Personalized Travel Time Prediction Using a Small Number of Probe Vehicles

YANG LI, Tsinghua-Berkeley Shenzhen Institute, China
DIMITRIOS GUNOPULOS, National and Kapodistrian University of Athens, Greece
CEWU LU, Shanghai Jiao Tong University, China
LEONIDAS J. GUIBAS, Stanford University, United States

Predicting the travel time of a path is an important task in route planning and navigation applications. As more GPS probe data has been collected to monitor urban traffic, GPS trajectories of the probe vehicles have been frequently used to predict path travel time. However, most trajectory-based methods rely on deploying GPS devices and collect real-time data on a large taxi fleet, which can be expensive and unreliable in smaller cities. This work deals with the problem of predicting path travel time when only a small number of cars are available. We propose an algorithm that learns local congestion patterns of a compact set of frequently shared paths from historical data. Given a travel time prediction query, we identify the current congestion patterns around the query path from recent trajectories, then infer its travel time in the near future. Driver identities are also used in predicting personalized travel time. Experimental results using 10–25 taxis in urban areas of Shenzhen, China, show that personal prediction has on average 3.4mins of error on trips of duration 10–75mins. This result improves the baseline approach of using purely historical trajectories by 16.8% on average, over four regions with various degrees of path regularity. It also outperforms a state-of-the-art travel time prediction method that uses both historical trajectories and real-time trajectories.

CCS Concepts: • **Information systems** → **Data mining**; *Data streaming*; *Global positioning systems*; • **Applied computing** → *Transportation*;

Additional Key Words and Phrases: Travel time prediction, mobile sensors, GPS trajectories

ACM Reference format:
Yang Li, Dimitrios Gunopulos, Cewu Lu, and Leonidas J. Guibas. 2019. Personalized Travel Time Prediction Using a Small Number of Probe Vehicles. *ACM Trans. Spatial Algorithms Syst.* 5, 1, Article 4 (May 2019), 27 pages.
https://doi.org/10.1145/3317663

This research was funded by NSF grants CCF-1514305 and DMS-1521608, ONR MURI grant N00014-13-1-0341, European Union Horizon2020 grant 688380 "VaVeL," a Google 2017 Faculty Research Award, and a gift from Google.
Authors' addresses: Y. Li, Tsinghua-Berkeley Shenzhen Institute, Building C2, Nanshan Park, #1001 Xueyuan Ave. Shenzhen, Guangdong, 518052, China; email: yangli@sz.tsinghua.edu.cn; D. Gunopulos, National and Kapodistrian University of Athens, 30 Panepistimiou street, Athens, 10679, Greece; email: dg@di.uoa.gr; C. Lu, Shanghai Jiao Tong University, 800 Dongchuan Rd. Shanghai, 200240, China; email: lucewu@sjtu.edu.cn; L. J. Guibas, Stanford University, 450 Serra Mall, Stanford, California, 94305, United States; email: guibas@cs.stanford.edu.

2374-0353/2019/05-ART4 $15.00
https://doi.org/10.1145/3317663

1 INTRODUCTION

Modern navigation and location-based applications rely on accurately predicting the travel time of a route at the current or future times. Due to the prevalence of traffic congestion in urban cities and the large variance of conditions in the traffic environment, the travel time of a route can vary significantly from hour to hour, day to day. Therefore the use of dynamic traffic data is extremely important in accurate travel time prediction. Traditional ways of monitoring traffic conditions use static sensors (e.g., induction loops, automatic license-plate-number recognition cameras) installed on selected streets and highways in the city. These data collection methods tend to be less up-to-date and are difficult to aggregate and maintain. As GPS devices have become mainstream in the recent decade, it becomes possible to estimate and predict traffic conditions from large trajectory data collected by GPS-equipped vehicles.

A great number of studies have been published on trajectory-based travel time prediction (Chen and Chien 2001; Hofleitner and Bayen 2011; Jiang and Li 2013; Rahmani et al. 2013; Wang et al. 2014b; Wu et al. 2004; Zhan et al. 2013; Zhang et al. 2016). Most of them make use of thousands of probe vehicles tracked simultaneously, such that the traffic speed on a subset of the roads is observed by at least one vehicle within a short time window. However, the large-scale deployment of GPS-equipped vehicles and the cloud infrastructure to process such data can still be unfeasible for smaller cities. The deployment process also takes time. For instance, the Gotcha project had deployed dozens of sensor-equipped electric taxis in Shenzhen, China, in several stages from 2014 to 2017 (Xu et al. 2014). During the pilot stage (Stage I), no more than 15 vehicles from the project were active (see Figures 1 and 2).

In this article, we use the Gotcha pilot data to investigate the difficult problem of travel time prediction when having a very small number of concurrently active probe vehicles on the road network. This constraint comes up in many real-life situations. Although typically many GPS tracks are available when we look at the full history of available trajectories as a whole, at a specific time instance, only a potentially much smaller subset will be available. Equally importantly, real-time information at any given time may be available from a smaller subset of the active probe vehicles. For example, when crowd-sourced trajectories are used, some drivers may decide to upload their traces in batches rather than immediately to conserve battery power, or when there is Wi-Fi. We show how latent structures in historical trajectories can be exploited to predict travel time with sparse observations at a given time. Applying this, we develop and evaluate a novel, flexible, and efficient mechanism that significantly improves travel prediction times.

This problem has two main challenges: The most obvious one is data sparsity. In the 15-vehicle dataset, on average only 3% of all road links are traversed during a 30-minute interval on a weekday morning. The other challenge is the large variance in travel time observations of the same path. Gotcha trajectories do not have labels to identify passenger trips, as do trajectories in previous works. While most travel time delays in taxi trajectories are likely caused by congestion, it may also include the times when drivers stop temporarily to drop off or pick up passengers or when drivers intentionally slow down to find passengers on the sidewalk. Due to the lack of trip labels, our problem is akin to using crowd-sourced trajectories. Therefore, our solution will need to handle both data sparsity and an amount of uncertainty in travel time observations.

To address these challenges, we use a **path-based** approach that decomposes a route into a sequence of popular paths on the road network and predicts the travel time on each path (Wang et al. 2014b). This approach differs from the popular **link-based** design, which predicts the travel time of a route based on the estimated time of each road segment, also called a *link*. We prefer the path-based approach, because path travel time also includes link-delays, the time spent transitioning from one link to the next. Such delay is difficult to estimate independently, given the sparse and uncertain nature of our problem. In this work, we refer to the popular paths as **pathlets**, selected

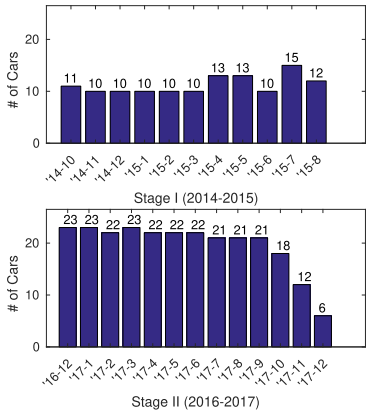

Fig. 1. The number of taxis employed each month by the Gotcha study.

Fig. 2. GPS traces of 10 taxis during the month of December 2014.

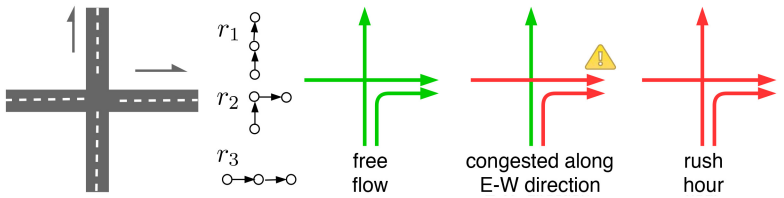

Fig. 3. Visualization of three congestion patterns over pathlets r_1, r_2, and r_3 at an intersection. Colors red and green represent congested and non-congested traffic states of a given pathlet.

based on the shared geometry in taxi trajectories. Although travel time information is not used directly to decompose a path, we can still control pathlet selection through a parameter learned from the training data.

The key to our travel time prediction method is leveraging hidden structures within historical travel time observations to infer the travel time of a path that has not been traversed by any probe vehicle recently. We observe that, on a local scale, such as several neighboring roads (or overlapping pathlets), the congestion patterns are reoccurring (see Figure 3). Our algorithm clusters the local congestion patterns of a pathlet across the entire time span of the historical data. Periodic factors including time of day and workday are implemented as soft clustering constraints rather than hard constraints. Hence, it allows traffic delays caused by random events, such as weather, accidents, and special events, to influence the clustering results. By learning such latent patterns, we are able to infer the near-future travel time of any path in the neighborhood if only a few paths have been observed in recent time.

In addition, research in human cognition and transportation science has shown that different drivers may have significantly different speed profiles under the same driving conditions (McNew 2012; Reymond et al. 2001). Hence, it is useful to incorporate driver identity into travel time prediction, especially for personal navigation, as well as estimating arrival time for taxi and ride-sharing services. A common approach to obtain a driver's speed profile is by modeling his or her driving behavior. A traditional driver model often requires accurate measurement of acceleration, throttle index, and other parameters of the vehicle. However, as our study focuses on GPS trajectory data of moderate sampling rate (<1Hz), we adopt a purely data-driven approach by clustering drivers into groups based on how fast they drive compared to others over the frequent pathlets. We

then extract congestion patterns while normalizing the differences in travel time distributions of different driver groups. At the query time, we determine which group a given driver belongs to using his or her recent trajectories, then predict the travel time accordingly.

The contributions of this work are threefold:

(1) Proposing a personalized travel time prediction framework that combines the prediction based on the current congestion pattern and the historical travel time of each pathlet.
(2) Extracting local congestion features by exploiting spatial relations among pathlets in a neighborhood.
(3) Developing an unsupervised learning approach to find congestion patterns that are robust against missing data.

This work is an extended version of (Li et al. 2017), which discusses travel time prediction without using driver identity information. The new contribution in the extended version is mainly on personalized prediction. We evaluate personalized prediction results on a new dataset containing GPS trajectories of 25 taxis during 2016–2017. The experiment results show a significant improvement in prediction accuracy over the non-personalized approach. The prediction accuracy also increases when more probing cars are employed. In addition, we provide details on how to process raw GPS trajectories into trips and how to predict travel time for a given query path for both the non-personalized and personalized scenarios. While Li et al. (2017) focus on extracting congestion patterns from travel time observations, this work presents a more complete picture of how our trajectory-based travel time prediction framework can be applied in practice.

2 PROBLEM DEFINITION

We represent a **GPS trajectory** as a sequence of n spatial-temporal points: $\tau = \{(p_i, t_i)\}_{i=1}^{n}$. Point $p_i = (x_i, y_i)$ is the GPS position projected in $\mathbb{R}^2$ and t_i is the sample timestamp. We define the cardinality of trajectory τ, $|\tau| = n$ as the number of GPS samples in τ.

Given the road network G and a trajectory τ, we can infer the path in G that τ represents through a process called **map matching**. We call this path the **map-matched path** of τ, written as a sequence of edges $\tau_G = \{e_1, \ldots, e_{|\tau_G|}\}$.

We formally define the travel time prediction problem as follows:

Definition 2.1 (Travel Time Prediction). Let T be a collection of historical trajectories on road network G. Let subset $R_\beta \subseteq T$ be the set of recent trajectories collected in the last β minutes. Given a query path P and the current time t, compute $d_t(P)$, the predicted travel time of a trip along P departing at t, based on trajectories in T and R_β.

The above problem computes the expected travel time of the query path over the entire driver population. To predict the travel time for a particular driver, we introduce the following problem:

Definition 2.2 (Personalized Travel Time Prediction). Let T, G, R_β, and β be defined as in Definition 2.1. Denote $u(\tau) \in \{1, \ldots, N\}$ to be the driver identity for each trajectory $\tau \in T$. Given a query path P and the current time t, predict the travel time for any driver u whom has some historical trajectories recorded in T.

In addition, we only study trips in the near future (10mins–1.5h). As we predict further into the future, the traffic status is less predictable by observations in R_β. In that case, our prediction will not be able to reflect delays due to random factors. One workaround for longer trips is to predict the travel time of an initial segment of the path using R_β, then predict the next segment when recent trajectories in R_β are updated.

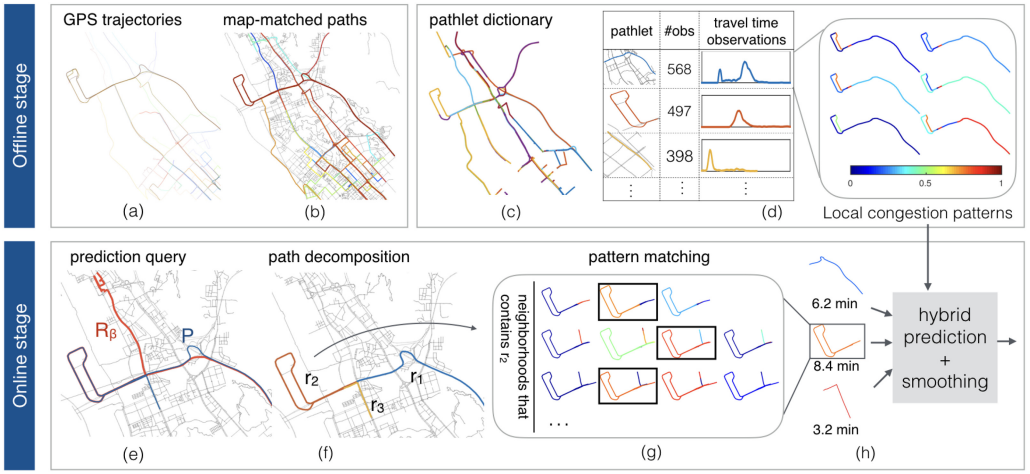

Fig. 4. Our travel time prediction framework consists of the offline stage that preprocesses data (a–b) and learns travel time patterns (c–d), and the online stage that process travel time prediction queries (e–h). See Section 3 for details.

3 ALGORITHM OUTLINE

Figure 4 illustrates our travel time prediction framework in three stages: (i) trajectory preprocessing, (ii) offline congestion pattern clustering, and (iii) online travel time prediction.

Preprocessing. Given unprocessed input trajectories shown in Figure 4(a), we partition each trajectory into trips by removing long gaps and *staying points*, i.e., clusters of GPS samples recorded when a car is not moving for an extended period of time. Then, we apply map matching to compute the map-matched path τ_G of every input trajectory τ (Figure 4(b)). We abuse the notation of T and R so they refer to trajectories after partitioning and map matching.

Congestion pattern learning. First, we compute the *pathlet dictionary*, a compact set of pathlets that are used to reconstruct the input trajectories. The top 100 most frequently used pathlets are visualized in Figure 4(c), along with three dictionary entries shown in the table. Each pathlet r in the dictionary is associated with a set of travel time observations at different times and dates from all trajectories in T. The dynamic congestion feature of r is computed by aggregating its travel time observations into *frames*, which are fixed time intervals within the entire date-range of the input data, e.g., 8:00am–8:30am on December 2, 2014. Driver identity is also used in this aggregation process, such that each group of drivers with similar driving styles has a unique distribution of travel time observations on r. Using the spatial and temporal relationships between travel time features, we extract a set of distinctive congestion patterns $C(r)$ among congestion features of pathlets in r's neighborhood. Figure 4(d) visualizes six congestion patterns from a pathlet's neighborhood using a color scale, where red implies the most congested state and blue implies free-flow state. Details are discussed in Sections 6 and 7.

Travel time prediction. Figure 4(e) shows the sample input of the online stage: recent trajectories R_β (red curves) and the query path P (blue curves). First, P is decomposed into three pathlets r_1, r_2, and r_3. For each r_i with $i = \{1, 2, 3\}$, we identify the current congestion pattern of any observed neighborhoods that contain r_i, and use them to predict $d_t(r_i)$, the travel time of r_i departing at t. For instance, Figure 4(g) shows the congestion patterns for three neighborhoods that contain r_2, and the patterns closest to observations in R_β in each neighborhood are highlighted by the black boxes.

The final travel time prediction of r_2, $d_t(r_2, u) = 8.4$mins is computed based on the predictions from each neighborhood. Lastly, we combine the predictions based on pattern matching and the historical travel time of each pathlet to obtain the travel time of path P (Figure 4(h)).

To make a personalized prediction for driver u, we first determine the most likely driver group based on its historical trajectories. Then, in Figure 4(h), we derive the travel time from the estimated congestion status using the travel time distribution for u's driver group. Details on the algorithm will be presented in Section 8.

4 RELATED WORKS

This section reviews previous works on the travel time prediction problem for urban road networks and driver-specific predictions. First, we classify existing travel time prediction algorithms in three main categories.

Link-based travel time prediction. Link-based approaches are the classical method to predict travel time on a road network. They are similar to prediction techniques designed for static traffic sensors, such as induction loop (Wu et al. 2004) and license-plate identification cameras (Chen et al. 2013a).

For probe vehicle data (a.k.a. floating car data), the travel time of individual links can be inferred by trajectories of cars passing through those links. This is called the *link travel time estimation problem*. For instance, Hofleitner and Bayen (2011) model the travel time distributions of links based on a traffic flow model. Zhan et al. (2013) uses least-square minimization to estimate link travel time from taxi trip data that only contain endpoint locations and meta information about the trip such as trip distances. More generally, one can estimate traffic parameters, such as the speed and the flow volume (De Fabritiis et al. 2008, Zhan et al. 2017) associated with individual links to infer link travel time. These works focus on inferring the current traffic parameters, rather than predicting the future.

Various prediction methods have been proposed to predict link travel time in the near future, such as dynamic Bayesian network (Hofleitner et al. 2012), pattern matching (Chen et al. 2013a), gradient boosting regression tree (Zhang et al. 2016), and deep learning (Niu et al. 2014). In both link travel time estimation and prediction problems, correlations between the travel time for nearby links (spatial) and different time windows (temporal) are often used to select the relevant features for inferring the traffic parameter on a particular link (Niu et al. 2014; Zhang et al. 2016). Our algorithm also makes use of spatial-temporal relationships on traffic states, but on a path level.

Many studies compute the travel time of a path as a summation of the predicted link travel time. This approach has the drawback that link-delays are not considered. In Rahmani et al. (2013), the authors designed several correction methods to take into account the travel time bias in the additive link-based travel time model. Yet such models require good dynamic coverage of the road network. As a result, these works only focus on a specific highway region or a few selected routes.

Path-based travel time prediction. In an early work that advocates the computation of path-based travel time over link-based travel time (Chen and Chien 2001), researchers demonstrated that the direct measuring of path-based travel time on a highway strip could generate a more accurate prediction than measuring link travel time independently.

Since it is not always possible to have a travel time measurement on an arbitrary path, large-scale path-based travel time prediction needs to decompose the query path into popular subpaths, whose travel time is more likely to be measured by some probe vehicles. Wang et al. (2014b) discussed the trade-off between subpath lengths and the minimum support size in path-based prediction. It computes the optimal decomposition by minimizing the total travel time variance of subpaths, normalized over the number of unique drivers on each subpath. It is worth noting that

path decomposition (partition) is also an important problem in trajectory compression on road networks. Earlier works on this topic are summarized in Sun et al. (2016) and Kellaris et al. (2013). In particular, Chen et al. (2013b) finds a compact dictionary of pathlets that reconstruct trajectories by fewer pieces (more compressed trajectory). We adopt this technique in our algorithm, since it is designed to maximize the path regularity in input trajectories. Path decomposition using this approach allows the same observations to account for the prediction of more query paths. It also allows finer control of the trade-off by a single parameter.

Some approaches, such as Chen and Chien (2001), use only historical data to predict the travel time of a query. In Wang et al. (2014b), trajectories in the recent past are used to estimate the current travel time of the query path. The historical travel time of each road link, calculated using tensor decomposition over spatial-temporal features and driver identities, is only used when recent observations are not available.

Trip-based travel time prediction. Trip-based methods rely on finding historical trips that match the origin, destination, and departure time interval of the query (Jiang and Li 2013; Wang et al. 2016; Xu et al. 2017). They usually assume trips between the same endpoints share the same route or a small number of alternative routes. Therefore, it is more often used for coarse-level prediction or predictions of predetermined routes such as bus trips. Jiang and Li (2013) compute the travel time distribution of a query trip from matched historical trajectories and use statistical tests to remove outliers. Wang et al. (2016) infer the travel time of trips that are not found in historical data from nearby trips, while adjusting for periodic traffic patterns. Deep-learning techniques have also been employed in travel time prediction from a large number of historical trips and their attributes (Xu et al. 2017). Trip-based travel time prediction can also be extended hierarchically to allow some route diversity. In Yuan et al. (2010), they represent a trip as a sequence of shorter trips between popular landmarks in the city. Trip travel times are computed as the sum of landmark-to-landmark travel times, plus the time spent from the origin to the first landmark, and from the last landmark to the destination. While trip-based travel time prediction has much better performance than link-based or path-based algorithms, while achieving useful results, they cannot be applied to our scenario, as we cannot reliably identify the true starting and ending points of a taxi trip in an unlabeled GPS trace.

Driver modeling and personalized predictions. Driver modeling is important for building accurate traffic simulation and advanced driver assistance systems. In transportation research, there have been many studies modeling drivers' behaviors for various tasks, such as car-following, lane-changing, route selection, and so on (Wang et al. 2014a). Such behaviors constitute the unique profile of a driver among other drivers on the same road. Some works independently predict certain states of individual drivers. For instance, McNew (2012) predicted driver-specific cruise speed using travel speed during the "launch time" of a car and the speed limit. Other works tend to categorize drivers into groups based on an "aggressiveness" metric, which characterizes the tendency of speeding and other risky behaviors. Kedar-Dongarkar and Das (2012) classified drivers into three categories (conservative, moderate, and aggressive) based on dynamic parameters including acceleration, braking, speeding index, and throttle activity index. A similar categorization is proposed in Shi et al. (2015), using an aggressiveness index (AggIn) computed based on the detected drivers' behaviors.

5 PREPROCESSING

The preprocessing step takes the raw GPS trajectories of floating cars as input and generates a collection of paths on the road network that correspond to the input trajectories. It consists of two tasks: trajectory partitioning and map matching.

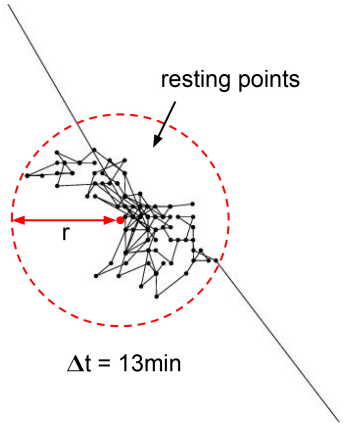

Fig. 5. Computation of resting points.

The first task, trajectory partitioning, involves extracting trips from GPS trajectories that are tracked continuously for each vehicle during data collection. For taxi trajectories, each GPS sample may include a binary status that indicates whether a passenger is on board. In this case, we can easily extract trips as the sub-trajectory between a pick-up event and a drop-off event, marked by a change in the status variable. However, since our data does not contain such information, we partition trajectories by "stay points," i.e., segments of the trajectory sampled when the car is not moving for some extended period of time, and by large temporal gaps. Although extracting trips or finding stay points (also referred as stop points or dwell points in the literature) from vehicle trajectories are challenging problems on their own (Du and Aultman-Hall 2007; Xiang et al. 2016; Zheng et al. 2009), for the scope of this project, we adopt a simple, conservative definition of stay points.

Definition 5.1 (Stay Points). Given trajectory $(p_n, t_n), \ldots, (p_n, t_n)$, the sequence of GPS points $p_i, \ldots, p_{i+w}$ with $1 \leq i \leq n - w$ are **stay points** if

(1) $t_{i+w} - t_i > d_{min}$ minutes and
(2) $\max_{j=i}^{i+w} ||p_j - \bar{p}||_2 < l_{max}$ meters where $\bar{p}$ is the centroid of the sequence.

Criterion (1) makes sure stops that are shorter than d_{min} minutes, such as when a car waits during a red light or a traffic jam, are excluded from real stop points.[1] Criterion (2) requires the vehicle to be stationary at a stay point, assuming the maximum deviation of GPS measurement is less than l_{max} meters. As shown in Figure 5, the GPS signal will hop around even when a car is still. In practice, we choose parameters based on the estimated GPS noise in the data. Conservative constraints are preferred, e.g., $d_{min} = 3$ and $l_{max} = 10$, to reduce the amount of outliers with extremely long travel time not caused by traffic congestion at the cost of having slightly less observations.

Based on Definition 5.1, we use a sliding window filter to identify stay points and replace each stay point sequence by two new points $(\bar{p}, t_i)$ and $(\bar{p}, t_i + w)$ located at the centroid. Then, we partition the trajectory when the difference between consecutive timestamps is greater than t_{max} minutes. The value of t_{max} has to be greater than d_{min}. We chose $t_{max} = 10$ minutes in our problem.

[1] In practice, we also partition trajectories in T by a maximum duration of $L = 1hr$ and remove trajectories shorter than $L = 4$ samples. Both map matching and travel time observation work less accurately when the trajectory is extremely short or extremely long. Experiments show that varying L between 30 minutes and 2 hours and changing starting positions does not significantly impact prediction results.

Note that for taxi trajectories, the aforementioned approach is not intended for finding a one-to-one correspondence between trajectories and passenger trips. Each output trajectory may start with a passenger pick-up, followed by a quick drop-off, and continue while the driver roams around the city until the next detected stop. Comparing to a typical passenger trip, these kinds of trips are prone to have larger travel time variance. We will explain in later sections how we handle this issue.

The second task of preprocessing is map matching. We project each partitioned trajectory $\tau \in H$ to its actual path τ_G on road network G. Since the sampling rate in our data is moderately high (10s per sample), we use the IVMM algorithm (Yuan et al. 2010) to perform map matching efficiently. For trajectories with a lower sampling rate, the joint map-matching algorithm (Y. Li and Guibas 2013) is recommended.

6 COMPUTING PATHLET TRAVEL TIME

The first task in the offline stage is obtaining travel time observations of pathlets from GPS trajectories. In this section, we will first review the formal definition of pathlets and the pathlet dictionary proposed by Chen et al. (2013b). Then, we will discuss how to derive pathlet travel time observations from input trajectories.

6.1 Pathlet Dictionary

A **pathlet** is a subpath on the road network that is traveled by one or more input trajectories. A **pathlet dictionary (PD)** is a set of pathlets that reconstructs all input trajectories T by concatenation. Let $p(\tau_G) = \{r_1, \ldots, r_m\} \subseteq PD$ denote the set of pathlets in the dictionary that reconstruct τ_G. We define the **support set** of a pathlet $r \in PD$, $\mathcal{T}(r)$ to be the set of trajectories that uses r in its decomposition, i.e., $\mathcal{T}(r) = \{\tau_G \in T \mid r \in p(\tau_G)\}$.

Given input trajectory set T, the **optimal pathlet dictionary** satisfies the following criteria: (i) Number of pathlets in the dictionary, $|PD|$ is minimized. (ii) For each $\tau_G \in T$, the number of pathlets used to reconstruct τ_G, $|p(\tau_G)|$ is minimized. It is shown in Chen et al. (2013b) that the optimal dictionary P can be learned by solving the following optimization problem. Let $x_{\tau,r}$ be an indicator variable that evaluates to 1 if $r \in p(\tau_G)$ and 0 otherwise. For each trajectory $\tau_G \in T$, the solution minimizes the following problem:

$$\min_{x_{\tau,r} \in \{0,1\}} \sum_{r \in p(\tau)} \left(\lambda + \frac{1}{|\mathcal{T}(r)|}\right) x_{\tau,r}. \tag{1}$$

Parameter λ determines the trade-off between objectives (i) and (ii). The smaller the value of λ, the smaller the dictionary size $|PD|$, and the larger the average trajectory decomposition size $|p(\tau)|$. Although 1 is an NP-hard problem to solve exactly, Chen et al. (2013b) presented an approximation algorithm to find solutions in $O(|T|)$. After a dictionary is computed, it is easy to query the decomposition of trajectories in the dataset using graph search.

The theoretical optimal pathlet dictionary does not guarantee that it covers the entire map, since there could be certain roads not traversed by any GPS trajectories in the dataset. This may be a problem when we later use the learned dictionary to decompose future trajectories. In practice, we solve this problem by augmenting the computed pathlet dictionary with edges of the road network. During the decomposition process, the added edges are treated as length one pathlets and have the least weight to be included in the decomposition of any trajectory. Another practical consideration for computing a pathlet dictionary is setting an upper bound on the length of a pathlet, thus making the computation scalable towards larger networks. For our experiments, we constrain the pathlet to contain no more than 20 road segments.

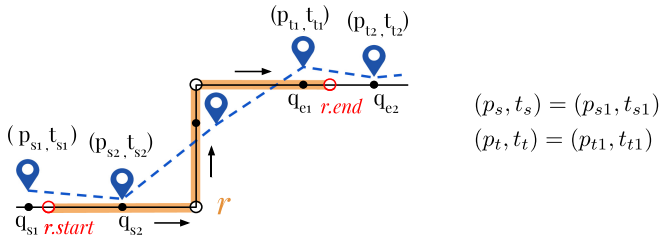

Fig. 6. This figure illustrates an example of computing the travel time observation of pathlet r (highlighted in orange) by a trajectory, which contains five GPS samples (blue drop pins) projected to pathlet r and its adjacent road links.

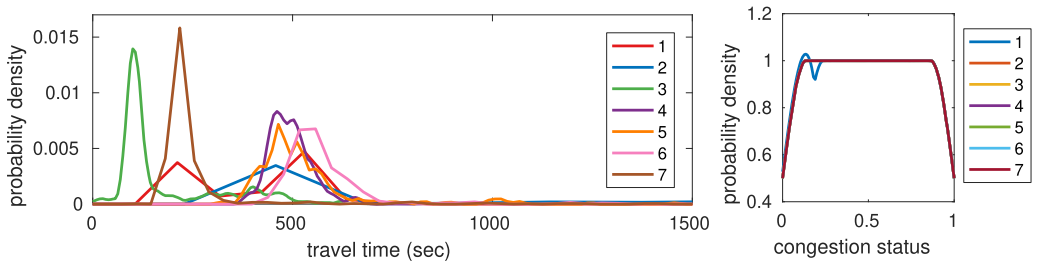

Fig. 7. (a) Travel time distribution of the seven most frequently used pathlets. (b) Distribution of the congestion status of the same pathlet using probability integral transformation.

6.2 Pathlet Travel Time Observations

We compute the travel time observation of a trajectory τ_G on some pathlet r as follows: First, we find the last sample point (p_{s1}, t_{s1}) in τ before it enters pathlet r and the first sample point (p_{s2}, t_{s2}) projected onto r (see Figure 6). Similarly, we find (p_{e1}, t_{e1}) and (p_{e2}, t_{e2}), the last GPS sample projected on r and the first GPS sample exiting r, respectively. Let q_{s1}, q_{s2}, q_{e1} and q_{e2} be the projections of these points on the road network. We define the nearest GPS samples (p_s, t_s) and (p_e, t_e) from the endpoint vertices of r, $r.start$, and to $r.end$:

$$(p_s, t_s) = \begin{cases} (p_{s1}, t_{s1}) & \text{if } dist(q_{s2}, r.start) < dist(q_{s1}, r.start), \\ (p_{s2}, t_{s2}) & \text{otherwise,} \end{cases}$$

$$(p_t, t_t) = \begin{cases} (p_{e1}, t_{e1}) & \text{if } dist(q_{e2}, r.end) < dist(q_{e1}, r.end), \\ (p_{e2}, t_{e2}) & \text{otherwise.} \end{cases}$$

Distance function $dist(x, y)$ is the geodesic distance from x to y along the path τ_G. The travel time of pathlet r observed by trajectory τ_G, $d_\tau(r)$ is the scaled timestamp difference $t_e - t_s$:

$$d_\tau(r) = \frac{dist(q_s, q_e)}{length(r)}(t_e - t_s).$$

Let $\mathcal{D}(r)$ be the set of all travel time observations of pathlet r from the input trajectories. Figure 7 shows the travel time distribution of the top-seven pathlets used to reconstruct input paths. Most of them are skewed towards larger values due to outliers that represent extremely long delays. To reduce the bias of outliers, we apply the *probability integral transformation* (Angus 1994) to $d_\tau(r)$, such that the transformed value, $\hat{d}_\tau(r) = cdf(D(r), d_\tau(r))$, is in a uniform distribution between 0 and 1. This transformation allows us to compare travel times between different pathlets.

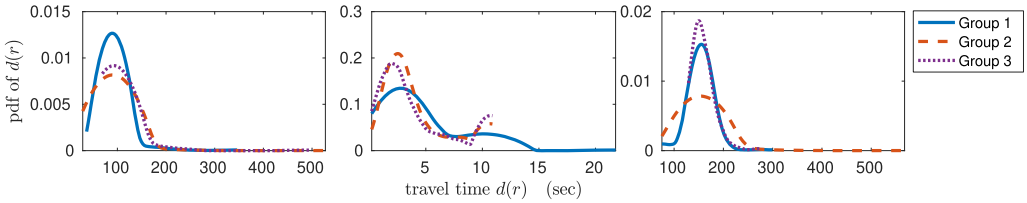

Fig. 8. Group-specific travel time distributions for the top three most popular pathlets in the dictionary. Each color represents a driver group.

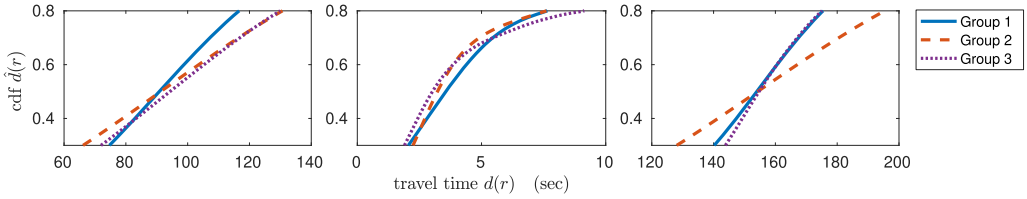

Fig. 9. Group-specific probability integral functions for the top three most popular pathlets in the dictionary, zoomed to show the differences between driver groups.

6.3 Group-specific Travel Time Observations

For personalized prediction, we need to learn the pathlet travel time distribution for each driver. Since one driver cannot visit all pathlets on the road network on a regular basis, it is best to utilize travel time observations from drivers with a similar preferred driving speed.

We cluster all N drivers into K groups based on their historical travel time for every pathlet in the dictionary. Let $d(u, i, j)$ denote Driver u's average travel time on the jth pathlet during the ith time-of-day interval ($1 \leq i \leq D$). We represent Driver u as a $|PD| \times D$ dimensional point $f(u) = [d(u, 1, 1), \ldots, d(u, D, 1), d(u, 1, 2), \ldots, d(u, D, |PD|)]$ in the feature space. Among many clustering techniques, we choose Ward's hierarchical clustering algorithm, since the driver groups it produces are often more balanced and consistent under parameter perturbations than other approaches. The cluster number K is set to 3, as drivers are often categorized into conservative, moderate, and aggressive groups in transportation literatures such as McNew (2012) and Shi et al. (2015).

Next, we describe how to normalize travel time observations from different driver groups into a common metric of congestion states. Given a driver group G_i, we define $\mathcal{D}(r, G_i) = \{d_\tau(r) \mid u(\tau) \in G_i\}$ to be the set of travel time observations in this group. Similar to the driver-independent scenario, we estimate the travel time distribution of pathlet r for each driver group G_i and apply probability integral transformation to observation $d_\tau(r)$:

$$\hat{d}_{\tau,G_i}(r) = cdf(\mathcal{D}(r, G_i), d_\tau(r)).$$

Figures 8 and 9 illustrate the probability density of $d_\tau(r)$ and the probability integral transformation function $cdf(D(r, G_i), \cdot)$ for each driver group on three sample pathlets. When the user group is known, we simply write the resulting congestion state as $\hat{d}_\tau(r)$.

7 LEARNING CONGESTION PATTERNS

In this section, we formulate the problem of learning congestion patterns and subsequently present algorithms to infer the congestion state of each pathlet at a given time from historical data. To address the problem of sparse concurrent observations, both spatial and temporal relationships are exploited.

7.1 Design Features with Spatial Relationship

We design features that capture local traffic patterns using spatial relationships among the pathlets. In particular, we observe that traffic states of pathlets that share common edges are not independent. Therefore, we define a neighborhood near pathlet r by selecting pathlets with a significant amount of overlap with r.

Specifically, let the *overlap ratio* of pathlet r' with respect to r, $o(r, r')$ be defined as the fraction of shared edges between the two pathlets. We further define the *overlapping neighborhood of* r, $OL(r) = \{o_1, \ldots, o_s\}$ as the set of s pathlets with the highest overlapping ratios with respect to r.

To capture the dynamic congestion state of neighborhood $OL(r)$, we aggregate travel time observations of r by discrete time steps (or *frames*) across the entire date range of the input trajectories. Due to the constraint of having very few probe vehicles, most frames either contain no observations or only contain a few observations. For each frame f_i with one or more observations, we use the median operator to aggregate observations into a single congestion state $\hat{d}_{f_i}(r)$. The step size is chosen empirically based on the observation sparsity. With 15–25 floating cars, we found that 30mins struck the best balance between the granularity of traffic states and the number of observed pathlets.

Given pathlet r with overlapping neighborhood $\{o_1, \ldots, o_s\}$, we define the feature vector of frame f_i, $M(r)_i$ as an s-dimensional vector consisting of observed congestion states from pathlet r's neighborhood $OL(r)$ during frame f_i, i.e., $M(r)_i = [\hat{d}_{f_i}(o_1) \ \ldots \ \hat{d}_{f_i}(o_s)]$. We stack feature vectors that contain at least two non-missing values into a single feature matrix $M(r)$. Let N be the number of such "partially observed" feature vectors. Matrix $M(r)$ has the following structure:

$$M(r) = \begin{bmatrix} M(r)_1 \\ \vdots \\ M(r)_N \end{bmatrix}.$$

7.2 Congestion Feature Clustering with Temporal Constraint

Having obtained matrix $M(r)$, which represents the dynamic congestion state of $OL(r)$, we first apply k-POD (Chi et al. 2016), an iterative k-mean-based clustering algorithm to cluster rows of $M(r)$ into k groups and fill in missing values in $M(r)$. Unlike other methods dealing with missing data, such as deletion and imputation, k-POD works well with unknown missing mechanisms and high missing rates.

Given k cluster centroids $c_1, \ldots, c_k$ computed using k-POD, we initialize missing values in $M(r)$ by finding the nearest centroid for each row in $M(r)$. Since the feature matrix size N varies a lot among different pathlets (e.g., from 5 to over 4,000 frames), it is important that we select cluster size k adaptively. We therefore adopt the Gaussian-Means (G-Means) method to find the optimal k. It works by starting with many small clusters, then recursively merging them into larger clusters if two clusters are sampled from the same Gaussian distribution (Hamerly and Elkan 2003).

Next, we refine the initial clustering result by introducing temporal relationships in congestion features.

Graph label optimization. We formulate our problem through k-means with Laplacian smoothing. Given feature matrix $M(r)$ and an affinity matrix W that represents the temporal consistency between any two frames in $M(r)$, we want to find k cluster centroids that minimize their distances to the observations while determining the soft assignment between a frame and one of the k clusters at the same time.

Formally speaking, let X be the $N \times k$ cluster assignment matrix. Each row x_i of X is a binary vector such that $x_i(j) = 1$ if frame f_i is assigned to cluster j or 0 otherwise. The k cluster centers

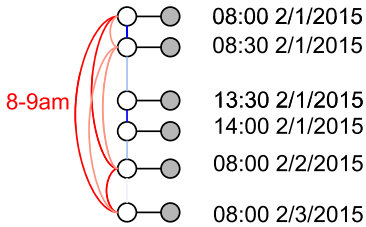

Fig. 10. Illustration of a weighted consistency graph of congestion patterns in observed frames (white nodes). Shaded nodes represent the observation in each frame labeled by its starting time. Red and blue edges represent the time-of-day similarity and the similarity between nearby frames. The opacity of an edge represents its weight.

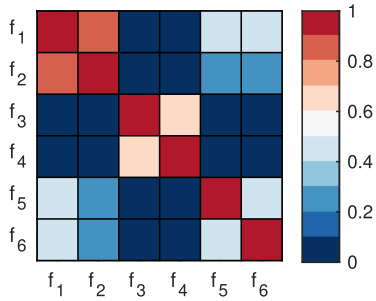

Fig. 11. Visualization of W, the adjacency matrix of the weighted consistency graph in Figure 10. In practice, W is mostly sparse.

are stored as row vectors in $k \times s$ matrix C. We initialize X and C using the results from Section 7.2, then find their optimal values by solving the following minimization problem:

$$\begin{aligned} &\underset{X,C}{\text{minimize}} ||XC - M(r)||_F^2 + \gamma Tr(X^T L X), \\ &\text{s.t. } X\mathbf{1} = \mathbf{1}, X = \{0,1\}, 0 \le C \le 1. \end{aligned} \tag{2}$$

L is the graph Laplacian matrix, obtained as $L = D - W$, where $D = diag(\sum_{j=1}^{n} w_{i,j})$ is the degree matrix of the weighted graph defined by adjacency matrix W. Constant coefficient γ controls the weight of temporal consistency in the clustering process. Figure 10 illustrates a simple example of how pairwise consistencies are defined among frames. We relax the integer constraint on X to be a real matrix of values $[0, 1]$. The resulting function can be solved using alternating direction optimization.

Formulation of W. The first type of temporal relationship is the similarity between adjacent frames, i.e., it would be less likely for local traffic to transition from free-flow to a fully congested state between two frames in consecutive time steps. Therefore, we define the **smoothness weight** between the ith frame and the jth frame as an exponential decay function:

$$C_{smooth}(i,j) = \exp\left(-\frac{(t_i - t_j)^2}{\sigma_{smooth}^2}\right),$$

where t_i and t_j are the starting times of frames f_i and f_j. σ_{smooth} is a small constant.

The second type of temporal relationship is the similarity based on the periodical nature of traffic flow in urban cities. In particular, we focus on the starting time-of-day (TOD) of a frame and whether or not it starts on a weekday or a weekend. We discretize starting time-of-day of each frame by a fixed window of ω frames and call it h_i, the *TOD identifier* of frame f_i. We define the **TOD weight** between frames f_i and f_j as

$$C_{tod}(i,j) = \exp\left(-\frac{\min\{|h_i - h_j|, h_{max} - |h_i - h_j|\}^2}{\sigma_{tod}^2}\right).$$

Edge weight $w(i, j)$ is computed as a linear combination of C_{tod} and C_{smooth}, i.e.,

$$w_{i,j} = \theta C_{smooth}(i, j) + (1 - \theta)C_{TOD}(i, j)$$

$w_{i,j} = 1$ if frame i and frame j are in the same time step. By default, $\sigma_{tod} = \sqrt{2}$ and $\sigma_{smooth} = 2$. The coefficient of the smoothness weight θ is chosen to be 0.5. Figure 11 shows the weight matrix W associated with the consistency graph in Figure 10.

8 TRAVEL TIME PREDICTION

This section describes the online step in path travel time prediction. We will explain how to predict path travel time using local congestion patterns we learned from historical data. Several variations of our algorithm will be discussed.

8.1 Prediction Using Historical Data

We first introduce a baseline prediction method that only relies on historical trajectories. Initially, query path P is decomposed into m pathlets $r_1, \ldots, r_m$. We proceed to compute the time-dependent historical travel time of each pathlet if the departure time is t.

Let t_i be the time when the predicted trip enters r_i, the ith pathlet in P. We compute the historical travel time of r_i at t_i, $d^H_{t_i}(r_i)$ as the median of all travel time observations in the same time-of-day interval as t_i. If pathlet r_i has no travel time observation during the given time interval, then we take the median over all available observations on r_i. For pathlets not traversed by any trajectories, speed limit and road geometry information are used to derive a travel time estimation.

The historical travel time of path P, $d^H_t(P)$ is computed recursively. Let P_i denote the sub-path of the first i pathlets $r_1, \ldots, r_i$ in P, such that $P_1 = \langle r_1 \rangle$ and $P_m = P$. Initially, set $t_1 = t$ and $d^H_t(P_0) = 0$. Then for all $i = 1, \ldots, m$, we update $d^H_t(P_i)$ and t_i using the following formulas:

$$d^H_t(P_i) = d^H_t(P_{i-1}) + d^H_{t_i}(r_i),$$
$$t_{i+1} = t_i + d^H_{t_i}(r_i).$$

The travel time of P is $d^H_t(P) = d^H_t(P_m)$.

8.2 Prediction Exploiting Current Observations

8.2.1 Congestion Pattern Matching. We use pattern matching to incorporate recent congestion states observed by R_β into travel time predictions. We will first introduce the prediction process for all drivers, then discuss the additional steps needed for personalized prediction.

Algorithm 1 summarizes our approach for travel time prediction without using driver identities. On Line 1, we test if pathlet r is observed by R_β. Specifically, we define the **inverse overlapping neighborhood** of pathlet r to be the set of all pathlets whose neighborhoods contain r:

$$OL^{-1}(r) = \{o \in PD | r \in OL(o)\}.$$

We say the neighborhood of some pathlet o, $OL(o)$ **is observed by** R_β if at least one pathlet in $OL(o)$ has been traversed by one of the trajectories in R_β. Then **pathlet r is observed by** R_β if there exists at least one pathlet $o \in OL^{-1}(r)$ such that its neighborhood $OL(o)$ is observed by R_β.

When r is observed by R_β, we predict the travel time of pathlet r using `PredictByPM` (Line 2), i.e., for each pathlet $o_l \in OL^{-1}(r)$ such that $OL(o_l)$ is observed in R_β, we identify the current congestion pattern of $OL(o_l)$ by matching recent observations in R_β against congestion patterns learned in Section 7. Pattern matching is done using least square, i.e., let matrix C_{o_l} represent k congestion patterns over neighborhood $OL(o_l)$, and $M^R(o_l)$ represent the most recently observed congestion

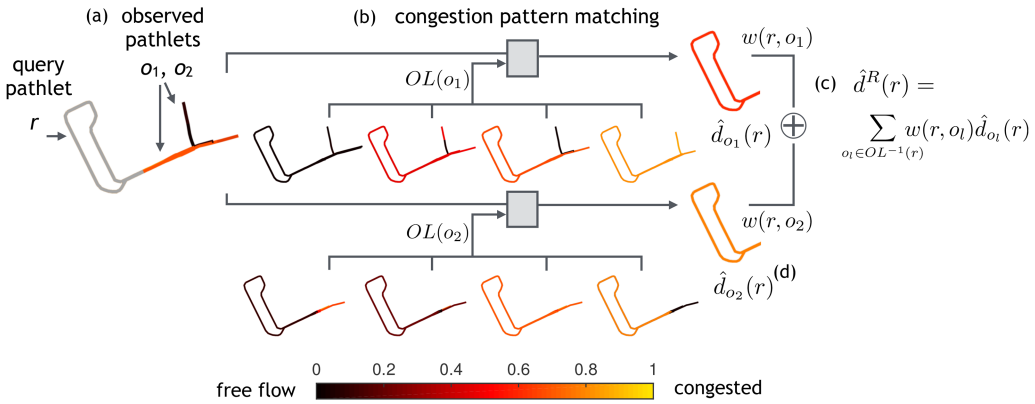

Fig. 12. Illustration of how PredictPM computes $\hat{d}^R(r)$, the congestion status of pathlet r, observed by R_β. (a) The query pathlet r to be estimated and observed pathlets in r's inverse neighborhood $OL^{-1}(r) = \{o_1, o_2\}$; (b) For each neighborhood $OL(o_l)$, congestion pattern matching outputs a congestion state $\hat{d}_{o_1}(r)$ for pathlet r; (c) The aggregated congestion state of r.

ALGORITHM 1: Algorithm for predicting travel time of P using pattern matching.

Input: Query path $P = \langle r_1, \ldots, r_m \rangle$, recent trajectories R_β, current (departure) time t
Transformation function $cdf(D(r), \cdot)$ from travel time to congestion state for each dictionary pathlet r
Output: Predicted travel time of path P: $d_t(P)$
Parameters: Time when user enters first pathlet r_1 of P: $t_1 = t$
for $i = 1 \ldots m$ **do**
  if r_i *is observed by* R_β **then**
    $\hat{d}^R(r_i) \leftarrow$ `PredictByPM(`r_i, R_β`)`
    $t_{i+1} = t_i + cdf^{-1}\left(D(r_i), \hat{d}^R_{t_i}(r_i)\right)$
  else
    $\hat{d}^R(r_i) \leftarrow cdf(D(r_i), d^H_{t_i}(r_i))$
    $t_{i+1} = t_i + d^H_{t_i}(r_i)$
  end
end
$d^R(r_i) = cdf^{-1}(D(r_i), \hat{d}^R(r_i))$
$d_t(P) \leftarrow \sum_i^m d^R(r_i)$

states in that neighborhood. We solve the following optimization problem:

$$x^* = \arg\min_x ||C_{o_l} x - M^R(o_l)||^2, \quad s.t.\ |x| = 1, 0 \le x_i \le 1 \text{ for all } i = 1, \ldots, k. \tag{3}$$

Solution x^* results in one prediction of the congestion state of pathlet r, $\hat{d}^R_{o_l}(r) = C_{o_l} x^*$. We then aggregate predictions from multiple neighborhoods via weighted average, $\hat{d}^R(r) = \sum_{o \in OL^{-1}(r)} w(r, o_l)\hat{d}^R_{o_l}(r)$, where the weight function $w(r, o_l)$ is the correlation coefficient between the congestion states of r and that of o_l in feature matrix $M(o_l)$. Figure 12 shows an example of this procedure.

If r is not observed, then we compute the time-dependent historical travel time of r instead (Line 4). After converting the predicted values to absolute travel time using the inverse probability integral transform, we compute the total path time as the sum of all pathlet travel time (Lines 6–7).

8.2.2 Optimizations for Prediction. The pattern-matching method presented so far has a few potential issues. First, function `PredictByPM` assumes that the traffic status of the road network observed from R_β does not change over the predicted trip duration. For longer trips, however, we need to take into account that the predictive power of R_β decays over time. Second, computing path travel time as a sum of pathlet travel times that are predicted independently does not consider the transition of congestion status between adjacent pathlets in the query path. We attempt to address these issues using the following heuristics.

Hybrid prediction. We replace Line 7 by a hybrid model where pathlet travel time is computed as a linear combination of `PredictByPM` and the historical prediction. By decreasing the coefficient on the former term for each subsequent pathlet, we can model the decaying predictive power of R_β as we predict further into the future.

Technically, consider a decreasing sequence $\alpha_1 \geq \alpha_2 \geq \cdots \geq \alpha_m$ such that $0 \leq \alpha_i \leq 1$ for all i. We define the *hybrid travel time prediction* of path P as follows:

$$d_t^R(P) = \sum_i^m d_{t_i}^E(r_i) = \sum_i^m \alpha_i d^R(r_i) + (1 - \alpha_i) d_{t_i}^H(r_i).$$

We use a heuristic approach to determine the optimal value of coefficients $a_1, \ldots, a_m$. Initially, assign $a_1, \ldots, a_m$ to a constant value a_0. The value of a_i should decrease as the fraction of travel time spent on the first i pathlets increases. We solve the following system of $2m$ equations iteratively:

$$d_{t_i}^E(r_i) = \alpha_i d^R(r_i) + (1 - \alpha_i) d_{t_i}^H(r_i),$$
$$a_i = 1 - \frac{\sum_{j=1}^i d_{t_j}^E(r_j)}{\sum_{j=1}^m d_{t_j}^E(r_j)} \quad \text{for all } i = 1, \ldots, m.$$

The initial value a_0 can be learned from data, e.g., $a_0 = 0.8$ is a good choice for our test dataset.

Adjacent pathlet smoothing. This optimization is motivated by the intuition that abrupt changes of congestion status tend to happen near (1) intersections or (2) on a road segment connecting to another road of different classes, as in the case of highway exits. To reduce the impact of outliers on pathlet travel time prediction, we apply a rule-based smoothness constraint on the independently predicted congestion status of each pathlet r_i.

Let x_i be the hybrid travel time prediction of pathlet $r_i \in P$ using *adjacent pathlet smoothing*. Let l_i be the length of pathlet r_i. We find the optimal values that minimize the following optimization problem:

$$\min_{x_1, \ldots, x_m} \sum_{i=1}^m (x_i - d_{t_i}^E(r_i))^2 + \mu \sum_{i=1}^{m-1} b(i, i+1, P) \left| \frac{x_i}{l_i} - \frac{x_{i+1}}{l_{i+1}} \right|.$$

The first term in the objective function measures the cost of perturbing the hybrid prediction of pathlet r_i, $d_{t_i}^E(r_i)$. The second term measures the cost of changing average speed on adjacent pathlets $r_i, r_{i+1} \in P$. The weight on each pair of adjacent pathlets is defined by the following function:

$$b(i, i+1, P) = \texttt{NotAJunction}(r_i, r_{i+1}) + \texttt{SameRoadClass}(r_i, r_{i+1}).$$

`NotAJunction` and `SameRoadClass` are indicator functions that decide whether the last link of r_i and the first link of r_{i+1} do not meet at a road intersection (i.e., a vertex of degree greater

than 2) and if they share the same road class. The choice of Parameter μ is learned from data via cross-validation.

8.3 Personalized Prediction

To predict path travel time for a particular user, we need to determine which driver group the user belongs to at query time. Then, we can use the travel time distribution for that driver group to transform the estimated congestion state of each pathlet back to a travel time prediction for that particular user.

Algorithm 2 describes the details of prediction for a given user u_q. First, we find the driver group that u_q belongs to based on the available travel time observations from u_q's recent trajectory history (Line 1). Let $f(u_q)$ be the feature vector of u_q as defined in Section 6.3. $f(u_q)$ would contain many missing values if the history is short. Procedure `MatchDriverGroup` estimates the nearest driver group while ignoring the missing pathlets. The distance function between query user u_q and driver group G_j is defined as follows:

$$d(u_q, G_j) = \frac{1}{|G_j|} \sum_{u \in G_j} \frac{1}{|x|} \sum_{i=1}^{|PD|} (f(u_q)_i - f(u)_i)^2 x_i,$$

where $x_i = 1$ if the ith pathlet in the dictionary is observed in u_q's recent history and $x_i = 0$ otherwise.

The computation of pathlet travel time is similar to that in Algorithm 1. The main difference is replacing the inverse probability integral functions $cdf^{-1}(D(r), \cdot)$ with ones that are defined for the matched driver group $cdf^{-1}(D(r, G_j), \cdot)$ on Lines 4 and 6. The optimization techniques proposed in Section 8.2.2 can also be applied to personalized prediction.

In practice, we also need to handle the rare scenario when none of the drivers in the historical data has a similar travel time profile to the query user's. Such an event can be detected by checking

ALGORITHM 2: Algorithm for predicting travel time of P for user u_q using pattern matching.

Input: Query path: $P = \langle r_1, \ldots, r_m \rangle$ of user u_q, recent trajectories R_β, current (departure) time t

Transformation function $cdf(D(r, G_i), \cdot)$ from travel time to congestion state for each dictionary pathlet r and each driver group G_i, $(1 \le i \le K)$.

Output: Predicted travel time of path P for user u_q: $d_t(P, u_q)$

Parameters: Time when user enters first pathlet r_1 of P: $t_1 = t$

1 $G_j \leftarrow$ `MatchDriverGroup`(R_β, u_q)
2 **for** $i = 1 \ldots m$ **do**
3 **if** r_i *is observed by* R_β **then**
4 $\hat{d}^R(r_i) \leftarrow$ `PredictByPM`(r_i, R_β)
5 $t_{i+1} = t_i + cdf^{-1}(D(r_i, G_j), \hat{d}^R_{t_i}(r_i))$
  else
6 $\hat{d}^R(r_i) \leftarrow cdf(D(r_i, G_j), d^H_{t_i}(r_i))$
7 $t_{i+1} = t_i + d^H_{t_i}(r_i)$
  end
end
8 $d^R(r_i) = cdf^{-1}(D(r_i, G_j), \hat{d}^R(r_i))$ for all $i = 1, \ldots, m$
9 $d_t(P, u_q) \leftarrow \sum_i^m d^R(r_i)$

whether the optimal distance computed by `MatchDriverGroup` is higher than a threshold, such as the average intra-cluster distance. We then resort to the non-personalized approach in Algorithm 1.

9 RESULTS

9.1 Experimental Data

We evaluate our algorithms using GPS trajectories of electric taxis deployed by Gotcha II (Xu et al. 2014), a two-stage project that ran from 2014 to 2017 in Shenzhen, China.[2] The Stage I dataset contains trajectories of 15 vehicles from August 2014 to November 2015, with an average sampling rate of 10s per sample. Driver identities are not available in this dataset, since vehicle IDs are coded differently each month. We use this dataset for experiments on non-personalized travel time prediction (Section 9.3).

The Stage II dataset contains trajectories of 25 vehicles from December 2016 to November 2017, with complete driver identity information. The sampling rates for these more recent trajectories are between 2–5s per sample. They are used for experiments on personalized travel time prediction (Sections 9.4 and 9.5).

We selected four urban districts in Shenzhen for experiments, labeled Regions 1–4 (Figure 13). Region 1 is the largest of all, with Shenzhen International Airport to the northwest. Regions 2–4 cover the densely populated downtown areas of the city. Table 1 summarizes the characteristics of the regions for both datasets. To illustrate the sparsity of concurrent observations, we show in the last column of Table 1 the percentage of links traversed within a 30-minute interval on the morning of November 2, 2014. Despite being taken from a weekday morning, on average only 3.32% of the road links in the test region are observed. These observed road links are highlighted as red dashed lines on the regional maps in Figure 13. Although the map of Region 1 contains a large number of links, as rider demand is lower than the downtown area, only a small number of taxi trips are observed in this region. Hence, it has the lowest coverage of traffic observations.

The Open Street Map of Shenzhen is used for map matching (osm 2015).[3] Road class information of each link (e.g., *motorway, primary, secondary, and tertiary* road segments and links) is extracted from the OSM tags in the map data.

Figure 13 also visualizes the map coverage of the optimal pathlet dictionary, computed using the approximation method mentioned in Chen et al. (2013b) with a minimum of five travel time observations on each pathlet. In Table 2, we show the detailed coverage for individual road classes based on the Open Street Map annotation.[4] The total coverage rate ranges between 30% to 70%. This is partially affected by the fact that the road network is larger in size than the trajectory bounding box to ensure successful map matching. Region 1 has the lowest coverage for its remoteness from the inner city. For Regions 2, 3, and 4, the coverage on non-residential roads is mostly above 85%. Roads labeled residential and others (e.g., including service roads and other non-standard road types) are less covered, as taxis are more likely to travel on major roads.

In each test region, we randomly selected 300 trajectories collected during the last six months in the dataset as the test data. The rest of the trajectories are considered training data for learning congestion patterns. This allows each test query to have at least five months of historical data to rely on. We use the actual trip duration of each test trajectory as the prediction ground truth.

[2]Shenzhen is a major city in southern China.

[3]The OpenStreetMap map used for processing the 2014–2015 dataset was retrieved in 2016. The map for processing the 2016–2017 dataset was retrieved in 2018.

[4]Road classes are defined based on the "highway" tag of the OSM ways. Classes *motorway*, *trunk*, *primary*, *secondary,* and *tertiary* include both the labeled ways and their associated links. The *other* class includes *service road*, *construction,* and other unclassified ways.

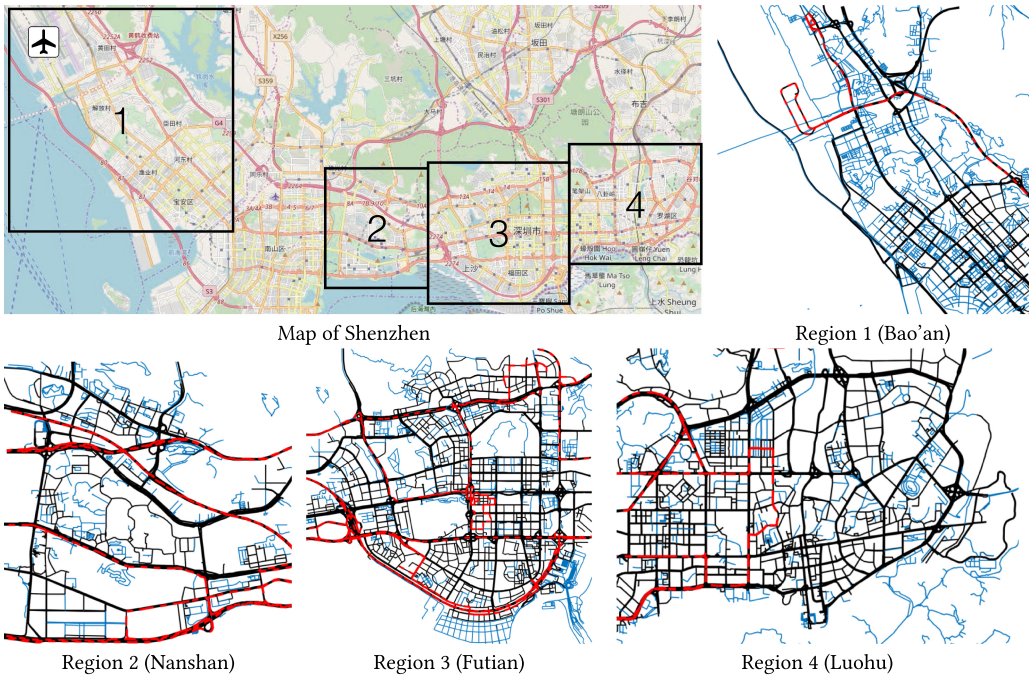

Fig. 13. Top left: Map of Shenzhen, China, with test regions annotated. Top right and bottom: Road networks for Regions 1–4. Black edges indicate road segments covered by the pathlet dictionary of the '14 dataset. Red dashed lines highlight road links visited by at least one probe vehicle from 7:00am to 7:30am on November 2, 2014. Segments not covered by the pathlet dictionary are represented as blue lines.

Table 1. Map and Trajectory Statistics of Test Regions During 2014–2015 ('14) and 2016–2017 ('16)

Region ID	Name	Area (km^2)	# of links ('14)	# of links ('16)	# of trajectories ('14)	# of trajectories ('16)	dictionary size ('14)	dictionary size ('16)	7-7:30am coverage
1	Bao'an	138.9	7,649	11,558	11,373	7,265	3,438	5,527	0.86%
2	Nanshan	29.2	2,991	4,728	33,859	31,680	11,363	7,726	5.35%
3	Futian	60.3	7,574	9,173	60,111	58,185	47,898	22,201	5.40%
4	Luohu	44.7	5,651	7,489	20,378	36,103	32,589	14,671	2.80%

Table 2. Road Link Coverage of the Pathlet Dictionary of the 2014–2015 Dataset

Link class	Region 1 (Baoan)		Region 2 (Nanshan)		Region 3 (Futian)		Region 4 (Luohu)	
	# of links	coverage	# of links	coverage	# of links	coverage	# of links	coverage
motorway	148	0.74	27	1.00	95	1.00	18	0.72
trunk	650	0.86	313	0.98	786	0.96	438	0.98
primary	428	0.79	259	0.96	749	0.99	623	0.97
secondary	664	0.70	102	0.99	563	0.89	649	0.97
tertiary	1,078	0.42	207	0.89	1,107	0.85	885	0.91
residential	2,268	0.06	1,264	0.49	2,767	0.58	1,880	0.59
others	2,413	0.08	819	0.28	1,507	0.33	1,158	0.33
total	7,649	0.30	2,991	0.57	7,574	0.68	5,651	0.70

Table 3. Regional Travel Time Prediction Results

Region ID	ρ^R	Baseline		PredictPM++		% improved
		MRE	MAE (min)	MRE	MAE(min)	
1	0.218	0.213	6.739	**0.148**	4.700	30.3%
2	0.124	0.155	2.959	0.151	2.891	2.43%
3	0.556	0.198	7.357	0.185	6.654	6.33%
4	0.628	0.238	8.472	0.201	7.310	15.36%
Mean	0.381	0.201	6.382	0.171	5.389	13.60%

Trip duration is bounded between 40–4,500s. Tested on a Dell T3600 workstation with 3.2GHz processors and 32GB RAM, the offline step takes approximately 4h per region. The average query processing time is 0.7s.

We evaluated non-personalized travel time prediction (Algorithm 1) and its variations against the baseline method that uses only historical data. The first one, `PredictPM` is the same as Algorithm 1. We refer to the variation with hybrid travel time prediction, described in Section 8.2.2 as `PredictPM+`; and refer to the variation that uses both hybrid prediction and adjacent pathlet smoothing as `PredictPM++`. Finally, we refer to the combination of personalized travel time prediction (Algorithm 2) and the optimizations as `PersonalPM++`.

9.2 Evaluation Metrics

Our first evaluation metric is the *observation rate* ρ^R, which measures the percentage of trajectories that contain at least one pathlet observed by R_β. For trajectories that do not contain any observed pathlet, their travel time prediction will be the same as the baseline. We will only use observed trajectories to evaluate prediction accuracy, as detailed below.

Let d^i_{pred} and d^i_{true} be the predicted travel time and the actual travel time of the ith test paths. We use two metrics to evaluate prediction accuracy over N test queries: the *Mean Absolute Error* (MAE) and the *Mean Relative Error* (MRE) (Yuan et al. 2010):

$$MAE = \sum_{i=1}^{N} \frac{|d^i_{pred} - d^i_{true}|}{N}, \qquad MRE = \sum_{i=1}^{N} \frac{|d^i_{pred} - d^i_{true}|}{\sum d^i_{true}}.$$

We typically prefer MRE when comparing the performance of two problem instances, since MAE is sensitive to the trip duration.

9.3 Non-personalized Travel Time Prediction Results

We compare the overall travel time prediction results in different test regions using the 2014–2015 data (Table 3). The observation rate ρ^R ranges from 12.4% in Region 2 to 62.8% in Region 3. On average, our algorithm can infer recent-time traffic status from either direct observations or indirect observations from overlapping pathlets for 38.1% of the test queries. The observation rate in Regions 3 and 4 is much higher than the other two because of higher route diversity and higher taxi demand.

Using (`PredictPM++`), the average MRE is 0.171, and the average MAE is 5.4mins. The lowest MRE is 0.148 in Region 1 (BaoAn), which is 30.3% lower than the baseline method. This is mainly because Region 1 has the strongest path regularity, as reflected by the small dictionary size in Table 1. Most trajectories in this region are either going to or coming from the airport, located at the northwest corner of the map. However, the prediction accuracy of Region 4 is the worst among the regions. Since Region 4 (Luo hu) is known as the shopping and nightlife district in Shenzhen,

Table 4. Prediction Accuracy of Baseline, PredictPM++, and CATD-OC

Region ID	success rate	CATD-OC		Baseline		PredictPM++	
		MRE	MAE	MRE	MAE	MRE	MAE
1	42.86%	0.2830	506.2	0.1963	343.6	0.1828	320.0
3	24.75%	0.2135	456.0	0.2065	441.1	0.2023	432.2

it has a denser street layout that contributes to higher route diversity. The GPS noise in this region is also more severe. This would negatively impact the preprocessing step and the computation of travel time observations.

Comparison with CATD-OC. We compared the baseline and PredictPM++ with CATD-OC, a popular path-based travel time-estimation algorithm from Wang et al. (2014b). To adapt CATD-OC under our problem setting, we introduce a few modifications to the algorithm in the original paper. First, we assume each trajectory is from a unique driver, since driver identities are not available in the 2014–2015 dataset. Second, the geographical features of road segments do not include their point of interest (POI) distributions, as POIs are not available in our datasets. Each time slice in the travel time tensor is 15mins long and the total number of time slices used in the tensor decomposition process is 8 (2h).

Table 4 compares the evaluation results between our methods and CATD-OC for Region 1 and Region 3. Note that due to the small number of travel time observations in one time slice, the travel time tensor, especially the portion representing real-time traffic, is extremely sparse. As a result, context-aware tensor decomposition fails to find valid core tensors in some test queries. The percentage of successful queries is reported in the *success rate* column in Table 4. Only test queries with successful tensor decomposition are used for evaluating all three methods.[5] The comparison results show that both Baseline and PredictPM++ out-perform CATD-OC on the test data.[6] It demonstrates that our approach is more robust against sparsity in real-time data and can handle large bias in travel time observations.

Effect of window size β. Figure 14 plots the MRE of test trajectories in Region 1 given recent trajectories of different window size β and using different algorithm variations. When β is small, we have fewer observations from recent trajectories. However, if β is too large, then the traffic observations may become out of date, so they could not be used to predict future travel time. In most test regions, we found $\beta = 1.5$h is the most stable choice for achieving good prediction accuracy. For the comparison among algorithm variations, we see that pattern matching using recent trajectories contributes to most of the improvement from the baseline method, especially when β is small. The two optimization heuristics introduced in Section 8.2.2 have a small but consistent improvement on the prediction accuracy.

Effect of dictionary size. We evaluate the impact of pathlet dictionary size on prediction error using trajectories in Region 1. Figure 15(a) shows how parameter λ, introduced in Section 6.1, allows us to control the dictionary size and the average trajectory decomposition size; Figure 15(b) plots the test MRE and the observation rate against λ. The lowest MRE (0.148) is achieved when $\lambda = 0.001$. We can see that, in general, the smaller the dictionary, the smaller the MRE and the

[5]Region 2 and Region 4 are omitted, since the success rate of tensor decomposition is very low in those areas.

[6]The MRE and MAE for PredictPM++ is much closer to Baseline than the results in Table 2, since some of the test paths used in this experiment are neither directly nor indirectly observed by the recent trajectories. In such cases, PredictPM++ predicts the travel time using the baseline approach.

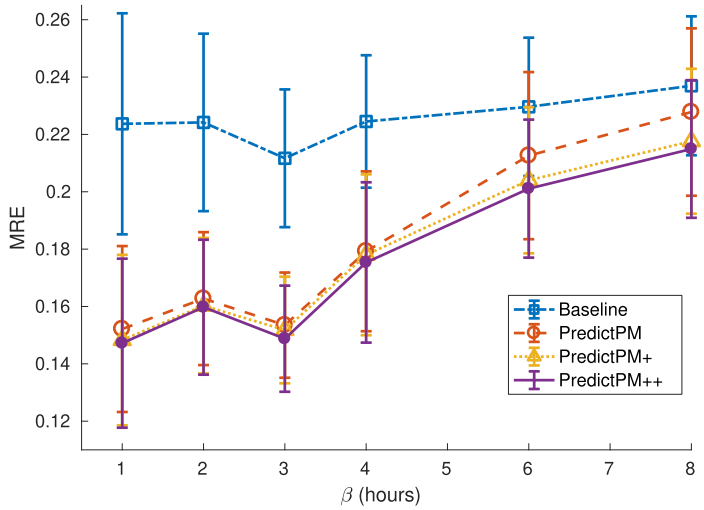

Fig. 14. Comparison of prediction accuracy between the baseline and variations of our algorithm. The error bars indicate the standard error in each experiment.

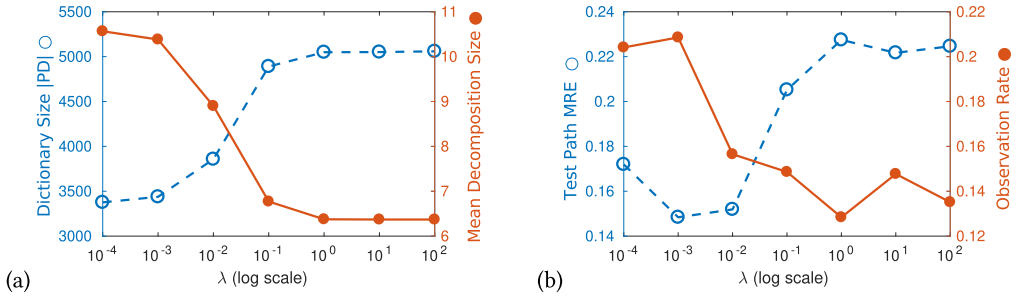

Fig. 15. (a) Effect of λ on dictionary size $|PD|$ and average decomposition size. (b) Effect of λ on MRE and the observation ratio ρ^R.

higher the observation rate. Though making the dictionary too small (e.g. $\lambda = 10^{-4}$) decreases the average pathlet length, thus losing the advantage of a path-based method.

Error analysis. We analyze the relative prediction error $(d^i_{pred} - d^i_{true})/d^i_{true}$ for the 32 observed weekday test queries in Region 1. Three prediction methods, `PredictPM++`, `Baseline`, and `CATD-OC` are tested. For `CATD-OC`, we use a heuristic to handle failure cases in the context-aware tensor decomposition process.[7] We found that for 62.5% of the test queries, `PredictPM++` outperforms the baseline. Half of those have made an improvement of more than 10%. In contrast, `CATD-OC` outperforms the baseline on 43.8% of the test queries.

Figures 16(a) and 16(b) plot the relative errors with respect to trip distance and trip duration. We observe a negative correlation between trip duration and relative prediction error. The correlation coefficients are $r_{\texttt{baseline}} = -0.522$, $r_{\texttt{CATD-OD}} = -0.591$ and $r_{\texttt{PredictPM++}} = -0.412$ for the three methods in comparison. However, the correlation between relative error and trip distance is less salient

[7]We identify and remove indices in the input tensor that cause tensor decomposition failure, then use historical average to estimate the travel time of the removed indices.

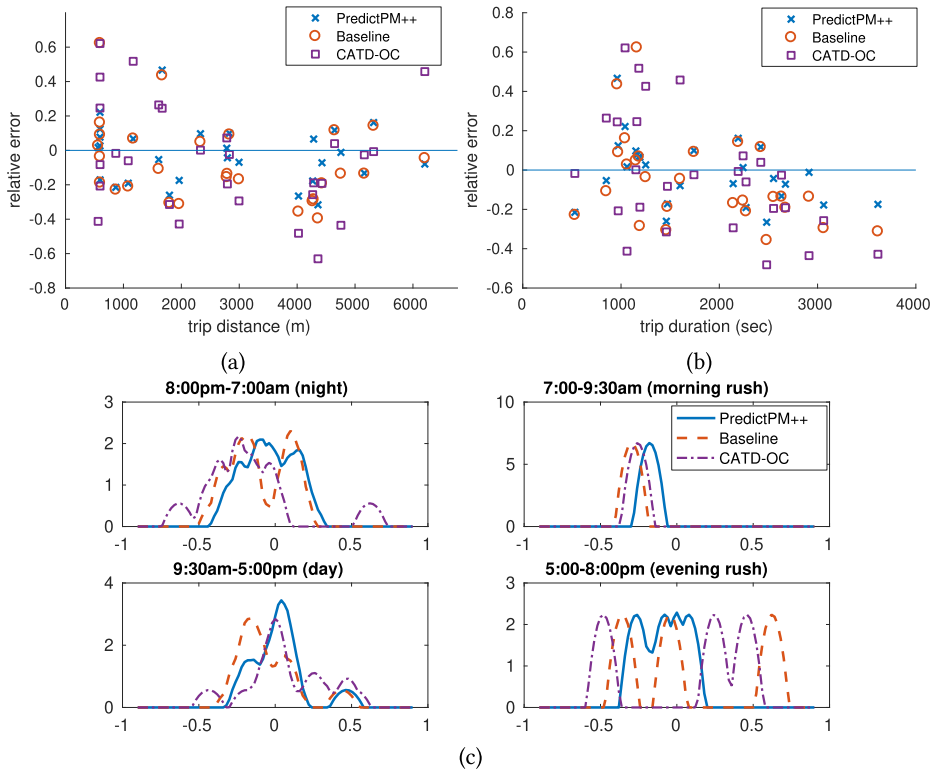

Fig. 16. Visualizations of prediction errors for weekday travel time queries in Region 1 using PredictPM++ (blue), the baseline method (orange) and CATD-OD (yellow). (a) Relative prediction error vs. trip distance. (b) Relative error vs. trip duration. (c) Distribution of prediction error for trips that depart within four different periods of a day.

($r_{\text{baseline}} = -0.292$, $r_{\text{CATD-OD}} = -0.261$, $r_{\text{PredictPM++}} = -0.116$). This indicates that some trips have longer delays than usual and PredictPM++ is the best among the three methods in predicting those unexpected delays.

Figure 16(c) plots the distribution of prediction errors at four time-of-day intervals, separated by the morning rush hour (7:30–9:30am) and the evening rush hour (5:00–8:00pm). During rush hour, PredictPM++ has better prediction accuracy improvement than the other two methods.

9.4 Personalized Travel Time Prediction Results

In this section, we consider two scenarios to evaluate the performance of personal travel time prediction (PersonalPM++). The first scenario is when the query user's historical trajectories are within the training set. In this case, the driver group that the user belongs to is already known at query time. The other scenario is when the query user is unknown. We need to use the approximation method presented in Section 8.3 to determine the closest driver group based on recent trajectories of that user.

The results using the 2016–2017 dataset are summarized in Table 5 and Figure 17. As this dataset contains more drivers than the 2014–2015 dataset used in previous experiments, non-personalized algorithm PredictPM++'s performance is not as good as in previous experiments. However, by incorporating driver information, the average MRE becomes much lower: 0.177 and 0.163 for the two scenarios.

Table 5. Travel Time Prediction Results Using PredictPM++ and PersonalPM++

Region ID	PredictPM++		PersonalPM++			
	No driver information		Driver in training data		Driver not in training data	
	MRE	MAE (sec)	MRE	MAE (sec)	MRE	MAE (sec)
1	0.139	194.3	0.113	168.4	0.135	156.6
2	0.198	179.7	0.188	215.1	0.183	188.3
3	0.264	289.1	0.203	278.2	0.172	224.9
4	0.209	292.7	0.205	214.2	0.162	257.3
Mean	0.203	238.9	0.177	219.0	0.163	206.8

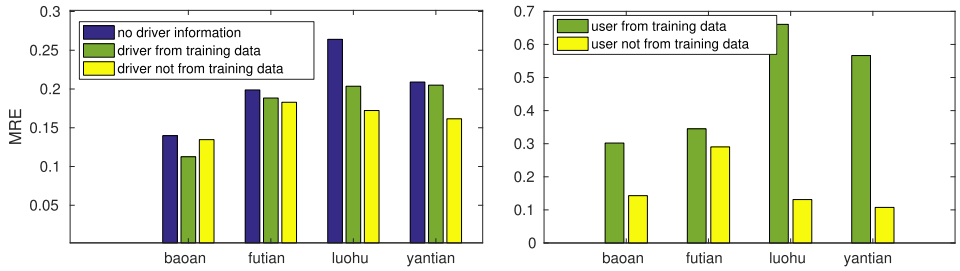

Fig. 17. (a) Travel time prediction results using drivers' information. (b) Observation rate ρ^R for each region for personalized prediction.

For Region 1, traffic patterns are more regularized, incorporating driver information results in an MRE of 0.113 and MAE of 2.8mins in the first scenario. In the second scenario, the error is slightly larger. Nevertheless, personalized prediction still achieved a 2.9% reduction in the mean relative error compared to non-personalized prediction.

For the other three regions, while incorporating driver information always has a positive effect on prediction accuracy, the first scenario results in larger prediction error than the second. There could be many possible reasons for this behavior. For instance, a driver's driving style may change over time, or the vehicle changed its driver during the one-year study. In addition, Figure 17(b) compares the observation rates of the two scenarios. We can clearly see that the observation rate is lower when the driver identity is unknown, as our experiment excludes users who cannot be confidently assigned to any driver groups based on their recent trajectories.

9.5 Number of Probe Vehicles in Personalized Travel Time Prediction

Finally, we analyze the performance of our algorithm with a different number of probe vehicles using Region 1 data. Here, we consider the second, more challenging scenario when the query users are not the same as the users in the historical data. The test data consists of 300 query paths of five drivers and their recent trajectories ($\beta = 1.5hr$). The training data for each test case contains historical trajectories from 8–20 unique drivers.

Figure 18 plots the MRE of PersonalPM++ and the baseline approach against the number of drivers in the training data. We observe that the MRE in general decreases as the number of drivers increases, which is consistent with the intuition that having more data helps prediction. It is worth noting that in our previous work (Li et al. 2017), which predicts travel time without driver information, prediction may be worse when the number of drivers increases due to large variations

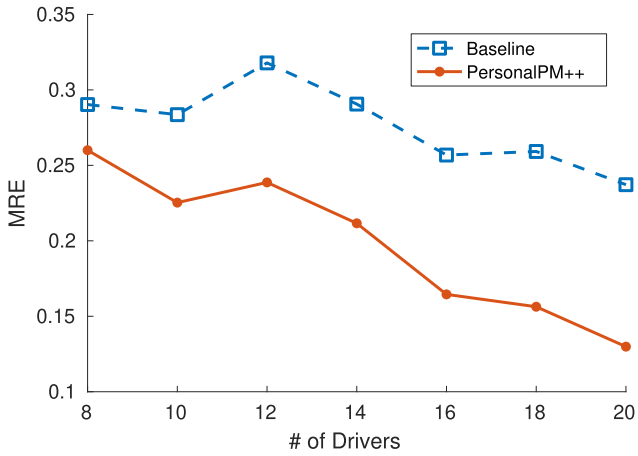

Fig. 18. MRE of PersonalPM++ (orange) and the baseline (blue) on Region 1 with a different number of drivers in the training data. Drivers in the test data do not appear in the training data.

in personal driving behaviors. The results in this experiment demonstrate that clustering drivers based on their travel time history can indeed leverage the variations in drivers' speed profiles, making the pathlet-based approach more scalable to a larger driver pool.

10 DISCUSSIONS AND FUTURE WORKS

While the real-time collection of GPS trajectories from taxis and mobile users are common today, a practical solution for trajectory-based travel time prediction needs to be robust to the situation of only having access to a small number of active mobile probes. This article presented an algorithm framework for predicting path travel time from GPS trajectories, under the scenario of 10–25 GPS-equipped vehicles and no trip labels. In the offline stage, it finds a pathlet dictionary that represents the frequently shared paths and learns congestion patterns from sparse pathlet travel time observations. In the query step, it identifies the current congestion pattern of relevant pathlets from recent trajectories, then infers the travel time of the query path from the identified pattern and the historical travel time. Driver information can also be incorporated to provide personalized prediction based on a driver's partial driving history. Experiments on taxi trajectories from Shenzhen, China, result in higher accuracy than the baseline approach of using only historical trajectories, as well as a state-of-the-art travel time prediction method that uses both historical trajectories and real-time trajectories.

This work has also demonstrated that regular patterns in travel time observations can help extrapolate traffic status information from incomplete data. As shown in the experiment, regions with higher path regularity benefit the most from the pattern-matching approach of travel time prediction. This concept can be applied to other data-mining applications using sparse mobile sensors. We also showed the benefits of using driver information in travel time prediction when trajectories are collected from drivers with diverse driving styles. Our approach works regardless of whether the training set includes the user we want to make predictions for.

As the focus of our study included the scenario of a limited number of probe vehicles, the training data as well as the evaluation data are both collected from a relatively small group of taxis from the Gotcha project. These trajectories are a sub-sample of all taxi trajectories in Shenzhen and reflect the general taxi mobility patterns. However, our evaluation results could not account for areas rarely traveled by taxis, such as the suburban areas in the northern part of Region 1.

Therefore, additional trajectory data, potentially from non-taxi vehicles, would be necessary to apply the proposed method in those areas.

The proposed algorithm can be easily extended with additional data sources and with a larger road network. A common data source used in urban traffic estimation is real-time traffic incident reports. This information can be incorporated into the hybrid prediction step (Section 8.2.2), such that the algorithm assigns higher weights to recent travel time observations than historical travel time estimations on pathlets near the incident location. With enough training data, we may even learn congestion patterns caused by different types of unexpected events. When extending the current framework to a city scale, we can partition the city map into districts and learn the congestion patterns in each district in parallel. To predict the travel time of a cross-district trajectory, we will first use Algorithm 2 to predict the pathlet congestion states in each district and use a similar way as adjacent pathlet smoothing (Section 8.2.2) to make sure the predicted congestion states of boundary pathlets are consistent.

For future works, we will also address some of the limitations in our method. For example, we can improve the prediction for longer trips by taking in a stream of real-time trajectory data from all probe vehicles during the trip and updating the prediction results periodically. Another potential improvement is considering different GPS noise levels in the collected trajectories; i.e., many modern GPS loggers also output the number of satellites used to produce each measurement, which is strongly correlated with its GPS noise. This value can be used as a weighting constant to the travel time observations, when we compute the travel time distribution of pathlets.

ACKNOWLEDGMENT

The authors would like to thank Xiangxiang Xu and Professor Lin Zhang from Tsinghua-Berkeley Shenzhen Institute for providing the Gotcha dataset. Map data copyrighted OpenStreetMap contributors and available from https://www.openstreetmap.org.

REFERENCES

OpenStreetMap contributors. 2015. *Planet Dump*. Retrieved from https://planet.osm.org. https://www.openstreetmap.org.

John E. Angus. 1994. The probability integral transform and related results. *SIAM Rev.* 36, 4 (1994), 652–654.

Chen Chen, Hao Su, Qixing Huang, Lin Zhang, and Leonidas Guibas. 2013b. Pathlet learning for compressing and planning trajectories. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 392–395.

Hao Chen, Hesham A. Rakha, and Catherine C. McGhee. 2013a. Dynamic travel time prediction using pattern recognition. In *Proceedings of the ITS World Congress 2013*.

Mei Chen and Steven Chien. 2001. Dynamic freeway travel-time prediction with probe vehicle data: Link based versus path based. *Transport. Res. Rec.: J. Transport. Res. Board* 1768 (2001), 157–161.

Jocelyn Chi, Eric Chi, and Richard Baraniuk. 2016. k-POD: A method for k-means clustering of missing data. *Amer. Stat.* 70, 1 (2016), 91–99.

Corrado De Fabritiis, Roberto Ragona, and Gaetano Valenti. 2008. Traffic estimation and prediction based on real time floating car data. In *Proceedings of the International Conference on Intelligent Transportation Systems (ITSC'08)*. IEEE, 197–203.

Jianhe Du and Lisa Aultman-Hall. 2007. Increasing the accuracy of trip rate information from passive multi-day {GPS} travel datasets: Automatic trip end identification issues. *Transport. Res. Part A: Policy Pract.* 41, 3 (2007), 220–232.

Greg Hamerly and Charles Elkan. 2003. Learning the K in K-means. In *Neural Information Processing Systems*. MIT Press, 2003.

Aude Hofleitner and Alexandre Bayen. 2011. Optimal decomposition of travel times measured by probe vehicles using a statistical traffic flow model. In *Proceedings of the International Conference on Intelligent Transportation Systems (ITSC'11)*. IEEE, 815–821.

Aude Hofleitner, Ryan Herring, Pieter Abbeel, and Alexandre Bayen. 2012. Learning the dynamics of arterial traffic from probe data using a dynamic Bayesian network. *IEEE Trans. Intell. Transport. Syst.* 13, 4 (2012), 1679–1693.

Yijuan Jiang and Xiang Li. 2013. Travel time prediction based on historical trajectory data. *Ann. GIS* 19, 1 (2013), 27–35.

Gurunath Kedar-Dongarkar and Manohar Das. 2012. Driver classification for optimization of energy usage in a vehicle. *Procedia Comput. Sci.* 8 (2012), 388–393.

Georgios Kellaris, Nikos Pelekis, and Yannis Theodoridis. 2013. Map-matched trajectory compression. *J. Syst. Softw.* 86, 6 (2013), 1566–1579.

Yang Li, Dimitrios Gunopulos, Cewu Lu, and Leonidas Guibas. 2017. Urban travel time prediction using a small number of GPS floating cars. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL'17)*. ACM, New York, NY, Article 3, 10 pages.

John-Michael McNew. 2012. Predicting cruising speed through data-driven driver modeling. In *Proceedings of the 15th International IEEE Conference on Intelligent Transportation Systems (ITSC'12)*. IEEE, 1789–1796.

Xiaoguang Niu, Ying Zhu, and Xining Zhang. 2014. DeepSense: A novel learning mechanism for traffic prediction with taxi GPS traces. In *Proceedings of the IEEE Global Communications Conference*. IEEE, 2745–2750.

Mahmood Rahmani, Erik Jenelius, and Haris N. Koutsopoulos. 2013. Route travel time estimation using low-frequency floating car data. In *Proceedings of the International Conference on Intelligent Transportation Systems*. IEEE, 2292–2297.

Gilles Reymond, Andras Kemeny, Jacques Droulez, and Alain Berthoz. 2001. Role of lateral acceleration in curve driving: Driver model and experiments on a real vehicle and a driving simulator. *Human Factors* 43, 3 (2001), 483–495.

Bin Shi, Li Xu, Jie Hu, Yun Tang, Hong Jiang, Wuqiang Meng, and Hui Liu. 2015. Evaluating driving styles by normalizing driving behavior based on personalized driver modeling. *IEEE Trans. Syst., Man, Cyber.: Syst.* 45, 12 (2015), 1502–1508.

Penghui Sun, Shixiong Xia, Guan Yuan, and Daxing Li. 2016. An overview of moving object trajectory compression algorithms. *Mathematical Problems in Engineering* 2016, Algorithms for Compressive Sensing Signal Reconstruction with Applications (Special Issue), Article 6587309 (2016).

Hongjian Wang, Yu-Hsuan Kuo, Daniel Kifer, and Zhenhui Li. 2016. A simple baseline for travel time estimation using large-scale trip data. In *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 61.

Wenshuo Wang, Junqiang Xi, and Huiyan Chen. 2014a. Modeling and recognizing driver behavior based on driving data: A survey. *Mathematical Problems in Engineering* 2014, Mathematical Modeling, Analysis, and Advanced Control of Complex Dynamical Systems (Special Issue), Article 245641 (2014).

Yilun Wang, Yu Zheng, and Yexiang Xue. 2014b. Travel time estimation of a path using sparse trajectories. In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 25–34.

Chun-Hsin Wu, Jan-Ming Ho, and Der-Tsai Lee. 2004. Travel-time prediction with support vector regression. *IEEE Trans. Intell. Transport. Syst.* 5, 4 (2004), 276–281.

Longgang Xiang, Meng Gao, and Tao Wu. 2016. Extracting stops from noisy trajectories: A sequence oriented clustering approach. *ISPRS Int. J. Geo-Inform.* 5, 3 (2016), 29.

Tao Xu, Xiang Li, and Christophe Claramunt. 2017. Trip-oriented travel time prediction (TOTTP) with historical vehicle trajectories. *Frontiers of Earth Science* 12, 2 (2017), 1–11.

Xiangxiang Xu, Pei Zhang, and Lin Zhang. 2014. Gotcha: A mobile urban sensing system. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys'14)*. ACM, 316–317.

M. Kerber, L. Zhang, Y. Li, Q. Huang, and L. Guibas. 2013. Large-scale joint map matching of GPS traces. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL'13)*.

Jing Yuan, Yu Zheng, Chengyang Zhang, Wenlei Xie, Xing Xie, Guangzhong Sun, and Yan Huang. 2010. T-drive: Driving directions based on taxi trajectories. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL'10)*. ACM, 99–108.

Xianyuan Zhan, Samiul Hasan, Satish V. Ukkusuri, and Camille Kamga. 2013. Urban link travel time estimation using large-scale taxi data with partial information. *Trans. Res. Part C: Emerg. Technol.* 33, 8 (2013), 37–49.

Xianyuan Zhan, Yu Zheng, Xiuwen Yi, and Satish V. Ukkusuri. 2017. Citywide traffic volume estimation using trajectory data. *IEEE Trans. Knowl. Data Eng.* 29, 2 (Feb. 2017), 272–285.

Faming Zhang, Xinyan Zhu, Tao Hu, Wei Guo, Chen Chen, and Lingjia Liu. 2016. Urban link travel time prediction based on a gradient boosting method considering spatiotemporal correlations. *ISPRS Int. J. Geo-Inform.* 5, 11 (2016), 201.

Yu Zheng, Lizhu Zhang, Xing Xie, and Wei-Ying Ma. 2009. Mining interesting locations and travel sequences from GPS trajectories. In *Proceedings of the Conference on World Wide Web*. ACM, 791–800.

Received May 2018; revised January 2019; accepted January 2019