provided optionally through the *OCI*.

We model the optional contexts in a unified manner. Let $\mathbb{S} = \{S_1, S_2, \ldots, S_M\}$ denote a set of M different optional contexts. For example, we consider $\mathbb{S} = \{S_1, S_2, S_3\}$, where S_1 represents departure time, S_2 represents weather conditions, and S_3 represents driver identifiers.

Next, for each optional context $S_i \in \mathbb{S}$, $I(S_i)$ denotes the set of all possible instances of S_i. For example, for departure time context S_1, if we only distinguish departure times during peak versus off-peak hours, then we have $I(S_1) = \{Peak, Off\text{-}Peak, null\}$, where *null* represents the case where no departure time is provided; if we consider 15-min intervals, $I(S_1)$ has 97 instances, namely the 96 15-min intervals, and an unknown interval *null*.

Based on the above, the optional context space is of size $Q = |I(S_1)| \times |I(S_2)| \times \ldots \times |I(S_M)|$. Thus, the whole context space has size $\Theta(N \times N \times Q)$.

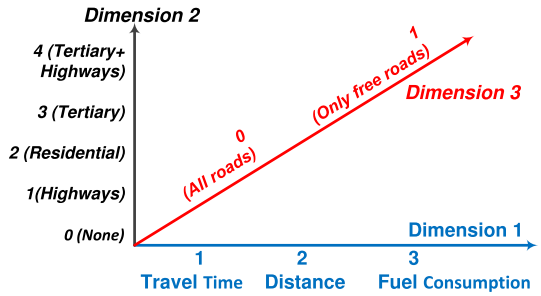


Fig. 2 Routing preference space

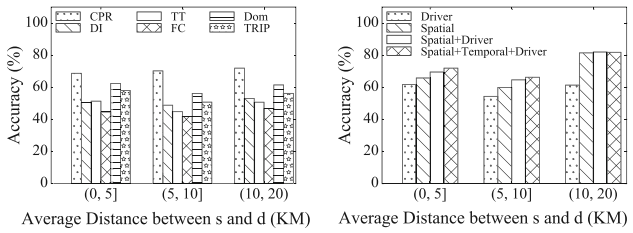
3.2 Routing preferences

A **routing preference** encompasses a travel cost that a driver aims to minimize and a number of route properties. We model such a routing preference as a vector in a multi-dimensional routing feature space. The first dimension models the travel costs that a driver would like to minimize, and the remaining dimensions model route properties that are able to express a driver's routing preferences.

Figure 2 shows an example routing feature space with three dimensions. The travel cost dimension is the *master* dimension, meaning that a travel cost, e.g., travel time, distance, or fuel consumption, must be specified that a routing algorithm then can minimize. The remaining dimensions are optional *slave* dimensions, meaning that it is possible to perform routing even if no features in these dimensions are specified. Thus, we represent a routing preference as a vector $V = \langle Master, Slaves \rangle$, with exactly one master feature and zero or more slave features. We consider two slave dimensions—road types and toll fees. For example, vector (1, 2, 0) indicates a routing preference for optimizing on travel time, using residential roads, and with no preference regarding toll fees.

Next, we justify the proposed routing preference format with reference to Fig. 3a. We split a trajectory set into a training set and a testing set. Using the training trajectories, we learn a single routing preference (to be detailed in Sect. 4). Then, for each testing trajectory, we identify the optimal path that connects its source and destination using the learned routing preference. We compare the similarity between the optimal path and the path used by the testing trajectory. If the similarity is high, we regard the learned routing preference as being accurate.

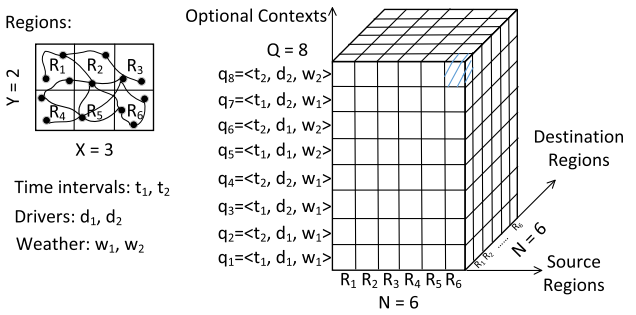
We consider five alternative forms of routing preferences. DI, TT, and FC represent simple routing preferences that minimize distance, travel time, and fuel consumption, respectively. TRIP [17] identifies routing preferences that are able to customize travel times and then minimizes travel time based on the customized travel times. Dom [6] identifies routing preferences that describe the trade-off among distance, travel time, and fuel consumption and then returns the optimal path



(a) Different Formats of Routing Preferences

(b) Routing Preferences With Contexts

Fig. 3 Routing preferences versus contexts



(a) Context Instances

(b) Contextual Preference Tensor

Fig. 4 Contextual preference tensor

based on the trade-off. Figure 3a suggests that the proposed routing preferences (denoted as CPR) achieve the best accuracy in all three settings when source s and destination d are at most 5, 10, or 20 km apart. This suggests that the proposed type of routing preference enables accurate routing.

3.3 Contextual preference tensor

Drivers may have different preferences in different contexts. To explore this assumption, we proceed with the study mentioned above. Rather than learning a single routing preference, we consider the following four settings. *Driver:* we learn a routing preference for each driver using the corresponding training trajectories, i.e., the trajectories from the driver; *Spatial:* we learn a routing preference between a pair of regions using the corresponding training trajectories; $Spatial + Driver$: we learn a routing preference for a specific driver for a specific region pair using the corresponding training trajectories; $spatial + temporal + driver$: we learn a routing preference for a driver, during a time interval, and between a pair of regions using the corresponding training trajectories. Next, we apply the routing preferences to identify paths and compare with the paths used in the testing trajectories. Figure 3b suggests that when the granularity of contexts becomes finer, we obtain more accurate routes.

To capture the relationship between contexts and routing preferences, we propose a **contextual preference tensor**, which is a three-dimensional tensor $\mathbf{T} \in \mathbb{R}^{N \times N \times Q}$, whose dimensions represent source regions, destination regions, and optional contexts, respectively. Recall that N denotes the total number of regions and Q represents the total number of optional context instances. Each entry in the tensor represents a routing preferences under the corresponding context.

In the example contextual preference tensor shown in Fig. 4a, the road network is partitioned into a 3×2 grid, which gives $N = 6$ regions $R_1, R_2, \ldots, R_6$. Next, each optional context has 2 instances—2 for departure time intervals, 2 for drivers, and 2 for weather conditions. Then, we have $Q = 2 \times 2 \times 2 = 8$.

Figure 4b shows the contextual preference tensor $\mathbf{T} \in \mathbb{R}^{6 \times 6 \times 8}$, where the two first dimensions represent the source and destination regions and the third dimension represents the optional contexts.

The shaded entry represents the routing preference under context $\langle R_6, R_1, q_8 \rangle$, where $q_8 = \langle t_2, d_2, w_2 \rangle$ denotes the context of traveling from region R_6 to region R_1, leaving during time interval t_2, with driver d_2 traveling, and when the weather condition is w_2.

The values in a dimension do not indicate an order. For example, in the third dimension, the 8 values are simply 8 distinct optional context instances, and q_1 is not smaller than q_2 although q_1 appears before q_2.

Since the choice of optional context information is flexible, the contextual preference tensor is generic and is able to cover a wide variety of use cases. For example, if the optional contexts only include driver identifiers, the tensor captures personalized routing preferences [17,18], and if the optional contexts only include departure time, the tensor captures time-dependent routing preferences.

To enable context-aware, preference-based routing, we need to populate the tensor with a routing preference for each entry. Given a context, if corresponding trajectories exist, we learn an appropriate routing preference for the context using the trajectories (cf., Sect. 4); and we transfer a routing preference from similar contexts with trajectories to the contexts without trajectories (cf., Sect. 5).

We aim at learning routing preferences from trajectories rather than relying on drivers to manually specify them because although a driver may know the "best" preferences under some contexts, a driver is unlikely to be aware of the best preferences for all contexts, especially when the contexts are in a fine granularity. Thus, it is useful to learn routing preferences in different contexts and recommend these to drivers. In addition, the learned routing preferences can guide autonomous vehicles to perform routing that mimics the routing choices of professional drivers.

4 Learning routing preferences

4.1 Intuition

Given an entry in tensor $\mathbf{T}$, e.g., the shaded context $\langle R_6, R_1, q_8 \rangle$ in Fig. 4, we identify the set of trajectories $\mathbb{TR}$ that occurred in the context. In this section, we consider the case where $\mathbb{TR}$ is non-empty. The other case is considered in Sect. 5.

For each trajectory $T_k \in \mathbb{TR}$, we obtain its path P_k with the help of map matching. This way, we obtain a path set $\mathbb{P}$. For example, a set $\mathbb{TR} = \{T_1, \ldots, T_6\}$ may yield $\mathbb{P} = \{P_1, \ldots, P_6\}$.

We observe that it is often insufficient to directly reuse existing paths in $\mathbb{P}$ to enable context-aware, preference-based routing for a specific context. For context $\langle R_6, R_1, q_8 \rangle$, if there are 5 vertices in R_6 and 4 vertices in R_1, we have $5 \times 4 = 20$ possible source–destination pairs from R_6 to R_1. However, since path set $\mathbb{P}$ has 6 paths, it can capture at most 6 source–destination pairs. For the remaining 14 pairs, we cannot directly reuse paths in $\mathbb{P}$ to enable context-aware, preference-based routing.

Instead, we learn a routing preference vector V from $\mathbb{P}$ and then use V to enable preference-based routing for any source–destination pair in the context. In particular, the preference-based routing algorithm takes as input a routing preference vector V with source v_s and destination v_d, and produces a path P^V that takes into account the preference. Formally, $P^V = \mathit{PrefRouting}(V, v_s, v_d)$.

For example, we learn a routing preference vector V_1 from $\mathbb{P} = \{P_1, \ldots, P_6\}$. When having to perform preference-based routing between v_s in region R_6 and v_d in region R_1, we call $P^{V_1} = \mathit{PrefRouting}(V_1, v_s, v_d)$ to obtain a path.

In Sect. 4.2, we show how to learn a routing preference vector V from the path set $\mathbb{P}$, and in Sect. 4.3, we detail the preference-based routing algorithm, which is a core operator of the routing preference learning.

4.2 Learning optimal preference vectors

A path $P_k \in \mathbb{P}$ has source v_s^k and destination v_d^k. Given a randomly initialized routing preference vector V, we can construct a path $P_k^V = \mathit{PrefRouting}(V, v_s^k, v_d^k)$ that connects v_s^k and v_d^k.

Given P_k^V, we can determine whether V is accurate: if V captures the routing preference under the context well, path P_k^V aligns with the actual, or ground truth, path P_k. We thus utilize the following objective function to evaluate the quality of a routing preference V.

$$O(V) = \sum_{P_k \in \mathbb{P}} \mathit{Sim}(P_k, P_k^V), \text{ where}$$
$$\mathit{Sim}(P_k, P_k^V) = \frac{\sum_{e \in \mathit{LCSS}(P_k, P_k^V)} \mathit{len}(e)}{\sum_{e \in P_k} \mathit{len}(e)} \tag{1}$$

Here, we identify the longest common subsequence (LCSS) between P_k and P_k^V. Then, the longer the sum of the lengths of the edges that exist in the LCSS, the more similar the two paths P_k and P_k^V are.

The goal is to find a routing preference vector V^* that maximizes the objective function. Formally, we solve the following optimization problem.

$$V^* = \arg\max_{V \in \mathbb{V}} O(V), \tag{2}$$

where $\mathbb{V}$ is the set of all possible routing preference vectors that exist in the routing preference space (cf. Fig. 2).

4.2.1 Finding preference vector V^*

A naive method is to use exhaustive search (denoted as *ex*)—we iterate through all possible routing preference vectors in the routing preference space and use all paths in trajectory set $\mathbb{P}$ to compute a score using the objective function in Eq. 1. We return the vector with the largest score. However, the search space can be very large, rendering *ex* inefficient.

As an alternative, we may consider efficient learning algorithms based on either gradient descent or coordinate descent [19]. However, such algorithms are inapplicable in our setting. First, they require a continuous objective function to compute the derivative to guide the search, but our objective function (i.e., Eq. 1) is not continuous. Second, they rely on line searches to explore specific directions, but our space (shown in Fig. 2) does not offer such directions. The values in our dimensions are not ordered, and one value is not smaller or larger than another value.

Instead, we propose an efficient approach (denoted as *co*) that resembles coordinate descent [19]. We identify the best feature for the master dimension and then the best feature for each slave dimension. However, instead of using line search, we consider all features when identifying the best feature in a dimension. Specifically, when identifying the best feature in the kth dimension, we keep the already identified best features for the first $k-1$ dimensions, consider all possible features in the kth dimension, and disregard the features in the remaining dimensions. We use all paths in $\mathbb{P}$ to compute a score according to Eq. 1 for each feature in a dimension, this way identifying the best feature for the dimension.

We also propose a more efficient approach (denoted as *st*) that is motivated by stochastic coordinate descent [20]. When identifying the best feature in a dimension, we use only some of the paths in $\mathbb{P}$ to compute a score for a feature using

Eq. 1. We identify the best feature by applying the learning procedure until the best score cannot be further improved. In the worst case, all paths in $\mathbb{P}$ are used for computing the score for a feature. However, it is often the case that we do not need to use all paths.

To characterize the computational complexity of the three above approaches, assume that the routing preference space has n dimensions, each with at most x features, and assume that the running time of preference-based routing is PR. Then, we have the following complexity: $O(x^n \cdot |\mathbb{P}| \cdot PR)$ for *ex*, $O(x \cdot n \cdot |\mathbb{P}| \cdot PR)$ for *co*, and $O(x \cdot n \cdot |\mathbb{P}'| \cdot PR)$ for *st*, where $\mathbb{P}' \subseteq \mathbb{P}$. We note that *ex* guarantees optimality, while *co* and *st* do not. We evaluate the accuracy of the three approaches empirically in Sect. 6.2.

4.2.2 Specifics of stochastic coordinate descent

We start with the master dimension and then consider the slave dimensions. Using the example in Fig. 2, we first identify the best travel cost feature from the travel cost dimension. Specifically, we enumerate the routing preference vectors as $V_{TT} = \langle 1, 0, 0\rangle$, $V_{DI} = \langle 2, 0, 0\rangle$, and $V_{FC} = \langle 3, 0, 0\rangle$, indicating that optimizing travel time (TT), distance (DI), and fuel consumption (FC) are preferred, respectively, and that other slave features are not considered.

We proceed to identify an optimal vector V^* from set $\mathbb{V} = \{V_{TT}, V_{DI}, V_{FC}\}$ that consists of the above feature vectors using paths in $\mathbb{P}$. The idea is to compute, for each candidate preference vector in $\mathbb{V}$, a similarity score using some paths in $\mathbb{P}$. The preference vector with the highest similarity score is the optimal vector. The pseudocode is shown in Algorithm 2.

Algorithm 2: StochasticPrefLearning

Input: Path set: $\mathbb{P}$; possible vectors: $\mathbb{V}$;
Output: Optimal routing preference vector: V^*;

1 $V^* \leftarrow \emptyset$; $sim^* \leftarrow 0$; $j \leftarrow 0$;
2 **while** sim^* *is not converged* **do**
3 Double[$|\mathbb{V}|$] $sims \leftarrow \{0\}$;
4 Integer[$|\mathbb{V}|$] $count \leftarrow \{0\}$;
5 **for** *each* P_k *in* $\mathbb{P}$, *starting from* P_j **do**
6 get source, destination $s_k, t_k \leftarrow P_k$;
7 /*k-th path for m-th vector*/
8 $m \leftarrow (k\%|\mathbb{P}|)\%|\mathbb{V}|$;
9 $P_k^{V_m} \leftarrow PrefRouting(s_k, t_k, V_m)$;
10 $sim_k^m \leftarrow Sim(P_k, P_k^{V_m})$;
11 $sims[m] \leftarrow sims[m] + sim_k^m$;
12 $count[m] \leftarrow count[m] + 1$;
13 **for** *each* $sims[m]$ *in* $sims$ **do**
14 $sims[m] \leftarrow sims[m]/count[m]$;
15 $sim^* \leftarrow \arg\max sims[m]$; $V^* \leftarrow V_{sim^*}$;
16 $j \leftarrow (j+1)\%|\mathbb{P}|$;
17 **return** V^*;

To explain how we use the paths in $\mathbb{P}$ in a stochastic manner, we continue with the example with six paths $\mathbb{P} = \{P_1, \ldots, P_6\}$ for context $\langle R_6, R_1, q_8\rangle$. Instead of using all 6 paths to get an average similarity $\frac{\sum_{k=1}^{6} sim_k^m}{6}$ for each vector V_m, we iteratively apply the 6 paths to the 3 candidate vectors in a round-robin manner. This design aims at improving the learning efficiency without compromising the learning quality.

For example, we apply paths P_1 and P_4 to vector V_{TT}, P_2 and P_5 to vector V_{DI}, and P_3 and P_6 to vector V_{FC}, thus obtaining $\frac{\sum_{k=\{1,4\}} sim_k^{TT}}{2}$, $\frac{\sum_{k=\{2,5\}} sim_k^{DI}}{2}$, and $\frac{\sum_{k=\{3,6\}} sim_k^{FC}}{2}$ as three average similarities. In the next iteration, we move the starting path to the next path (line 16) and repeat the process. In the example, we get $\frac{\sum_{k=\{2,5\}} sim_k^{TT}}{2}$, $\frac{\sum_{k=\{3,6\}} sim_k^{DI}}{2}$, and $\frac{\sum_{k=\{1,4\}} sim_k^{FC}}{2}$. This process continues until the similarities converge (line 2).

Next, we cover how to compute similarity sim_k^m. Given a path $P_k \in \mathbb{P}$ from source s_k to destination t_k, we call function *PrefRouting* to construct a path $P_k^{V_m}$ from s_k to t_k while using preference vector V_m (line 9). The specifics of *PrefRouting* are detailed in Algorithm 5 in Sect. 4.3. After obtaining $P_k^{V_m}$, we calculate similarity sim_k^m between path P_k and $P_k^{V_m}$ using Eq. 1 in Sect. 4.2.

Finally, we get an optimal vector that has the highest similarity, and the corresponding master travel cost feature can also be determined. For example, if $V^* = V_{DI} = \langle 2, 0, 0\rangle$ then the optimal master travel cost feature is $m^* = DI$. This means that among the shortest paths, fastest paths, and most fuel efficient paths, the shortest paths are most similar to the paths in $\mathbb{P}$.

Having chosen the optimal master feature m^* and obtained its corresponding vector V^*, we proceed to learn the optimal features from the remaining slave dimensions. The intuition is that since the master dimension is identified, e.g., is *DI*, we aim to determine whether considering additional slave features of the shortest paths yields paths that match the paths in $\mathbb{P}$ better than do the shortest paths. For example, if shortest paths using only toll free edges match the paths in $\mathbb{P}$ better than do shortest paths using all edges then "toll free" should be included as a slave feature in the final optimal routing preference vector.

Assume that n slave dimensions exist and that slave features from the first $i-1$ slave dimension, $1 \leq i \leq n$, have been chosen, and thus are fixed. The next step is to identify an optimal feature s_i^* from the features in the ith dimension. The overall procedure for learning a routing preference given a set of paths and a routing preference space is shown in Algorithm 3.

Following the example for the master dimension, we identify the best feature for the road type dimension. Based on the optimal vector $V^* = \langle 2, 0, 0\rangle$ learned at the master

Algorithm 3: RoutingPrefLearning

Input: Path set: $\mathbb{P}$, Routing preference space: $\langle Dom_{md}, Dom_{s_1}, \cdots, Dom_{s_n} \rangle$;
Output: Optimal routing preference vector: V_{max};

```
/* Master dimension */
V_md ← EnumVectors(Dom_md);
V* ← StochasticPrefLearning(ℙ, V_md);
/* Slave dimensions */
V_s1 ← EnumVectors(V*, Dom_s1);
V*_s1 ← StochasticPrefLearning(ℙ, V_s1);
for each slave dimension s_i (i ∈ [2, n]) do
    V_si ← EnumVectors(V*_{s_{i-1}}, Dom_si);
    V*_si ← StochasticPrefLearning(ℙ, V_si);
return V*_sn;
```

dimension, we enumerate vectors as $V_N = \langle 2, \mathbf{0}, 0 \rangle$, $V_H = \langle 2, \mathbf{1}, 0 \rangle$, $V_R = \langle 2, \mathbf{2}, 0 \rangle$, and $V_T = \langle 2, \mathbf{3}, 0 \rangle$, meaning that the shortest path search is combined with a preference for nothing, highways, residential roads, or tertiary roads, respectively. Then, we feed $\mathbb{P}$ and $\mathbb{V} = \{V_N, V_H, V_R, V_T\}$ to the stochastic preference learning algorithm (cf. Algorithm 2) to learn an optimal routing preference vector, e.g., $V^*_{s_1} = V_H$, which enables us to identify the optimal road type feature, e.g., highways.

We use the optimal vector learned using the last slave dimension $V^*_{s_n}$ as the final optimal routing preference vector. Here, $V^*_{s_n}$ considers (1) the optimal master feature m^* on the master dimension, (2) the optimal features $s^*_1, \cdots, s^*_{n-1}$ on the first $n-1$ slave dimensions, and (3) the optimal feature s^*_n on the nth slave dimension.

Routing preference vector $V^*_{s_n}$ is inserted into the corresponding entry in the contextual preference tensor.

For example, we obtain preference vector $V_1 = \langle 2, 1, 0 \rangle$ for context $\langle R_6, R_1, q_8 \rangle$, meaning that the local drivers prefer shortest paths and highways when traveling from region R_6 to region R_1 given optional context q_8.

4.3 Preference-based routing

Preference-based routing $P^V = PrefRouting(V, v_s, v_d)$ plays an important role when identifying optimal preference vectors. Since preference-based routing is expensive, to ensure efficient learning, we provide means of making preference-based routing efficient.

We speed up preference-based routing by introducing *preference-based contraction hierarchies (PCH)*. Classical contraction hierarchies (CH) [21] encompass an indexing phase and a querying phase. In the indexing phase, so-called shortcut edges are built for a road network $\mathcal{G}$. The weight of a shortcut is the sum of the weights of the edges that the shortcut represents. In the querying phase, bi-directional Dijkstra's search is applied to the union of the original edges in $\mathcal{G}$ and the shortcuts.

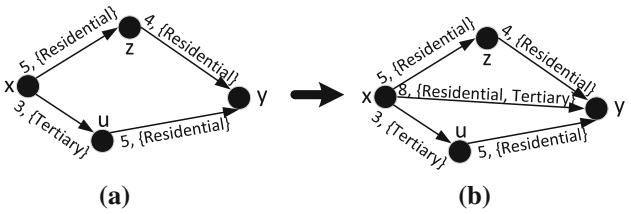


Fig. 5 Preference-based contraction hierarchies

CH only considers edge weights, i.e., master features such as travel time, distance, or fuel consumption, not slave features such as road types. In our setting, it is essential that slave features are also captured by shortcuts, so that preference-based routing is able to make use of them to find paths that not only optimizing a travel cost, but also satisfy different slave features. To this end, we propose preference-based contraction hierarchies (*PCH*).

4.3.1 Building PCH

Given a road network $\mathcal{G} = (\mathbb{V}, \mathbb{E}, \mathbb{W}_m, \mathbb{W}_s)$, where $\mathbb{W}_m$ maintains three weight functions: $w_{DI}(\cdot)$, $w_{TT}(\cdot)$, and $w_{FC}(\cdot)$, we construct a *PCH* for each travel cost. In the following, we use weight function $w_{DI}(\cdot)$ to illustrate the construction process.

Figure 5a shows 4 vertices u, x, y, and z, and each edge is labeled with its length and road type, e.g., "4, {*Residential*}" for the edge (y, z). As in CH, the indexing phase follows a pre-defined vertex order. Different heuristics [21] are proposed to order vertices, and they consider different characteristics such as the edge difference and uniformity. While any heuristic would work, we order vertices by edge difference, i.e., the difference between the number of shortcuts introduced when contracting a vertex u and the number of edges incident to u [21] .

We take vertex u as an example and consider whether we can add a shortcut between its adjacent vertices x and y. We use $w_{DI}(x, y) = w_{DI}(x, u) + w_{DI}(u, y)$ to denote the weight of path $\langle x, u, y \rangle$, which is 8. We then issue a so-called witness path search to see if we can identify another path from x to y without going through u. In this example, we find that $\langle x, z, y \rangle$ is a witness path that has weight 9. If $w_{DI}(x, y)$ is smaller than the weights of all witness paths, we insert a shortcut edge between x and y with weight $w_{DI}(x, y)$. Thus, we insert a shortcut edge (x, y) with weight 8, as shown in Fig. 5b.

Next, we construct the slave features of a shortcut as the union of the slave features of the edges in the shortcut path. In this example, $w_{RT}(x, y) = w_{RT}(x, u) \cup w_{RT}(u, y) = \{Residential, Tertiary\}$.

Algorithm 4 details the process of constructing the preference-based contraction hierarchies. We use $w_*(\cdot)$ to

denote one of the travel cost weight functions $w_{DI}(\cdot)$, $w_{TT}(\cdot)$, or $w_{FC}(\cdot)$.

Given a road network $\mathcal{G} = (\mathbb{V}, \mathbb{E}, \mathbb{W}_m, \mathbb{W}_s)$, the algorithm establishes a new road network $\mathcal{G}_{PCH}$ by adding shortcut edges to the original edge set. In addition, it records both weights and slave features of the shortcuts in $\mathbb{W}'_m$ and $\mathbb{W}'_s$, respectively. Including additional slave features into $\mathcal{G}_{PCH}$ does not violate the vertex order utilized by the classic CH; thus, preference-based routing is able to adopt the principle of the bi-directional Dijkstra's search with the same vertex order. Different heuristics [21] are proposed to assign orders to vertices, and they consider characteristics such as edge difference and uniformity. Any such heuristics works here. Thus, we simply use the default heuristics.

Algorithm 4: BuildingPCH

```
   Input: Road network G = (V, E, W_m, W_s);
   Output: Road network with PCH G_PCH = (V, E', W'_m, W'_s)
1  E' ← E; W'_m ← W_m; W'_s ← W_s;
2  Initialize a priority queue Q for vertices in V based on the vertex
   order;
3  while Q ≠ ∅ do
4      vertex u ← Q.extractMin();
5      for each higher-order vertex x that is adjacent to u do
6          for each higher-order vertex y that is adjacent to u ∧
           x ≠ y do
7              w_*(x, y) ← w_*(x, u) + w_*(u, y);
8              if w_*(x, y) is smaller than all witness paths then
9                  E' ← E' ∪ {(x, y)};
10                 W'_m ← W'_m ∪ {w_*(x, y)};
11                 for each slave feature s do
12                     w'_s(x, y) ← w'_s(x, u) ∪ w'_s(u, y);
13                     W'_s ← W'_s ∪ {w'_s(x, y)};
14     Update the vertex order based on the newly added shortcut
       edge;
15 Return G_PCH;
```

In each iteration, the vertex u with the currently lowest order is used for constructing shortcuts (line 4). Next, for each pair of vertices x and y that are adjacent to u and have higher orders, we compute a weight $w_*(x, y)$ as the sum of $w_*(x, u)$ and $w_*(u, y)$ (lines 5–7). If we cannot find a witness path between x and y that does not pass through u, and whose weight is less than $w_*(x, y)$, we build a new shortcut edge (x, y) (lines 8–13). In particular, we add the edge (x, y) to edge set $\mathbb{E}'$ and record its corresponding weight and slave features. Finally, we update the vertex order since the newly added shortcut edge changes the previous vertex adjacency relationships (line 14).

The complexity of Algorithm 4 is $O(|\mathbb{V}| \cdot \lg |\mathbb{V}| \cdot N^2 \cdot \mathit{WitnessSearch})$, where $|\mathbb{V}|$ denotes the number of vertices in the road network graph $\mathcal{G}$, N is the largest degree of any vertex in graph $\mathcal{G}$, and *WitnessSearch* represents the complexity of a witness search. In a road network, the largest degree of a vertex is typically small, e.g., 3 to 5, and less than 10. Due to physical constraints, an intersection cannot connect, say, 1,000 streets. Thus, N can be considered a constant. The witness search is often conducted by running a Dijkstra's search within a neighborhood of vertex u, e.g., within 100 hops of vertex u. Assuming that the neighborhood is a subgraph with x vertices and y edges, where $x \ll |\mathbb{V}|$ and $y \ll |\mathbb{E}|$, the complexity of the Dijkstra's search in the subgraph is $O(y + x \cdot \lg x)$. Thus, the complexity of Algorithm 4 is $O(|\mathbb{V}| \cdot \lg |\mathbb{V}| \cdot (y + x \cdot \lg x))$.

The end result is a contracted road network $\mathcal{G}_{PCH} = (\mathbb{V}, \mathbb{E}', \mathbb{W}'_m, \mathbb{W}'_s)$, where $\mathbb{V}$ is the vertex set of $\mathcal{G}$, $\mathbb{E}'$ is the union of the original edge set $\mathbb{E}$ and the set of shortcut edges, and $\mathbb{W}'_m$ and $\mathbb{W}'_s$ record the travel costs and the slave features of $\mathbb{E}'$.

4.3.2 Preference-based routing on PCH

We proceed to propose a preference-based routing algorithm that is able to use the *PCH*. In particular, we utilize the bi-directional Dijkstra's algorithm as a basis for enabling preference-based routing. The pseudocode is shown in Algorithm 5.

The algorithm explores the road network from both source and destination—the upward search from source v_s only explores vertices whose orders exceed that of the current vertex, and the downward search from destination v_d only explores vertices whose orders are lower than that of the current vertex. The algorithm maintains two priority queues Q_U and Q_D for the two searches (lines 1–5). The priority is based on the travel cost specified in preference vector V. For example, if $V = \langle 1, 1, 0 \rangle$, travel time (cf. Fig. 2) is used as the priority.

The algorithm stops when a vertex u_U (u_D) explored by the upward (downward) search meets v_d (v_s) or appears in Q_D (Q_U), and the algorithm constructs a path P from u_U (u_D) using its upward and downward parent attributes (lines 8–9 or lines 18–19).

In each search (lines 7–16 or lines 17–26), the algorithm chooses to explore an adjacent vertex x of the current vertex that also satisfies the vertex order constraints (i.e., higher order for the upward search and lower order for the downward search). When we explore adjacent vertices of a vertex u_U, we differentiate two cases: (i) at least one edge exists that satisfies the slave preferences specified in V; and (ii) no edge exists that satisfies the slave preferences specified in V.

For case (i), if edge (u_U, x) satisfy $V.\mathit{slaves}$, we explore the edge and add vertex x to priority queue Q_u. This can be checked by using the slave features maintained for w'_m, which also include shortcuts. The intuition is that since we find edges that satisfy the slave features, we explore such edges to find optimal paths.

Algorithm 5: PrefRouting

Input: Preference Vector: V; source and destination vertices: v_s, v_d; contracted road network: $\mathcal{G}_{PCH}$;
Output: Path P from v_s to v_d

```
for each vertex v ∈ G_PCH.V do
    v.Ucost ← +∞; v.Uparent ← null;
    v.Dcost ← +∞; v.Dparent ← null;
v_s.Ucost ← 0; v_d.Dcost ← 0;
Q_U.insert(v_s); Q_D.insert(v_d);
while Q_U ≠ ∅ ∧ Q_D ≠ ∅ do
    vertex u_U ← Q_U.ExtractMin();
    if u_U = v_d ∨ u_U ∈ Q_D then
        construct P from u_U and return;
    Boolean noneSat ← false;
    if there does not exist a higher-order vertex x such that x is
    u_U's adjacent vertex and w'_s(u_U, x) satisfies V.slave then
        noneSat ← true;
    for each higher-order vertex x that is adjacent to u_U do
        if w'_s(u_U, x) satisfies V.slave ∨ noneSat then
            if u_U.Ucost + w_{V.master}(u_U, x) < x.Ucost then
                x.Ucost ← u_U.Ucost + w_{V.master}(u_U, x);
                x.Uparent ← u_U; Q_U.insert(x);
    vertex u_D ← Q_D.ExtractMin();
    if u_D = v_s ∨ u_D ∈ Q_U then
        construct P from u_D and return;
    Boolean noneSat ← false;
    if there does not exist a lower order vertex x such that x is
    u_D's adjacent vertex and w'_s(u_D, x) satisfies V.slave then
        noneSat ← true;
    for each lower order vertex x that is adjacent to u_D do
        if w'_s(u_D, x) satisfies V.slave ∨ noneSat then
            if u_D.Dcost + w_{V.master}(u_D, x) < x.Dcost then
                x.Dcost ← u_D.Dcost + w_{V.master}(u_D, x);
                x.Dparent ← u_D; Q_D.insert(x);
```

For case (ii), as long as u_U's adjacent vertices satisfy the order constraints, they are inserted into priority queue Q_u. The intuition is that since no edge satisfies $V.slaves$, we explore all edges in order to ensure that the bi-directional search proceeds until the two searches meet. This way, we ensure that the preferences on both the master and slave dimensions are accommodated by the algorithm.

5 Transferring preferences

So far, we have populated the tensor entries for which corresponding trajectories exist. We call such entries *known entries*. For the remaining *unknown entries*, no corresponding trajectories exist. In this section, we present two frameworks to transfer labels from known to unknown entries to obtain a *full tensor*.

In particular, the basic framework (Sect. 5.1) is a naive implementation of graph-based learning to transfer labels.

Table 1 Basic versus advanced frameworks

	Basic framework	Advanced framework
# Entry graphs	1	N
# Nodes per graph	$N^2 \cdot Q$	$N \cdot Q$
Similarity matrix size per graph	$(N^2 \cdot Q) \times (N^2 \cdot Q)$	$(N \cdot Q) \times (N \cdot Q)$
Space overhead	$(N^2 \cdot Q) \times (N^2 \cdot Q)$	N^2

The basic framework faces two challenges that are solved by the advanced framework (Sect. 5.2). The basic framework is used as a baseline to compare with the advanced framework in Sect. 6.3.

5.1 Basic framework

We build an **entry graph** from the sparse contextual preference tensor. In this graph, a node represents an entry in the tensor. A node for a known entry is labeled with the corresponding routing preference, and a node for an unknown entry has no label. The goal is then to transfer labels, i.e., routing preferences, to nodes without labels based on the intuition that similar entries tend to share labels.

To achieve this goal, we can adopt a learning algorithm, e.g., graph-based transduction learning [22] or graph convolutional networks [23–25], to transfer labels from known entries to *similar*, but unknown, entries. However, we need to contend with two challenges.

Tricky similarity functions: a similarity function needs to measure the similarity between two nodes, i.e., contexts. It needs to combine similarities among the spatial, temporal, and other contexts. Weighting among these contexts is non-trivial, which also makes transferring accuracy low, as shown in Sect. 6.3.

Very large similarity matrix we have $N^2 \cdot Q$ nodes, yielding a similarity matrix of size $(N^2 \cdot Q) \times (N^2 \cdot Q)$, as shown in Table 1, to represent the similarities of all node pairs. Applying learning to a graph this large is inefficient and does not scale w.r.t. the granularity of contexts, i.e., the number of spatial regions and optional contextual instances.

5.2 Advanced framework

To address the two challenges of the basic framework, we propose an advanced framework that exploits a divide and conquer principle and leverages multi-task learning [26,27]. We first divide the original problem into a number of small tasks, thus addressing the scalability challenge faced by the basic framework. To contend with the trick similarity function challenge, we utilize graph convolutional neural networks when solving a task. Here, the similarity function only

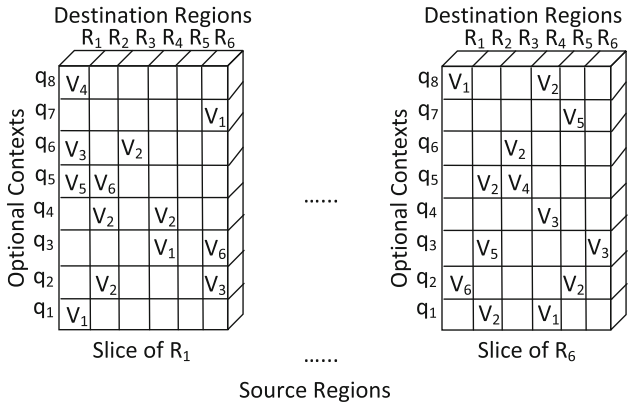


Fig. 6 Advanced framework with slices on source regions

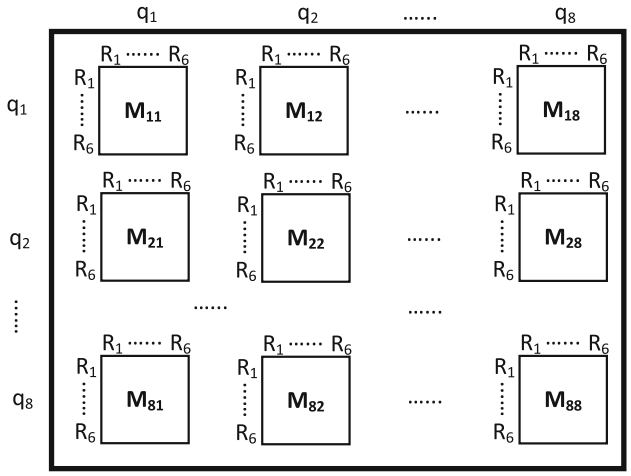


Fig. 7 Similarity matrix **M**

captures spatial contexts, which is then easy to compute. But we do not ignore the temporal and other contexts. Instead, we use them as features for learning.

In particular, we slice the contextual preference tensor and build a small entry graph for each slice. Then, we treat the transfer of routing preferences in each small entry graph as a task. Finally, we solve all tasks together in a multi-task learning setting to obtain a full tensor.

5.2.1 Slicing the tensor

We slice the contextual preference tensor along one dimension. We may use any dimension, but use the source region dimension to illustrate the framework. We include a discussion on how to slice on the destination region dimension and on the optional contexts dimension.

Since we have N source regions, the slicing yields N slices. Each slice has $N \cdot Q$ entries with a column for each of the N destination regions and a row for each of the Q optional context instances.

Figure 6 shows the result of slicing the tensor in Fig. 4b. Each slice corresponds to a source region and has $6 \cdot 8$ entries, where the columns represent the 6 destination regions and the rows represent the 8 optional context instances. For example, the first slice corresponds to source region R_1. The 6 entries in the bottom row represent the following contexts: $\langle R_1, R_1, q_1\rangle$, $\langle R_1, R_2, q_1\rangle$, ..., $\langle R_1, R_6, q_1\rangle$. Among the 6 entries, only the first is a known entry, labeled with routing preference V_1; the 5 other entries are unknown entries and without labels. Similarly, the slice of R_6 contains contexts whose source region is R_6, where the upper left entry $\langle R_6, R_1, q_8\rangle$ has the learned preference vector V_1 (cf. Sect. 4), which can be transferred to other unknown entries.

5.2.2 Similarity matrix M

We define a *task* as the process of transferring routing preferences from known entries to unknown entries inside a slice, i.e., for a specific source region. Since we have N source regions, we have N tasks. In our example, we have 6 slices and thus 6 tasks.

For each task, we build an entry graph, which has $N \cdot Q$ nodes. In our example, the entry graph for a task has $6 \cdot 8 = 48$ nodes.

We use similarity matrix $\mathbf{M} \in \mathbb{R}^{(N \cdot Q) \times (N \cdot Q)}$ to represent the similarities of all node pairs in the entry graph, which is much smaller than the entry graph used in the basic framework, as shown in Table 1.

Next, we organize similarity matrix $\mathbf{M}$ into $Q \times Q$ blocks, where each block is of size $N \times N$, as shown in Fig. 7.

Formally, we define $\mathbf{M} = \begin{pmatrix} \mathbf{M}_{11} & \cdots & \mathbf{M}_{1Q} \\ \mathbf{M}_{21} & \cdots & \mathbf{M}_{2Q} \\ \cdots & \cdots & \cdots \\ \mathbf{M}_{Q1} & \cdots & \mathbf{M}_{QQ} \end{pmatrix}$, where $\mathbf{M}_{ij} \in \mathbb{R}^{N \times N}$, $1 \le i, j \le Q$. Matrix $\mathbf{M}_{ij}$ captures the spatial adjacency among N destination regions, where each element $m_{xy} \in \mathbf{M}_{ij}$, $1 \le x, y \le N$, captures the spatial adjacency between regions R_x and R_y. If R_x and R_y are adjacent, element m_{xy} is set to 1. If R_x and R_y are neighbors within δ hops, we exponentially decay the weight: $m_{ij} = e^{-\lambda \times dist}$, where *dist* is the Euclidean distance of the centroids of the two regions [28]. If R_x and R_y are more than δ hops apart, m_{ij} is set to 0.

All $\mathbf{M}_{ij}$, $1 \le i, j \le Q$, are identical since they all capture the spatial adjacency among destination regions. In addition, the similarity matrices for different tasks are also identical. This means that we need maintain only one similarity matrix for all N tasks, which only requires $\Theta(N^2)$ space. This addresses the second challenge of having a very large adjacency matrix in the basic framework.

Next, the similarity function takes only spatial contexts, i.e., destination regions, into account, which simplifies the design of similarity functions and thus addresses the first challenge in the basic framework. However, the temporal and other contexts are not simply ignored, we push them into a

feature matrix that is used as input to the learning algorithm along with the similarity matrix.

Alternatively, we could consider organizing similarity matrix $\mathbf{M}$ into $N \times N$ blocks, each of which is of size $Q \times Q$ and captures the similarities between the optional contexts. However, then the non-trivial challenge of defining an appropriate similarity function that considers more than one context, i.e., temporal and other contexts, remains, meaning that this design leaves the first challenge in the basic framework unaddressed.

5.2.3 Feature matrix X

Similarity matrix $\mathbf{M}$ of a task only considers spatial relationships among entries, and it ignores the optional context features of the nodes, e.g., different departure time intervals and weather conditions. To remedy this shortcoming, we introduce a feature matrix that is fed into the multi-task learning framework together with $\mathbf{M}$.

We define this feature matrix as $\mathbf{X} \in \mathbb{R}^{(N \cdot Q) \times F}$, where $N \cdot Q$ is the number of nodes in the entry graph of a task and F denotes the number of features that we consider for a node.

We choose a set of F features to describe every node including optional contexts and additional features derived from destination regions. Equation 3 defines an F dimensional feature vector x_i, $i \in [1, n]$, with $n = N \cdot Q$, for the ith node.

$$x_i = [x_i^{(1)}, \ldots, x_i^{(M)}, \ldots, x_i^{(F)}], \tag{3}$$

The feature matrix is defined as $\mathbf{X} = [x_1, x_2, \cdots, x_n]^T$.

First, we consider all the optional contexts as features. Recall that in Sect. 3.1, we introduced $\mathbb{S} = \{S_1, S_2, \ldots, S_M\}$ as a set of M optional contexts. We include these M contexts into the feature matrix, i.e., the first M elements $x_i^{(1)}, \ldots, x_i^{(M)}$ in Eq. 3. In the example in Fig. 4, we have $M = 3$ optional contexts, i.e., departure time, driver, and weather conditions.

Second, we consider additional features derived from destination regions. For example, we may consider the road type distribution of the roads in the destination region, which has been shown to be effective at differentiating regions with different functionalities, e.g., residential versus industrial regions [16]. If we consider 3 road types, e.g., highways, residential roads, and tertiary roads, we have 3 additional features. This yields a total of $F = 6$ features for a node.

Consider the node $\langle R_6, R_1, q_8 \rangle$ in the slice for source region R_6. First, we consider the optional contexts represented in $q_8 = \langle t_2, d_2, w_2 \rangle$. Second, assume that 30%, 50%, and 20% of the roads in R_1 are highways, residential roads, and tertiary roads, respectively. We then construct the following 6-dimensional feature vector for this node: $\langle t_2, d_2, w_2, 0.3, 0.5, 0.2 \rangle$.

As for the similarity matrix, different tasks use the same feature matrix $\mathbf{X}$ because they have nodes that represent the same set of destination regions and optional contexts.

5.2.4 Slicing on other dimensions

Slicing on the destination region dimension works identically to slicing on the source region dimension. We are able to construct similarity matrix $\mathbf{M}$ and feature matrix $\mathbf{X}$ using the same methodology by just replacing source regions with destination regions. However, since source regions are identical to the destination regions, slicing on the destination region dimension and slicing on the source region dimension yield the same result.

Slicing on the optional contexts dimension is slightly different from slicing on the source region dimension. However, we are still able to follow the same methodology to construct similarity matrix $\mathbf{M}$, where we can choose either source regions or destination regions. Then, we organize similarity matrix $\mathbf{M}$ into $N \times N$ blocks, where each block is also of size $N \times N$. Each block can be computed in the same way as when slicing the source region dimension. Next, feature matrix $\mathbf{X}$ can be constructed by using the road type distributions of both source and destination regions.

5.2.5 Transferring routing preferences

Figure 8 illustrates the final multi-task learning model architecture for transferring routing preferences using N tasks. Each task transfers routing preferences within its own slice. A shared layer is employed to model the relationships among different tasks.

Learning in a single task Each task takes three matrices as input—similarity matrix $\mathbf{M}$, feature matrix $\mathbf{X}$, and label matrix $\mathbf{Y}_i$. All tasks use the same similarity matrix $\mathbf{M}$ and feature matrix $\mathbf{X}$. However, different tasks use different label matrices. In particular, label matrix $\mathbf{Y}_i \in \mathbb{R}^{(N \cdot Q) \times K}$ records the labels of the nodes for the ith task. Thus, K is the total number of labels, i.e., the number of distinct routing preferences from known entries. If the jth node corresponds a known entry and its label is the kth label then the (j, k)th element is set to 1 and all remaining elements in the jth row are set to 0. If the jth node corresponds an unknown entry, all the elements in the jth row are set to 0.

Each task solves a classification problem—it assigns a label to each unknown entry. Deep learning has demonstrated superior performance at solving classification problems. In particular, graph convolutional networks (GCNs) are the state-of-the-art deep learning classification algorithms for graph data. Thus, we choose use a GCN to solve each individual task.

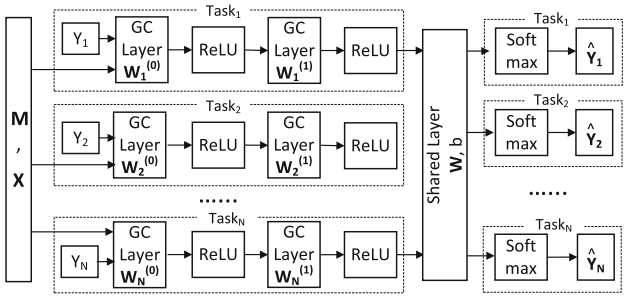


Fig. 8 Multi-task GCN model architecture

In GCNs, the input matrix, i.e., feature matrix $\mathbf{X}$, is convoluted with multiple graph convolutional filters to obtain multiple feature maps in a graph convolutional (GC) layer. Then, the feature maps are combined and fed into activation functions to obtain outputs that can be compared with ground truth labels. Here, a graph convolutional filter utilizes the similarity matrix that models the similarities of node pairs to realize the intuition that similar nodes should have similar labels. We can apply such "GC layer + activation functions" multiple times.

We use Cheby-Net [24] as the graph convolutional filter, due to its high accuracy and efficiency. Specifically, we transform the input feature matrix $\mathbf{X} \in \mathbb{R}^{(N \cdot Q) \times F}$ into F encoded feature matrices $\mathbf{Z}_f \in \mathbb{R}^{(N \cdot Q) \times H}$, $f \in [1, F]$. Each $\mathbf{Z}_f = [z_1, z_2, \ldots, z_H]$ has H vectors $z_h \in \mathbb{R}^{(N \cdot Q)}$, $h \in [1, H]$, that are defined recursively: $z_1 = \mathbf{X}_{:,f}$, $z_2 = \tilde{\mathbf{L}} \times z_1$, and $z_h = 2\tilde{\mathbf{L}} \times z_{h-1} - z_{h-2}$, when $h > 2$. Here, $\tilde{\mathbf{L}}$ is the normalized Laplacian matrix of similarity matrix $\mathbf{M}$. We first compute the degree matrix $\mathbf{D} = diag(\sum_j M_{ij})$, followed by a Laplacian operation $\mathbf{L} = \mathbf{D} - \mathbf{M}$, and normalize it as $\tilde{\mathbf{L}} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{L}\mathbf{D}^{-\frac{1}{2}}$, where $\mathbf{I}$ is an identity matrix.

Then, we apply μ filters to proceed the graph convolution. For the uth filter W_u, $u \in [1, \mu]$, we have

$$\mathbf{X} \star W_u = \sum_{f=1}^{F} (\mathbf{Z}_f \otimes W_u + b_u), \tag{4}$$

where $\star$ denotes the graph convolution operator, $\otimes$ denotes matrix multiplication, $W_u \in \mathbb{R}^{H \times 1}$, and $b_u \in \mathbb{R}^{(N \cdot Q) \times 1}$. The output of the graph convolution using the uth filter is $\mathbf{X} \star W_u \in \mathbb{R}^{(N \cdot Q) \times 1}$. Given all μ filters $\mathbf{W} = [W_1, \ldots, W_\mu]$, we concatenate the outputs of the μ filters as the final output. Specifically,

$$\mathbf{X} \star \mathbf{W} = [\mathbf{X} \star W_1, \ldots, \mathbf{X} \star W_\mu]. \tag{5}$$

We construct two GC layers for each task and present each layer using Eq. 5. The output from the two GC layers for the ith task, which is also the input to the shared layer, is $g_i \in \mathbb{R}^{(N \cdot Q) \times \mu'}$ as defined in Eq. 6, where μ' is the number of graph filters in the second GC layer.

$$g_i = ReLU(ReLU(\mathbf{X} \star \mathbf{W}_i^{(0)}) \star \mathbf{W}_i^{(1)}), \tag{6}$$

where $ReLU = \max(0, \cdot)$ is a nonlinear activation function, and $\mathbf{W}_i^{(0)}$ and $\mathbf{W}_i^{(1)}$ are the graph convolutional filters (cf. Eq. 5) of the two GC layers for the ith task.

Shared layer Recall that multi-task learning is designed to exploit the intrinsic relations among multiple tasks to improve the overall performance of all tasks. In our setting, we would like to exploit the relationships among tasks for different source regions. Here, we do not explicitly model the relationships among different tasks. Rather, we let the model itself learn the relationships using a fully connected, shared layer based on the labeled data.

The shared layer utilizes an weight matrix $\mathbf{W}$ to capture the relationships among different tasks. More specifically, each task's output g_i is multiplied with the shared weight matrix $\mathbf{W}$ and then is fed into an activation function to produce estimated labels for the task. The output of the ith task, i.e., the estimated labels for nodes in the ith partition, is defined in Eq. 7.

$$\hat{\mathbf{Y}}_i = softmax(g_i \otimes \mathbf{W} + b), \tag{7}$$

where $\mathbf{W} \in \mathbb{R}^{\mu' \times K}$ is the shared weight matrix in the shared layer, $b \in \mathbb{R}^{(N \cdot Q) \times 1}$ is a bias vector, and *softmax* is the activation function used for conducting classification.

Objective function The objective function is defined to evaluate the cross-entropy error between the estimated labels $\hat{\mathbf{Y}}_i$ and the ground truth labels $\mathbf{Y}_i$ over all tasks:

$$\mathcal{L} = -\sum_{i=1}^{N} \sum_{j=1}^{N \cdot Q} \sum_{k=1}^{K} y_{ij}^{f} \ln \hat{y}_{ij}^{k}, \tag{8}$$

where N is the total number of tasks, i.e., source regions; $N \cdot Q$ is the total number of nodes in a task; and K is the total number of labels, i.e., the number of distinct routing preferences from known entries. Further, y_{ij}^{k} and $\hat{y}_{ij}^{k}$ are the ground truth label and estimated label for the kth label of the ith source region on the jth node, respectively.

By minimizing the objective function, we obtain an estimated label matrix $\hat{\mathbf{Y}}_i$ for each task that contains an estimated label for each unknown entry. This gives us a full tensor.

The complexity of the multi-task learning framework is $O((NQ)^2 \cdot H + \mu \cdot NQ \cdot H) \cdot F \cdot N + N^2 \cdot Q \cdot \mu \cdot K)$.

5.3 Enabling CPR

After transferring routing preferences, we obtain a full contextual preference tensor. Given a *CPR* query $CPR(v_s, v_d, OCI)$, we perform a lookup in the tensor using v_s, v_d, and

OCI to obtain a routing preference vector V. Then, we perform a preference-based routing using the proposed PCH by calling function *PrefRouting*(V, v_s, v_d) (cf. Algorithm 5).

6 Empirical studies

We conduct an in-depth empirical study using two substantial GPS trajectory data sets from two different cities with different sampling rates.

6.1 Experimental setup

Road networks and GPS trajectories We obtain two road networks, N_1 and N_2, from OpenStreetMap[1]. Network N_1 represents the road network of Aalborg, Denmark, which includes 33,226 vertices and 78,348 edges. Network N_2 represents the road network of Chengdu, China, which includes 27,671 vertices and 77,444 edges. We build preference-based contraction hierarchies (PCH) (cf. Sect. 4.3.1) on both networks, which include 101,530 and 228,178 shortcuts for Aalborg and Chengdu networks, respectively.

We use two GPS data sets D_1 and D_2 that relate to N_1 and N_2, respectively. D_1 consists of more than 180 million high-frequency GPS records collected from private passenger vehicles at 1 Hz (i.e., one GPS record per second) in 2007 and 2008. D_2 consists of 1 billion low-frequency GPS records collected from taxis from August 3rd to 30th 2014. The sampling rate varies from 0.03 Hz to 0.1 Hz. We only use the parts of the trajectories where the taxis carry passengers on. Outlier detection algorithms [29,30] can be applied to filter outlier data. We map match [13] the GPS records in D_1 and D_2 onto N_1 and N_2, respectively, obtaining 239,146 and 1,649,299 trajectories.

We partition the trajectories into two. *Part1*, containing 90% of the original trajectories, is used for learning and transferring routing preferences to obtain a full contextual preference tensor. *Part2*, containing 10% of the original trajectories, is used for evaluating the *CPR* queries based on the obtained tensor. More specifically, *Part1* is used in the experiments covered in Sects. 6.2 and 6.3, and *Part2* is used in the experiments covered in Sect. 6.4.

Contextual preference tensor For each road network, we create a contextual preference tensor $\mathbf{T} \in \mathbb{R}^{N \times N \times Q}$. We run experiments when considering only departure time (denoted as *Time*), only driver identities (denoted as *Driver*), and their combinations (denoted as $T + D$) as optional contexts, respectively. Further, we use an $X \times X$ grid to partition the road network into equal sized regions. We vary X differently for the three types of optional contexts in order to study the efficiency, effectiveness, and scalability of our approach in different settings.

We apply "SELECT-FROM-WHERE" functionality to retrieve trajectories under specific contexts in a database. In particular, the "WHERE" clause specifies the contexts, e.g., start and end regions, departure time, and drivers.

For *Time*, we set $Q = 4$, which corresponds to the morning and afternoon peak periods and the off-peak periods in-between. We vary X across 10, 15, 20, 25, 30, and 35, meaning that N varies across 10^2, 15^2, 20^2, 25^2, 30^2, and 35^2, and all trajectories in *Part1* are used.

For *Driver*, we consider the $Q = 4$ drivers who have the most trajectories in data sets D_1 and D_2. We vary X across 5, 6, 7, 8, 9, and 10, and thus vary N across 5^2, 6^2, 7^2, 8^2, 9^2, and 10^2, and we only use trajectories that belong to the 4 drivers in *Part1*.

For $T + D$, we combine the 4 time periods with the 4 chosen drivers, which yields $Q = 4 \times 4 = 16$ optional contexts. We vary N across 5^2, 6^2, 7^2, 8^2, 9^2, and 10^2. We only conduct this experiment on $T + D$ using data set D_1, since the number of trajectories generated for each driver in data set D_2 is insufficient to produce reasonable transferred results when we consider time periods. Dealing with insufficient training data is beyond the scope of the paper and is considered as future work.

To observe the sparseness of the contextual preference tensor, we report the average label coverage (*ALC*) for different N. Specifically, we define the label coverage for a task, i.e., a source region, as the percentage of contexts with trajectories, i.e., known entries, divided by the total number of contexts in the task, i.e., $N \cdot Q$. The *ALC* is the average label coverage over all tasks, defined as follows.

$$ALC = \frac{1}{N} \cdot \sum_{i=1}^{N} \frac{\#known\ entries\ in\ the\ i\text{-}th\ task}{N \cdot Q}$$

Table 2 shows the *ALC* for different optional contexts. As can be seen, the trajectories are unable to cover all contexts in any setting. As we partition a road network into more regions, we get more contexts, and the average label coverage decreases.

Table 2 Statistics of average label coverage

Context	N	10^2	15^2	20^2	25^2	30^2	35^2
Time	D_1	0.453	0.322	0.238	0.233	0.212	0.201
	D_2	0.478	0.348	0.307	0.278	0.271	0.251
Context	N	5^2	6^2	7^2	8^2	9^2	10^2
Driver	D_1	0.426	0.374	0.279	0.258	0.24	0.237
	D_2	0.236	0.164	0.151	0.139	0.123	0.118
$T + D$	D_1	0.315	0.289	0.219	0.182	0.173	0.147

[1] openstreetmap.org.

Routing preferences For the master dimension, we consider distance (DI), travel time (TT), and fuel consumption (FC) as the travel cost features. Specifically, distances are obtained from OpenStreetMap, travel times are computed based on GPS trajectories and speed limits, and fuel consumption is computed from speeds using vehicular environmental impact models [31,32].

We consider two slave dimensions. The first represents road types, where we use six road types from OpenStreetMap: motorway, trunk, primary, secondary, tertiary, and residential. The second captures whether or not a road is toll free.

Implementation details The algorithms for learning routing preferences, including the preference-based contraction hierarchies, are implemented in Java using JDK 1.8, and the related experiments are conducted on a server with a 64-core AMD Opteron 2.24 GHZ CPU, 528 GB main memory under Ubuntu Linux. The algorithm for transferring routing preferences is implemented in Python 3.5 and run with tensorflow on an Abacus 2.0 (escience.sdu.dk) server with two Intel E5-2680v3 CPUs, each with 12 CPU cores and 64 GB RAM and with two NVIDIA K40 GPU cards, each with 2880 CUDA cores and 12 GB RAM.

6.2 Learning routing preferences

In this set of experiments, we study the accuracy and efficiency of the algorithms for learning routing preferences.

For each entry in tensor **T**, we first obtain a set of corresponding trajectories. We only consider the known entries whose trajectory sets are non-empty. For each known entry, we learn a routing preference using the learning algorithms covered in Sect. 4.2, observing the accuracy and efficiency of the algorithms.

Algorithms We consider six routing preference learning algorithms. In particular, we consider two shortest path finding strategies: Dijkstra's algorithm that takes into account preferences (*PDI*) and preference-based contraction hierarchies (*PCH*) (see Algorithm 5). In addition, we consider three different search strategies: exhaustive search (*ex*), coordinate descent search (*co*), and stochastic coordinate descent search (*st*) (see Sect. 4.2). This yields 6 combinations and thus 6 different algorithms: *PDI-ex*, *PDI-co*, *PDI-st*, *PCH-ex*, *PCH-co*, and *PCH-st*. The proposed Algorithm 5 uses *PCH* and *st*, and is thus denoted as *PCH-st* in the following.

Accuracy It is non-trivial to measure the accuracy of the learned routing preferences (RPs), since we do not have ground truth RPs. Instead, we use the following strategy. First, given a set of trajectories $\mathbb{TR}$ for a known entry, we learn an RP vector V. Second, for each trajectory $T_i \in \mathbb{TR}$, we obtain its source s_i, destination d_i, and its path P_i. Then, we apply preference-based routing using s_i, d_i, and V to obtain a path P_i^V. Third, if the learned RP vector V reflects accurately the actual RP, the path P_i used by the trajectory and path P_i^V should be similar. However, it is often unlikely that two paths are identical because the routing preference space does not cover all possible features. Drivers may consider many other features not covered by the preference space. Based on the above strategy, we define and measure the accuracy of the learned RP for a known entry as follows.

$$Acc = \frac{1}{|\mathbb{TR}|} \sum_{i=1}^{|\mathbb{TR}|} Sim(P_i, P_i^V), \tag{9}$$

where $Sim(\cdot, \cdot)$ measures the similarity between two paths, as defined in Eq. 1.

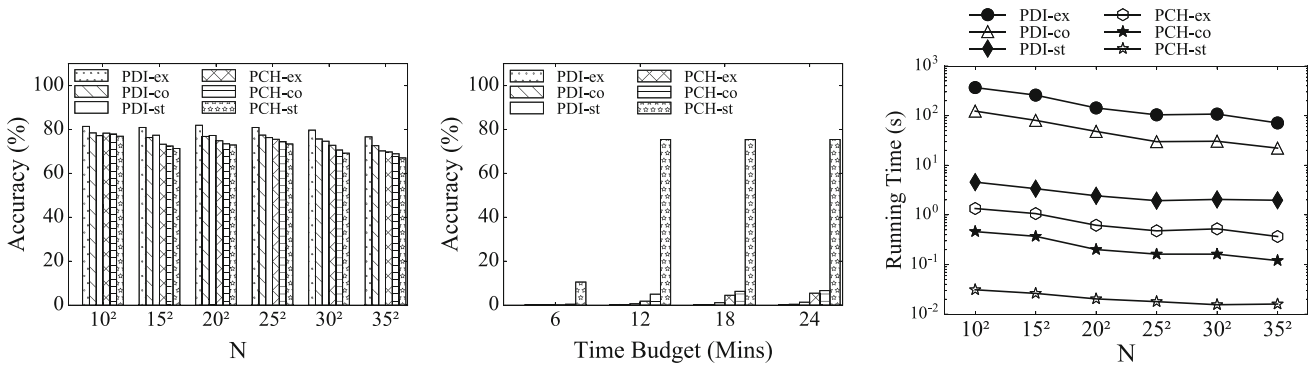
We use optional context *Time* as an example in this experiment, since results for other contexts follow the same trend, and we report two types of average accuracy over known entries to observe the effectiveness of different routing preference learning algorithms.

The first type of average accuracy is reported over the known entries that all 6 preference learning algorithms are able to generate RPs for within a week. *PDI-ex* is the slowest algorithm; it is able to learn RPs for 22 and 27 known entries for D_1 and D_2, respectively, when $N = 10^2$. The average accuracy is reported over these entries, although the most efficient algorithm *PCH-st* is able to learn RPs for all entries using approximately 12 and 90 min for D_1 and D_2, respectively.

The second type of average accuracy is reported over all known entries within a runtime budget, and we use $N = 10^2$. For D_1, we set the budget to 6, 12, 18, and 24 min; for D_2, we set the budget to 30, 60, 90, and 120 min, since D_2 contains many more trajectories than does D_1. If an algorithm is able to learn an RP for an entry within a time budget, we calculate *Acc* for the entry; otherwise, we set *Acc* as 0 for the entry.

The two types of average accuracy for D_1 for the different algorithms are shown in Fig. 9a, b. Data set D_2 yields similar results and is not shown.

When all algorithms are able to learn a preference for an entry, the average accuracy of *ex*, *co*, and *st* decreases gradually, which is reasonable because the levels of their search approximation increase. *PDI* generally outperforms *PCH*, i.e., Algorithm 5, since the features, e.g., road types, of shortcuts are only an approximation of the features of the edges in the shortcuts when creating contraction hierarchies. However, using *st* and *PCH* only slightly reduces the accuracy. Consider a concrete example using Fig. 5. Assume that we consider the case where the source is x, the destination is y, and the road type preference is "Residential." We first assume that we have no *PCH*. Then, $\langle x, z, y \rangle$ is returned with a cost of 9 and using only Residential roads. We then assume that we have a *PCH*. Here, the shortcut $\langle x, y \rangle$ has slave features $\{Residential, Tertiary\}$, which includes "Residential," so



(a) Accuracy, Type 1 **(b)** Accuracy, Type 2 **(c)** Off-line Learning Time Per Entry

Fig. 9 Evaluation of preference learning, D_1, *time* as optional context

we use the shortcut. This yields a path $\langle x, u, y\rangle$ with cost 8 using both residential and tertiary roads. Thus, using the *PCH* produces an inaccurate result. However, if the road types of both edges xz and zy are tertiary, no matter whether we use the *PCH*, we always obtain $\langle x, u, y\rangle$. In this case, the *PCH* is accurate.

When we control the time budget, the average accuracy of each algorithm, except *PCH-st*, is poor because all these other algorithms take too long time and can only identify RPs for very few entries.

Efficiency To evaluate efficiency, we record the average off-line learning time for obtaining an RP for a known entry. The learning time is reported in Fig. 9c for data set D_1 (results in D_2 again show a similar trend). The learning time required by *PCH-st* is significantly lower than those for the other algorithms. This also justifies the finding for the second type of accuracy. Further, the learning time for a method decreases as we increase N. This is because the learning time is influenced by the number of trajectories available for a know entry. When given the same set of trajectories, if we consider more regions and thus more contexts, each context gets fewer trajectories, which reduces the learning time per entry.

Considering the above findings, we focus on *PCH-st* in the sequel because it requires the least off-line running time, e.g., only 12 min for D_1, and achieves reasonable accuracy when compared to the other methods. Although the other methods are able to produce slightly more accurate RPs, it is impractical to wait for days or weeks to get small improvements.

6.3 Transfer of routing preferences

Training, validation, and testing data For each task, i.e., each source region, we have a set of known entries (see Table 2 for statistics), whose RPs, i.e., labels, are learned from available trajectories. We aim at transferring these labels to unknown entries. Although the aim is to transfer labels to the unknown entries, we cannot evaluate the accuracy of the transferred labels for the unknown entries since we do not have ground truth labels for them. Instead, we partition the known entries into two sets, each of which has 50% of the known entries. We use the labels in the first set as the training and validation data, and use those in the second set as testing data. We apply fivefold cross-validation.

To give a concrete example, assume that we have 150 entries of which 100 are known and 50 are unknown. Although our algorithm is able to estimate labels for the 50 unknown entries, we cannot evaluate the accuracy of the 50 new labels. Instead, we use the 100 known entries to evaluate the accuracy of estimated labels. In particular, from the 100 known entries, we "hide" 50 entries from the transfer algorithm. The transfer algorithm utilizes the remaining 50 known entries where 40 are used to build the model and 10 are used for validating the model via fivefold cross-validation. Then, the algorithm estimates labels for the remaining 50 hidden known entries. For these 50 estimated labels, we can compare with the known ground truth labels. This evaluation methodology is also used in existing studies [28].

Evaluation criteria For each testing entry, if a predicted label is the same as the ground truth label, we set the accuracy to 1; otherwise, we set it to 0. We report the average accuracy over all testing entries over all tasks in the testing data.

Baselines We compare the advanced framework (cf. Sect. 5.2), i.e., multi-task learning with GCN (denoted as MTL-GCN) with two baselines: (1) The basic framework with Label Propagation (denoted as LP): LP (github.com/parthatalukdar/junto) is a representative of graph-based transduction learning, and it does not use multi-task learning. (2) The basic framework with GCN (N-GCN) (cf. Sect. 5.1): a graph convolutional network without applying the multi-task learning approach (github.com/tkipf/gcn), which is the state of the art. Due to the space limitation, details on these are omitted.

Hyperparameters For LP and N-GCN, we apply a threshold to the similarity pairs for LP and to the adjacency matrix for N-GCN. We do this in order to achieve higher transfer accu-

Table 3 HyperParameters, *Time* as optional context

HyperParameters	D_1		D_2	
	MTL-GCN	N-GCN	MTL-GCN	N-GCN
Dropout rate	0.1	0.3	0.1	0.7
Learning rate	0.001	0.1	0.001	0.001
Weight decay	$5 \cdot 10^{-4}$	$5 \cdot 10^{-4}$	$5 \cdot 10^{-4}$	$5 \cdot 10^{-4}$
μ/μ'	16/8	64	16/8	64
Dist λ	0.3	0.3	0.3	0.3

Table 4 Accuracy of transferred preferences

N	*Time*, D_1			*Time*, D_2		
	MTL-GCN	LP	N-GCN	MTL-GCN	LP	N-GCN
10^2	0.683	0.52	0.645	0.709	0.651	0.701
15^2	0.678	0.488	0.562	0.721	0.638	0.690
20^2	0.705	0.456	0.561	0.731	0.591	0.669
25^2	0.724	0.449	0.517	0.732	0.49	0.673
30^2	0.723	–	–	0.729	–	–
35^2	0.719	–	–	0.721	–	–

N	*Driver*, D_1			*Driver*, D_2	$T + D$, D_1
	MTL-GCN	LP	N-GCN	MTL-GCN	MTL-GCN
5^2	0.741	0.534	0.575	0.681	0.723
6^2	0.732	0.483	0.638	0.705	0.733
7^2	0.773	0.457	0.627	0.743	0.724
8^2	0.696	0.454	0.605	0.686	0.729
9^2	0.714	0.483	0.603	0.604	0.741
10^2	0.696	0.446	0.626	0.582	0.738

racy and to ensure that the similarity pairs and the adjacency matrix can be loaded into main memory.

We do not run experiments with LP and N-GCN for some settings of N (see Table 4) because: (1) when N increases, generating the similarity pairs or the adjacency matrix is not scalable because the time complexity is $O((N \cdot N \cdot Q)^2)$; and (2) the experimental results exhibit decreasing accuracy as we increase N. In contrast, the time complexity for generating an adjacency matrix in MTL-GCN with $O((N \cdot Q)^2)$.

We apply random search [33] to get an optimal set of hyperparameters for both N-GCN and MTL-GCN. We report these hyperparameters for optional contexts *Time* in Table 3 as an example, and the results are not sensitive to changes in the hyperparameters for the other optional contexts.

Accuracy The results of transferring RPs for both data sets for three types of optional contexts are shown in Table 4. We omit those for LP and N-GCN for data set D_2 with context *Driver* and for data set D_1 with context $T + D$ due to the space limitation and because they are worse than MTL-GCN in all other settings.

The scalability and accuracy of LP and N-GCN suffer as N increases. In contrast, MTL-GCN uses multiple tasks, where each task has a much smaller entry graph, making it more scalable w.r.t. N. Further, since the MTL-GCN uses the shared layer, which is able to learn mutual relationships among tasks, the accuracy of MTL-GCN does not decrease as we increase N. This offers evidence that the proposed advanced framework is more scalable and accurate than the state of the art.

Table 5 Effect of varying transfer training size

Train %, Test %	50, 50	60, 40	70, 30	80, 20	90, 10
Transfer Accuracy	0.683	0.698	0.743	0.786	0.803

Varying training sizes We vary the training data size when transferring routing preferences from some known entries to some other known entries, i.e., entries serving as "hidden" testing entries. We vary the size of the set of training and validation entries from 50% to 90% of all known entries, and we use 50% to 10% known entries as testing entries accordingly. See Table 5 that reports the transfer accuracy and shows an increasing trend when more training entries are available.

6.4 Context-aware preference-based routing

Implementation details For on-line routing, a user must provide the spatial contexts, i.e., source vertex v_s and destination vertex v_d. The temporal and other contexts are specified in *OCI* as optional contextual information, e.g., departure time and driver identifier.

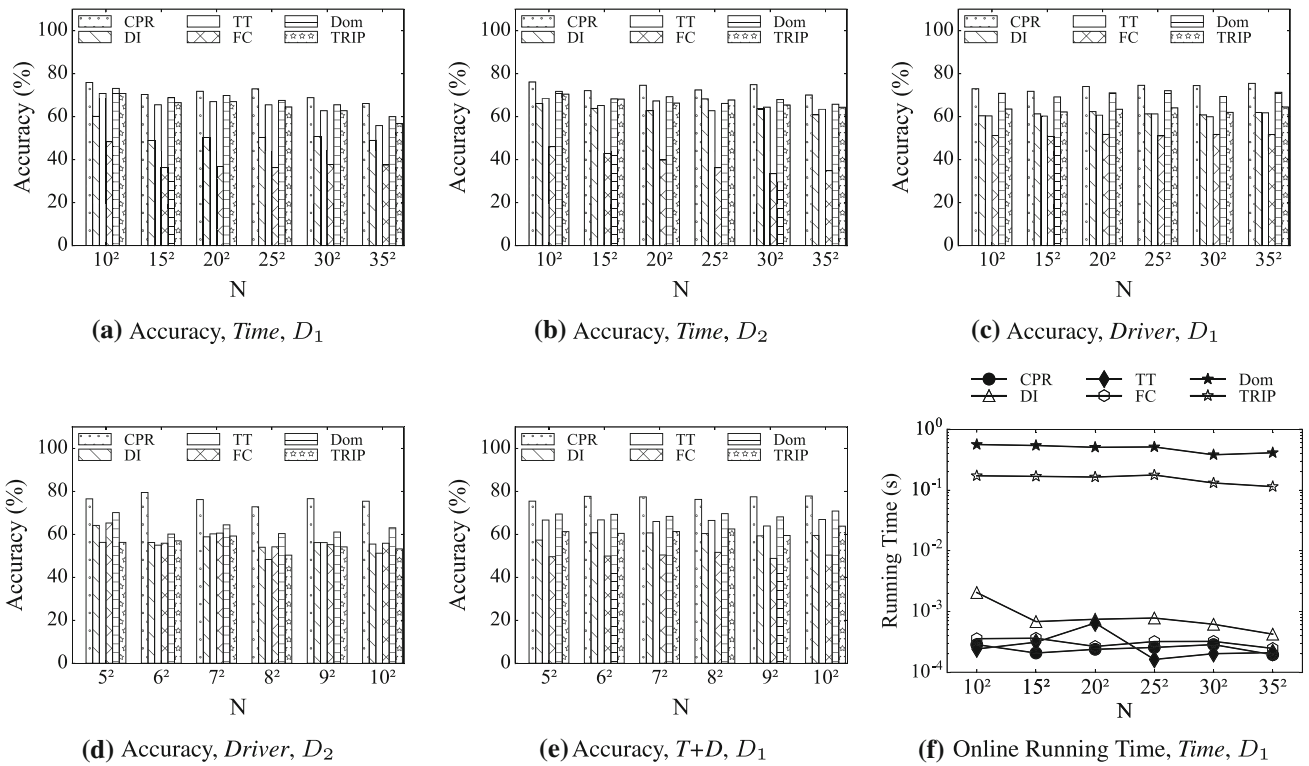
As long as the spatial contexts, i.e., source vertex v_s and destination vertex v_d, are available, preference-based routing can be conducted even if *OCI* is missing. We enable this by randomly picking a routing preference from the full tensor whose source and destination regions contain vertices v_s and v_d, respectively.

Evaluation strategy For each entry in the contextual preference tensor, we are able to obtain an RP vector V since the tensor is full. Recall that we use a second, separate partition of trajectories, i.e., *Part2*, to evaluate the *CPR* queries in this set of experiments. We obtain a set of trajectories $\mathbb{TR}$ for each entry from the second partition. If no trajectories exist, we disregard the entry in these experiments. For each trajectory $T_i \in \mathbb{TR}$, we record its source s_i, destination d_i, and actual path P_i.

We use different routing algorithms to identify paths from s_i and d_i and compare the similarity between the identified paths and P_i, which is treated as the ground truth. If a routing algorithm returns a path that is very similar to P_i, this increases the quality of the algorithm. We report the average quality over all known entries. In particular, the accuracy of a routing algorithm is defined as $Acc = \frac{1}{|\mathbb{TR}|}\sum_{i=1}^{|\mathbb{TR}|} Sim(P_i, P_i')$, where P_i' is a path returned by the routing algorithm and $Sim(\cdot,\cdot)$ is defined in Eq. 1.

Baselines We compare our routing algorithm (CPR) with five routing methods, denoted as *DI*, *TT*, *FC*, *Dom*, and *Trip*. *DI*, *TT*, and *FC* are shortest path finding algorithms using classic contraction hierarchies to return paths that have the smallest distance (*DI*), fastest travel time (*TT*), and least fuel consumption (*FC*). We initialize the time-dependent travel time-based weights [11] to run *TT*, where a day is divided into 96 15-min windows.

Dom [6] and *TRIP* [17] are two personalized routing algorithms that are able to find personalized "shortest" paths. In particular, *Dom* and *TRIP* first learn an RP for each driver (while disregarding other contexts such as spatial contexts) from the driver's historical trajectories. *Dom* utilizes an RP that considers distance, travel time, and fuel consumption, whereas *TRIP* uses an RP that considers only travel time. *Dom* first computes a set of skyline paths and then choose one best skyline path according to the learned preference. *TRIP* uses the learned preference to obtain new, personalized weights for all edges and finally apply shortest path finding using the new edge weights. We apply both algorithms to learn an RP according to a driver's trajectories in the first partition of the trajectories. For each trajectory in the second partition of the trajectories, we obtain the source, destination,



(a) Accuracy, *Time*, D_1 **(b)** Accuracy, *Time*, D_2 **(c)** Accuracy, *Driver*, D_1

(d) Accuracy, *Driver*, D_2 **(e)** Accuracy, *T+D*, D_1 **(f)** Online Running Time, *Time*, D_1

Fig. 10 Evaluation of the overall preference learning and routing

and driver identifier. Then, we apply *Dom* and *TRIP* to compute the personalized, shortest path connecting the source and the destination according to the driver identifier.

Accuracy The accuracy of these methods is reported for data sets D_1 and D_2 using the three optional contexts in Fig. 10a, e. *CPR* outperforms the other methods on both data sets given different optional contexts. *Dom*, which considers the trade-off among distance, travel time, and fuel consumption for individual drivers, is second best. However, *Dom* needs significantly more running time than *CPR* (see Fig. 10f) because it conducts multi-objective skyline routing between source and destination which is expensive. *TRIP* and *TT* have slightly lower accuracies than *Dom*; *TRIP*, which considers personalized travel times learned for individual drivers, performs slightly better than *TT*. *TT* considers only travel times and returns the fastest paths. However, the ground truth is a path that fits the preferences of local drivers. Thus, it may be that the fastest paths do not align with the preferences of local drivers. *FC* is worst, which indicates that merely considering fuel consumption is not competitive; other factors, e.g., travel time, should be considered as well.

The results indicate that *CPR* is versatile: when either departure time or driver identifiers is used as an optional context, *CPR* offers either time-dependent or personalized preference-based routing; when departure time and driver identifiers are used together as optional contexts, *CPR* offers time-dependent personalized preference-based routing. In addition, *CPR* considers spatial contexts, while existing algorithms, e.g., *Dom* and *Trip*, ignore spatial contexts.

On-line running time The on-line running time of all methods is reported for data set D_1 in Fig. 10f; the result for D_2 is similar and is omitted. The proposed *CPR*, *DI*, *TT*, and *GE* run at the same magnitude since they are implemented based on contraction hierarchies. *TRIP* is slower since it is implemented based on Dijkstra's algorithm. *Dom* runs much slower than other methods since it requires significantly more time to conduct multi-criteria skyline routing and because speed-up techniques such as contraction hierarchies cannot be applied directly to support skyline routing.

Varying training sizes To test the effect of the training data set size on preference-based routing, we conduct experiments where we choose at random 50%, 60%, 70%, 80%, and 90% of the original trajectories to form *Part 1* (cf. Sect. 6.1).

The training data set size mainly influences: (1) the number of trajectories between each pair of grids and (2) the number of grid pairs connected by trajectories.

The accuracy is not influenced much because we use stochastic coordinate descent to learn a preference from trajectories between a grid pair, meaning that we already do not use all trajectories. Hence, fewer trajectories between each grid pair do not influence accuracy substantially.

The accuracy is affected if fewer pairs of grids are connected by trajectories. When the training data set size is small, the average label coverage is also small, as shown in Table 6, resulting in slight accuracy decrease in preference-based routing.

Table 6 Effect of varying training trajectory sizes

Train size	50%	60%	70%	80%	90%
Label coverage	0.407	0.417	0.433	0.443	0.453
Routing accuracy	74.15	73.91	75.27	76.06	75.89

Table 7 Statistics of road networks

Graph	#Vertices	#Edges	Area (km^2)
AAL	33,226	78,348	1,140
NJ	68,318	163,587	7,933
DK	667,950	1,623,457	42,933

6.5 Scalability

We study the scalability of the proposed methods using graphs of different sizes. In particular, we partition the road network of Denmark into small, medium, and large road networks, as shown in Table 7, where AAL, NJ, and DK represent Aalborg, North Jutland, and Denmark, respectively. We also generate 2,221,866 synthetic trajectories, where a trajectory traverses 589 edges on average.

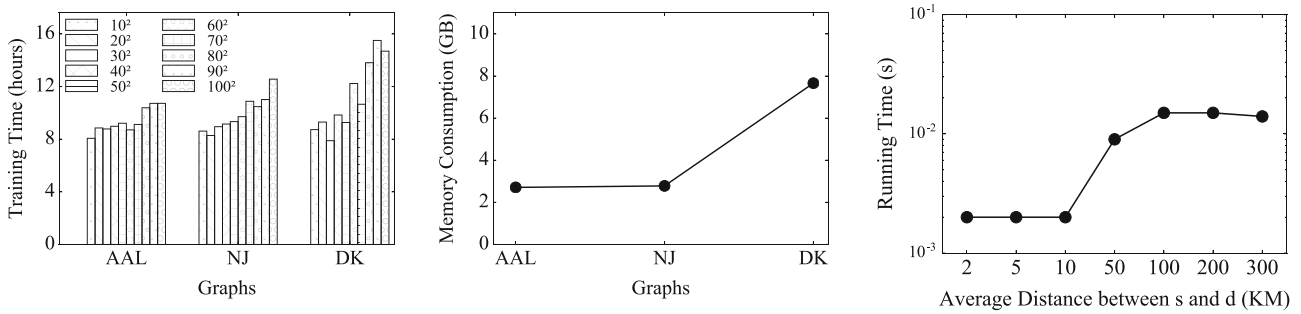
We partition each graph using $X \times X$ grids, where X is varied from 10 to 100 in increments of 10, meaning that N varies from 10^2 to 100^2.

Figure 11a shows the training time for learning preferences in the three graphs with grid sizes ranging from 10^2 to 100^2. The training time also contains the time for reading trajectories from the database, approximately 3 h. Figure 11b shows the memory consumption for building preference-based contraction hierarchies (PCH) (cf. Sect. 4.3.1). Figure 11c reports on the on-line run time of Algorithm 5, ordered by routing distances.

7 Related work

Existing studies on using historical trajectories for routing can be categorized into two categories: *trajectory-based routing*, where historical trajectories are cut and stitched together to form new paths [16,18,34–36], and *preference-based routing*, where routing preferences are learned from historical trajectories and then used to guide routing algorithms to produce paths [6,17,37,38].

Trajectory-based routing has the potential to preserve local drivers' knowledge more accurately than preference-based routing, since original trajectories are reused directly or



(a) Training Time for Learning Preferences **(b)** PCH Memory Consumption **(c)** Online Routing Run Time By Distances

Fig. 11 Evaluation of scalability

almost directly. However, trajectory-based routing is unable to support routing between arbitrary (s, d) pairs, since any set of historical trajectories is sparse in the sense that it is very unlikely to provide paths for all possible (s, d) pairs. Although a recent trajectory-based routing method [16] reuses historical trajectories to find paths between any (s, d) pair, this method needs to use preference-based routing to pre-construct paths that are pertinent to local drivers' knowledge for those parts of the underlying road network where no trajectories are available.

In general, the preference-based routing has potential beyond being a supplement to the trajectory-based routing—it is an important routing method that makes it possible to utilize the knowledge of local drivers as captured in trajectories to identify high-quality paths between arbitrary (s, d) pairs.

In the following, we first review how to learn routing preferences from trajectories and then how to conduct preference-based routing. Next, we review two other relevant topics—transferring labels in graphs and contextual preference modeling.

Learning routing preferences Several proposals [6,17,37–39] are able to learn personalized routing preferences. Specifically, a single routing preference is learned for a driver. We further categorize these studies into two classes. In the first class, a driver's preference is defined by a vector that gives different weights to different routing features, such as different travel costs or whether or not to use left turns. In particular, two methods [6,37] compare the paths used by a driver to skyline paths [11] to derive this kind of vector. Another method [40] employs linear programming to identify a vector. A recent study from Microsoft presents an algorithm that learns driver-specific parameters for Bing Maps' ranking function based on individual drivers' trajectories [38]. In the second class, TRIP [17] uses the ratios between individual drivers' travel times and average travel times as routing preferences to model personalized travel times and provide personalized fastest routing. We propose a novel routing preference space to model routing preferences. The superiority over the existing routing preference modeling approaches is demonstrated in Fig. 3 in Sect. 3.2.

In addition, we introduce a contextual preference tensor that is sufficiently general to be able to capture complex relationships between contexts and routing preferences. In particular, a personalized routing preference can be regarded as a contextual preference tensor with one single source/destination region as the spatial context and where the optional context only encompasses a driver dimension. The experiments show that the proposed *CPR* outperforms *Dom* [6] and TRIP [17], which represent the two classes of personalized routing.

We note that one recent study [39] employs trajectories to learn a ranking mechanism for a set of path without explicitly modeling the preferences and one study [41] considers risk preferences under travel time uncertainty, which are different from our proposal.

Preference-based routing After preferences are learned, preference-based routing algorithms [6,17,38,40] identify the "best" paths according to the learned preferences. Two categories of such algorithms can be distinguished. In the first category, learned preferences are applied on top of a set of skyline paths from which the algorithm picks the best path [6]. However, computing skyline paths is expensive, as also seen in Fig. 10f. In the second category, preferences are employed to derive personalized weights [17,38,40], and routing is conducted on the resulting personalized road network. In contrast, our preference-based routing method based on *PCH* does not need to maintain personalized weights, only preference vectors.

The proposed *PCH* differs from label-restricted contraction hierarchies (LRCH) [42]. LRCH target settings where the idea is to avoid specific labels (e.g., road types) during routing, while our method aims to include specific labels during routing. More importantly, in LRCH, the labels to be avoided are known before the CH is built and are removed during the construction of the CH. After that, standard routing can be applied using the CH. However, the CH must be updated every time the labels to avoid change. This is unac-

ceptable in our setting because we need to support routing preferences with different labels simultaneously. Thus, we build a PCH once and develop a novel and flexible preference-based routing algorithm on top of a PCH to support different RPs.

Transferring labels in graphs Many machine learning methods, including graph-based transductive learning [22,43] and graph convolutional neural networks [23–25,44,45], are able to transfer labels among nodes in a graph. However, most of them fail to scale well with regard to the graph size. Thus, these methods are not appropriate for our setting, as shown in the experiments covered in Sect. 6.3. We devise a novel multi-task learning on top of graph convolutional networks [25] to transfer labels among graph nodes in a scalable manner.

Contextual preferences Stefanidis et al. [46] construct a model for expressing contextual preferences of queries. They model context using hierarchical attributes and then use the contexts to personalize query results in relational databases. While the general idea is similar to ours, the application domains of the two papers are different: Stefanidis et al. focus on querying in relational databases, while we focus on road network routing. Therefore, their proposal is very different from our proposal.

8 Conclusion

We propose a context-aware, preference-based routing method to enable routing between arbitrary source and destination, with the goal of providing higher routing quality in an efficient manner by exploiting massive volumes of vehicle trajectory data. The proposal supports a generic method, in the form of a contextual preference tensor, of capturing and indexing context-aware routing preferences. It learns routing preference from trajectories for each preference. When trajectories are available and preferences are known in some contexts of the tensor, the proposed routing method offers efficient routing between source and destination guided by the given routing preference. For contexts where trajectories are unavailable and no preferences are known, the proposal transfers preferences from other contexts to the contexts with unknown preferences. The paper's empirical evaluation indicates that the overall solution is better than existing alternatives, is practical, and is able to deliver high-quality preference-based routing in an efficient and scalable manner. In future work, it is of interest to consider travel cost uncertainty [47,48] when both preference modeling and preference-based routing. It is also of interest to give contexts different weights when transferring routing preferences.

Acknowledgements This research was supported in part by a grant from the Obel Family Foundation, a grant from Independent Research Fund Denmark, and by the DiCyPS center, funded by Innovation Fund Denmark.

References

1. Guo, C., Jensen, C.S., Yang, B.: Towards total traffic awareness. SIGMOD Rec. **43**(3), 18–23 (2014)
2. Liu, H., Jin, C., Yang, B., Zhou, A.: Finding top-k optimal sequenced routes. In: ICDE, pp. 569–580 (2018)
3. Liu, H., Jin, C., Yang, B., Zhou, A.: Finding top-k shortest paths with diversity. IEEE Trans. Knowl. Data Eng. **30**(3), 488–502 (2018)
4. Pedersen, S.A., Yang, B., Jensen, C.S.: Fast stochastic routing under time-varying uncertainty. VLDB J (2019). https://doi.org/10.1007/s00778-019-00585-6
5. Ceikute, V., Jensen, C.S.: Routing service quality—local driver behavior versus routing services. In: MDM, pp 97–106 (2013)
6. Yang, B., Guo, C., Ma, Y., Jensen, C.S.: Toward personalized, context-aware routing. VLDB J. **24**(2), 297–318 (2015)
7. Aljubayrin, S., Yang, B., Jensen, C.S., Zhang, R.: Finding non-dominated paths in uncertain road networks. In: SIGSPATIAL, pp. 15:1–15:10 (2016)
8. Yang, B., Fantini, N., Jensen, C.S.: iPark: identifying parking spaces from trajectories. In: EDBT, pp. 705–708 (2013)
9. Dai, J., Yang, B., Guo, C., Jensen, C.S., Hu, J.: Path cost distribution estimation using trajectory data. PVLDB **10**(3), 85–96 (2017)
10. Yang, B., Dai, J., Guo, C., Jensen, C.S., Jilin, H.: PACE: a path-centric paradigm for stochastic path finding. VLDB J. **27**(2), 153–178 (2018)
11. Yang, B, Guo, C, Jensen, CS., Kaul, M, Shang S (2014) Stochastic skyline route planning under time-varying uncertainty. In: ICDE, pp. 136–147
12. Yang, B., Guo, C., Jensen, C.S.: Travel cost inference from sparse, spatio-temporally correlated time series using markov models. PVLDB **6**(9), 769–780 (2013)
13. Newson, P., Krumm, J.: Hidden Markov map matching through noise and sparseness. In: SIGSPATIAL, pp. 336–343 (2009)
14. Gonzalez, H., Han, J., Li, X., Myslinska, M., Sondag, J.P.: Adaptive fastest path computation on a road network: a traffic mining approach. In: VLDB, pp. 794–805 (2007)
15. Wei, L.-Y., Zheng, Y., Peng, W.-C.: Constructing popular routes from uncertain trajectories. In: SIGKDD, pp. 195–203 (2012)
16. Guo, C., Yang, B., Hu, J., Jensen, C.S.: Learning to route with sparse trajectory sets. In: ICDE, pp. 1073–1084 (2018)
17. Letchner, J., Krumm, J., Horvitz, E.: Trip router with individualized preferences (TRIP): incorporating personalization into route planning. In: AAAI, pp. 1795–1800 (2006)
18. Dai, J., Yang, B., Guo, C., Ding, Z.: Personalized route recommendation using big trajectory data. In: ICDE, pp. 543–554 (2015)
19. Bishop, C.M.: Pattern Recognition and Machine Learning (Information Science and Statistics). Springer, Berlin (2006)
20. Liu, J., Wright, S.J, Ré, C., Bittorf, V., Sridhar, S.: An asynchronous parallel stochastic coordinate descent algorithm. In: ICML, pp. 469–477 (2014)
21. Geisberger, R., Sanders, P., Schultes, D., Delling, D.: Contraction hierarchies: faster and simpler hierarchical routing in road networks. In: WEA, pp. 319–333 (2008)
22. Wang, Y., Yang, B., Qu, L., Spaniol, M., Weikum, G.: Harvesting facts from textual web sources by constrained label propagation. In: CIKM, pp. 837–846 (2011)
23. Bruna, J., Zaremba, W., Szlam, A., LeCun, Y.: Spectral Networks and Locally Connected Networks on Graphs (2013). arXiv:1312.6203 [cs]

24. Defferrard, M., Bresson, X., Vandergheynst, P.: Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering (2016). arXiv:1606.09375 [cs, stat]
25. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: ICLR (2017)
26. Cirstea, R.-G., Micu, D.-V., Muresan, G.-M., Guo, C., Yang, B.: Correlated time series forecasting using multi-task deep neural networks. In: CIKM, pp. 1527–1530 (2018)
27. Kieu, T., Yang, B., Guo, C., Jensen, C.S.: Distinguishing trajectories from different drivers using incompletely labeled trajectories. In: CIKM, pp. 863–872 (2018)
28. Deng, D., Shahabi, C., Demiryurek, U., Zhu, L., Yu, R., Liu, Y.: Latent space model for road networks to predict time-varying traffic. In: SIGKDD, pp. 1525–1534 (2016)
29. Kieu, T., Yang, B., Guo, C., Jensen, C.S.: Outlier detection for time series with recurrent autoencoder ensembles. In: IJCAI, pp. 2725–2732 (2019)
30. Kieu, T., Yang, B., Jensen, C.S.: Outlier detection for multidimensional time series using deep neural networks. In: MDM, pp. 125–134 (2018)
31. Guo, C., Yang, B., Andersen, O., Jensen, C.S., Torp, K.: Ecomark 2.0: empowering eco-routing with vehicular environmental models and actual vehicle fuel consumption data. GeoInformatica **19**(3), 567–599 (2015)
32. Guo, C., Ma, Y., Yang, B., Jensen, C.S., Kaul, M.: Ecomark: evaluating models of vehicular environmental impact. In: SIGSPATIAL, pp. 269–278 (2012)
33. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. J. Mach. Learn. Res. **13**, 281–305 (2012)
34. Chen, Z., Shen, H.T., Zhou, X.: Discovering popular routes from trajectories. In: ICDE, pp. 900–911 (2011)
35. Ceikute, V., Jensen, C.S.: Vehicle routing with user-generated trajectory data. In: MDM, pp. 14–23 (2015)
36. Luo, W., Tan, H., Chen, L., Ni, L.M: Finding time period-based most frequent path in big trajectory data. In: SIGMOD, pp. 713–724 (2013)
37. Balteanu, A., Jossé, G., Schubert, M.: Mining driving preferences in multi-cost networks. In: SSTD, pp. 74–91 (2013)
38. Delling, D., Goldberg, A.V., Goldszmidt, M., Krumm, J., Talwar, K., Werneck, R.F.: Navigation made personal: inferring driving preferences from GPS traces. In: SIGSPATIAL, pp. 31:1–31:9 (2015)
39. Yang, S.B., Yang, B.: Learning to rank paths in spatial networks. In: ICDE (**to appear**) (2020)
40. Funke, S., Laue, S., Storandt, S.: Deducing individual driving preferences for user-aware navigation. In: SIGSPATIAL, pp. 14:1–14:9 (2016)
41. Jilin, H., Yang, B., Guo, C., Jensen, C.S.: Risk-aware path selection with time-varying, uncertain travel costs: a time series approach. VLDB J. **27**(2), 179–200 (2018)
42. Rice, M.N., Tsotras, V.J.: Graph indexing of road networks for shortest path queries with label restrictions. PVLDB **4**(2), 69–80 (2010)
43. Talukdar, P.P., Crammer, K.: New regularized algorithms for transductive learning. In: ECML/PKDD, pp. 442–457 (2009)
44. Hu, J., Guo, C., Yang, B., Jensen, C.S.: Stochastic weight completion for road networks using graph convolutional networks. In: ICDE, pp. 1274–1285 (2019)
45. Hu, J., Yang, B., Guo, C., Jensen, C.S., Xiong, H.: Stochastic origin-destination matrix forecasting using dual-stage graph convolutional, recurrent neural networks. In: ICDE (**to appear**) (2020)
46. Stefanidis, K., Pitoura, E., Vassiliadis, P.: Managing contextual preferences. Inf. Syst. **36**(8), 1158–1180 (2011)
47. Jilin, H., Yang, B., Jensen, C.S., Ma, Y.: Enabling time-dependent uncertain eco-weights for road networks. GeoInformatica **21**(1), 57–88 (2017)
48. Pedersen, S.A., Yang, B., Jensen, C.S.: A hybrid learning approach to stochastic routing. In: ICDE (**to appear**) (2020)