Urban Travel Time Prediction using a Small Number of GPS Floating Cars

Yang Li *
yangli1@stanford.edu

Dimitrios Gunopulos†
dg@di.uoa.gr

Cewu Lu
lucewu06@hotmail.com

Leonidas Guibas
guibas@cs.stanford.edu

ABSTRACT

Predicting the travel time of a path is an important task in route planning and navigation applications. As more GPS floating car data has been collected to monitor urban traffic, GPS trajectories of floating cars 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 GPS floating cars are available. We developed 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. Experimental results using 10-15 taxis tracked for 11 months in urban areas of Shenzhen, China show that our prediction has on average 5.4 minutes of error on trips of duration 10-75 minutes. This result improves the baseline approach of using purely historical trajectories by 2-30% on regions with various degree 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;*

KEYWORDS

Travel time prediction, Mobile sensors, GPS trajectories

*Yang Li, Cewu Lu and Leonidas Guibas are affiliated with Stanford University, Stanford, CA, United States. Yang Li is also affiliated with Tsinghua-Berkeley Shenzhen Institute, Shenzhen, China

†Dimitrios Gunopulos is affiliated with National and Kapodistrian University of Athens, Zografou, Athens, Greece

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SIGSPATIAL'17, Los Angeles Area, CA, USA
© 2017 ACM. 978-1-4503-5490-5/17/11…$15.00
DOI: 10.1145/3139958.3139971

ACM Reference format:

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 SIGSPATIAL'17, Los Angeles Area, CA, USA, Nov 7–10, 2017,* 10 pages.

DOI: 10.1145/3139958.3139971

1 INTRODUCTION

Modern navigation and location-based applications rely on accurately predicting of 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 [5, 9, 11, 14, 17, 18, 21, 23]. Most of them make use of thousands of probe vehicles tracked simultaneously, such that the traffic speed on a subset of the roads are 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 deployed 100 sensor-equipped electric taxis in Shenzhen, China in several stages from 2014 to 2016 [19]. During the initial 15-month pilot stage, no more than 15 vehicles from the project were active (Figure 1).

In this paper we use the Gotcha pilot data to investigate the difficult problem of travel time prediction when having a very small number of concurrently active GPS-floating cars 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

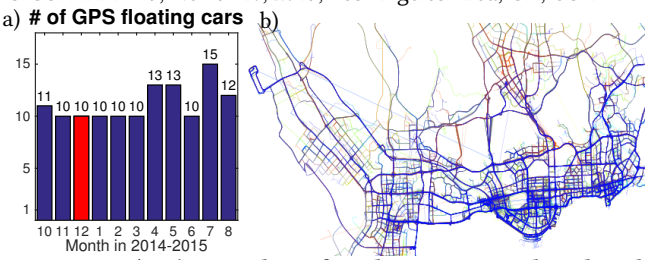

Figure 1: a.) The number of probing cars employed each month by the Gotcha pilot study. b.) GPS traces of 10 cars in December, 2014.

be available from a smaller subset of the active GPS cars. For example, when crowdsourced trajectories are used, some drivers may decide to upload their traces in batches rather than immediately, to conserve battery power, or when there is WIFI. 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 contain the time when drivers stop temporarily to drop off or pick up passengers, or intentionally slows down to find passengers on the side walk. 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 various amount of uncertainty in travel time observations.

To address these challenges, we use a **path-based** approach, which decomposes a route into a sequence of popular paths on the road network and predicts the travel time on each path [17]. 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 path-based approach because path travel time also include 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 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 which hasn't been traversed by any probing vehicle recently. We observe that on a local scale, such as several neighboring roads (or overlapping pathlets), the congestion patterns are reoccurring (Figure 2). 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

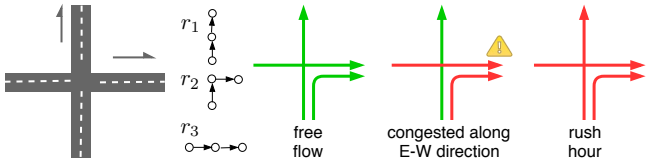

Figure 2: Visualization of 3 congestion patterns over pathlets r_1, r_2 and r_3 at an intersection. Colors red and green represent congested and non-congested traffic states of a given pathlet.

constraints. Hence it allows traffic delays caused by random events, such as weather, accident 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.

The contributions of this work are threefold:

(1) Proposing a 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.

2 RELATED WORKS

This section reviews previous works on the travel time prediction problem for urban road networks. We classify existing methods 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 [18] and license plate identification cameras [4].

For 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 et al. models the travel time distributions of links based on a traffic flow model [9]. Zhan et al. 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 [21]. More generally, one can estimate traffic parameters, such as the speed and the flow volume [7][22] 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 [10], pattern matching [4], gradient boosting regression tree [23] and deep learning [13]. 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 [13][23]. 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 [14], 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 [5], 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's 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 probing vehicle. Wang et al. discussed the trade off between subpath lengths and the minimum support size in path-based prediction [17]. 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 [15] and [12]. In particular, Chen et al. finds a compact dictionary of pathlets that reconstruct trajectories by fewer pieces (more compressed trajectory) [3]. 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 [5] only use historical data to predict the travel time of a query. In [17], 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, imputated 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 [11, 16]. 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 et al. compute the travel time distribution of a query trip from matched historical trajectories and use statistical tests to remove outliers [11]. Wang et al. infer the travel time of trips that are not found in historical data from nearby trips, while adjusting for periodic traffic patterns [16]. Trip-based travel time prediction can also be extended hierarchically to allow some route diversity. In [20], Yuan et al. 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 can not be applied to our scenario as we can not reliably identify the true starting and ending points of a taxi trip in an unlabeled GPS trace.

3 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 is 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 $\tau_G = \left\{e_1, \ldots, e_{|\tau_G|}\right\}$.

We formally define the travel time prediction problem as follows:

Definition 3.1 (The travel time prediction problem). 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.

In this problem, we compute the expected travel time of the query path, rather than the travel time of a particular driver. Hence driver identities are not considered. In addition, we only study trips in the near future (10min-1.5hr). 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.

4 ALGORITHM OUTLINE

Figure 3 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 3.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 3.b). We abuse the notation of T and R so that 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 3.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 time 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 Dec 2, 2014. 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 3.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 Section 5- 6.

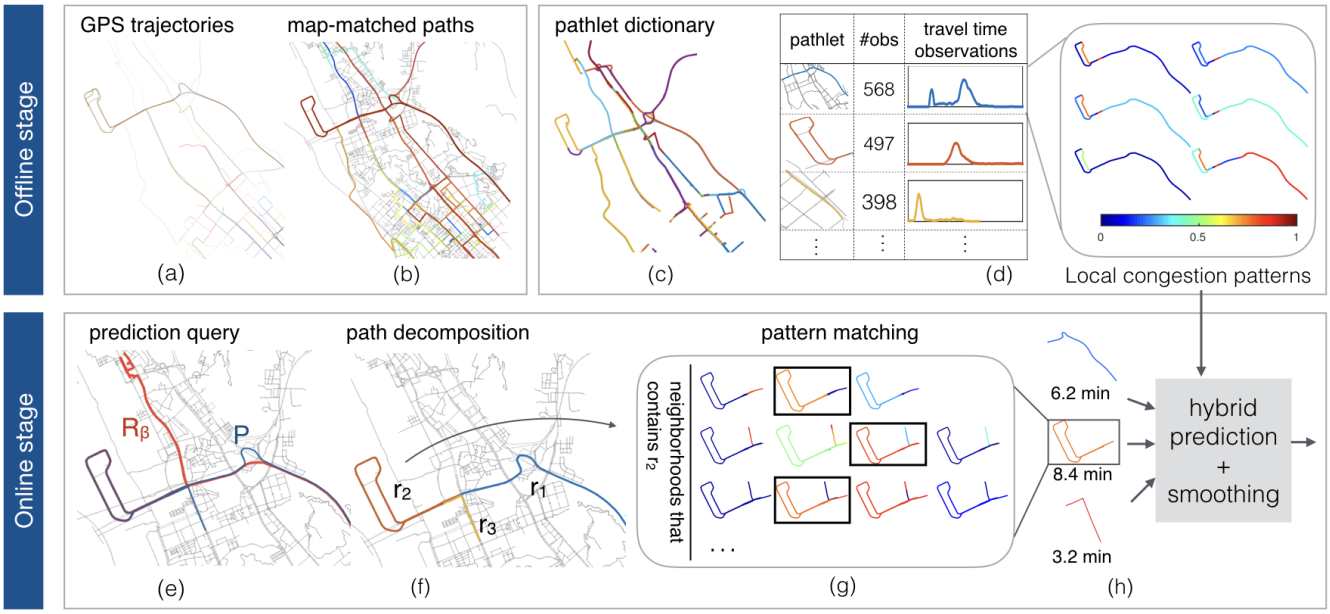

Figure 3: Our travel time prediction framework consists of the offline stage that preprocesses data (a-b) and learns travel time patterns (c-d), and the online stage that process travel time prediction queries (e-h). See section 4 for details.

Travel time prediction. Figure 3.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 patterns 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 3.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, $d_t(r_2) = 8.4$min 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 3.h). The algorithm will be presented in Section 7.

5 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 [3]. Then we will discuss how to derive pathlet travel time observations from input trajectories.

5.1 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 [3] 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:

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

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 $|p(\tau)|$.

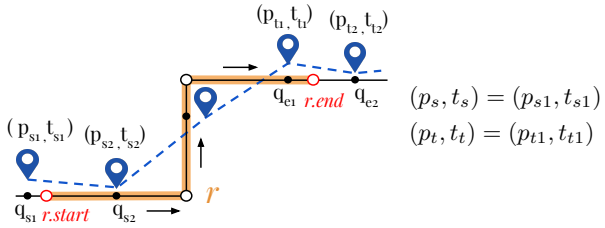

Figure 4: This figure illustrates an example of computing the travel time observation of pathlet r (highlighted in orange) by a trajectory, which contains 5 GPS samples (blue drop pins) projected to pathlet r and its adjacent road links.

5.2 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. 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 (see Figure 4). 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)

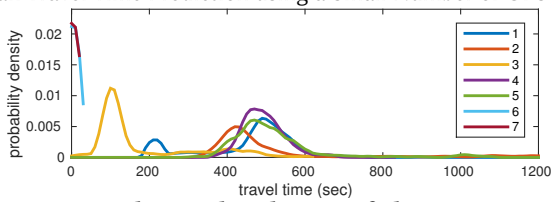

Figure 5: Travel time distribution of the seven most frequently used pathlets.

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 5 shows the travel time distribution of the top ten 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* [2] to $d_\pi(r)$, such that the transformed value, $\hat{d}_\pi(r)$ is in a uniform distribution between 0 and 1. This transformation allows us to compare travel times between different pathlets.

6 LEARNING CONGESTION PATTERNS

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

6.1 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 floating cars, most frames will 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 scalar value $d_{f_i}(r)$ and compute its congestion state $\hat{d}_{f_i}(r)$. The step size is chosen empirically based on the observation sparsity. With 15 floating cars, we found 30 minutes to strike 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. $M(r)_i = \begin{bmatrix} \hat{d}_{f_i}(o_1) & \ldots & \hat{d}_{f_i}(o_s) \end{bmatrix}$. 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:

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

6.2 Congestion feature clustering with temporal constraint

Having obtained matrix $M(r)$ that represents the dynamic congestion status of $OL(r)$, we first apply k-POD [6], 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 mechanism and high missing rate.

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 4000 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 [8].

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 P be the $N \times k$ cluster assignment matrix. Each row p_i of P is a binary vector such that $p_i(j) = 1$ if frame f_i is assigned to cluster j or 0 otherwise. The k cluster centers are stored as row vectors in $k \times s$ matrix C. We initialize P and C using the results from Section 6.2, then find their optimal values by solving the following minimization problem.

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

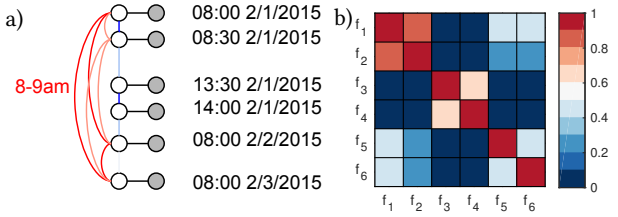

Figure 6: a) Illustration of a weighted consistency graph of congestion patterns in observed frames (white nodes). Shaded nodes represent the observation in each frame labeled by its starting time. Red and blue edges represent the time-of-day similarity and the similarity between nearby frames. The opacity of an edge represents its weight. b.) Visualization of W, the adjacency matrix of the weighted consistency graph in (a). In practice, W is mostly sparse.

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 6 illustrates a simple example of how pairwise consistencies are defined among frames. We relax the integer constraint on P 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:

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

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

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

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.

7 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.

7.1 Prediction using historical data

We first introduce a baseline prediction method that only relies on historical trajectories H. 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. 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 $d^H_t(P) = d^H_t(P_m)$.

7.2 Prediction exploiting current observations

7.2.1 *Congestion pattern matching.*

We use pattern matching to incorporate recent congestion states observed by R_β into travel time predictions. Our approach is summarized in Algorithm 1.

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: H
Current (departure) time: t
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**
1 **if** r_i *is observed by* R_β **then**
2 $\hat{d}^R(r_i) \leftarrow$ `PredictByPM`$(r_i,\ R_\beta)$
3 $t_{i+1} = t_i + cdf^{-1}\left(\hat{d}^R_{t_i}(r_i)\right)$
else
4 $\hat{d}^R(r_i) \leftarrow cdf(D(r_i), d^H_{t_i}(r_i))$
5 $t_{i+1} = t_i + d^H_{t_i}(r_i)$
end
end
6 $d^R(r_i) = cdf^{-1}(D(r_i), \hat{d}^R(r_i))$
7 $d_t(P) \leftarrow \sum_i^m d^R(r_i)$

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 6. This results in one prediction of the travel time of pathlet r, $\hat{d}_{o_l}(r)$. We aggregate predictions from multiple neighborhoods using a weighted average, where the weight function $w(r, o_l)$ is the correlation coefficient between the congestion status of r and the status of o_l in $M(r)$.

If r is not observed, 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 (Line 6-7).

7.2.2 *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 doesn't 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 time that are predicted independently doesn't 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. In order 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 RESULTS

8.1 Experimental data

We test our algorithm using GPS trajectories of 15 GPS-equipped electric taxis deployed by the Gotcha II project from Oct 2014 to September 2015 in Shenzhen, China [19]. The GPS sampling rate is 10 seconds per sample. Only 10 unique taxis were active for six months of the eleven-month study. We divide the urban districts of Shenzhen into 4 regions (Figure 7.a) and test travel time prediction in each region.Table 1 summarizes the characteristics of the districts from west to east. 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 in the morning of November 2, 2014. Despite taken from a weekday morning, on average only 3.32% of the road links in the test region are observed.

Table 1: Specs of test regions

Region ID	Name	# of links	# of trajectories	dictionary size	7-7:30am coverage
1	Bao'an	7649	11373	3438	0.86%
2	Nanshan	2991	33859	11363	5.35%
3	Futian	7574	60111	47898	5.40%
4	Luohu	5651	20378	32589	2.80%

The Open Street Map of Shenzhen is used for map matching [1]. 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.

In each test region, we randomly select 300 trajectories from January to September 2015 as test data. The rest of the trajectories are considered the training data for learning congestion patterns. This allows each test query to have at least 5 months of historical data to rely on. We use the actual trip duration of each test trajectory as the prediction ground truth. Trip duration is bounded between 40-4500 seconds. Tested on a Dell T3600 workstation with 3.2GHz

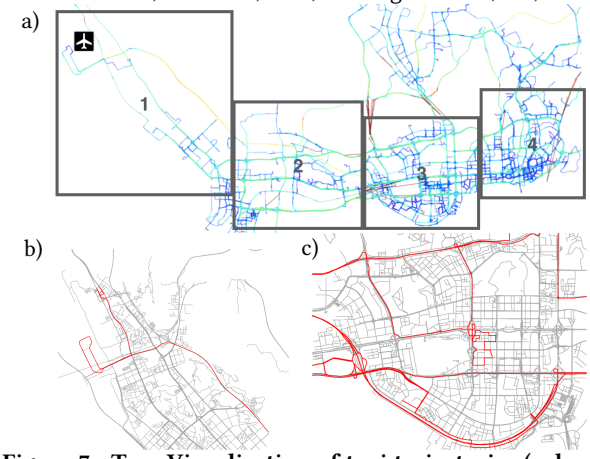

Figure 7: Top: Visualization of taxi trajectories (colored by id) in one month and the bounding box of each region. The plane icon in Region 1 marks the Shenzhen International Airport. Bottom: The road network for Region 1 (BaoAn) and Region 3 (Futian). Red edges indicate road segments that are visited by at least one floating car from 7:00am to 7:30am on Nov 2, 2014.

processors and 32GB RAM, the offline step takes approximately 4 hours per region.The average query processing time is 0.7 seconds.

We evaluate 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 7.2.2 as `PredictPM+`; and refer to the variation that uses both hybrid prediction and adjacent pathlet smoothing as `PredictPM++`.

We also compare the baseline and `PredictPM++` with `CATD-OC`, a popular path-based travel time estimation algorithm from [17]. 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 preserved in our test datasets. This is the case when anonymous trajectories are used, or when the driver pool is constantly changing. 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 15 minutes long and the total number of time slices used in the tensor decomposition process is 8 (2 hours).

8.2 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 don't 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) [20]:

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

8.3 Travel time prediction result

We compare the overall travel time prediction results in different test regions in Table 2. 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 Region 3 and 4 is much higher than the other two because of higher route diversity and higher taxi demand.

Table 2: Regional travel time prediction results

Region ID	ρ^R	Baseline		PredictPM++		%
		MRE	MAE	MRE	MAE	improved
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%

Using (`PredictPM++`), the average MRE is 0.171, and the average MAE is 5.4 minutes. 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. On the other hand, 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, 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. Table 3 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 3. Only test queries with successful tensor decomposition are used for evaluating all three methods[1]. The comparison results show that both `Baseline` and `PredictPM++` out-perform `CATD-OC` on the test data. [2] 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 9.a plots the MRE of test trajectories in Region 1 given recent trajectories of different window

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

Table 3: Prediction Accuracy of `Baseline`, `PredictPM++` and `CATD-OC`

Region ID	success rate	CATD-OC MRE	CATD-OC MAE	Baseline MRE	Baseline MAE	PredictPM++ MRE	PredictPM++ 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

size β and using different algorithm variations. When β is small, we have fewer observations from recent trajectories. On the other hand, if β is too large, the traffic observations may become out of date, so that they couldn't be used to predict future travel time. In most test regions, we found $\beta = 1.5$ hours 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 7.2.2 have a small but consistent improvement on the prediction accuracy.

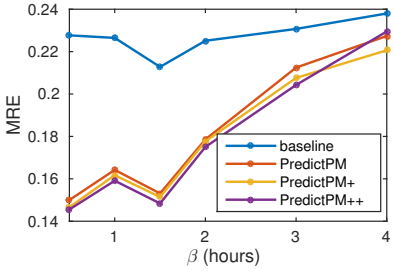

Figure 8: Comparison of prediction accuracy between the baseline and variations of our algorithm

Effect of dictionary size. We evaluate the impact of pathlet dictionary size on prediction error using trajectories in Region 1.

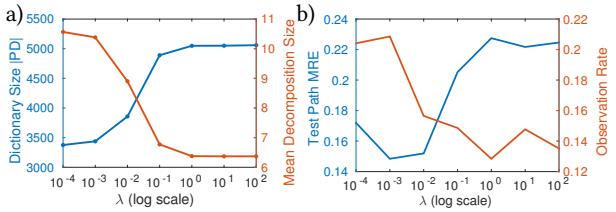

Figure 9: a.) Effect of λ on dictionary size $|PD|$ and average decomposition size. b.) Effect of λ on MRE and the observation ratio ρ^R.

Figure 9.b shows how parameter λ, introduced in Section 5.1, allows us to control the dictionary size and the average trajectory decompostion size; Figure 9.c 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 MRE and the 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.

Number of GPS floating cars. To determine the performance of the algorithm with even fewer floating cars, we limit the number of taxis observed per month to 5,7,9,11 and 13 and evaluate the prediction results using 500 random queries from Region 1. As shown in Figure 10.a , when the number of taxis decreases, the observation rate ρ^R drops from 23.8% to 3.3%, while the prediction accuracy actually increases for the observed trajectories, since having fewer drivers reduces the variability in the historical travel time of a pathlet. Nevertheless, Figure 10.b shows the accuracy improvement of the pattern matching approach consistently increases when more floating cars are used. With 11 taxis, the MRE of `PredictPM++` is 21.6% smaller than the MRE of the baseline.

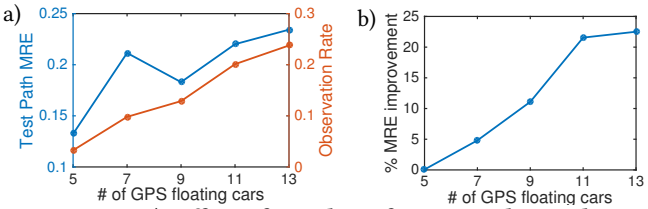

Figure 10: a.) Effect of number of taxis on the prediction accuracy (MRE) and the observation ratio. b.) Relative reduction in MRE of `PredictPM++` from the baseline.

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 [3]. 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.

Figure 11.a and 11.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. On the other hand, the correlation between relative error and trip distance is less salient ($r_{\texttt{baseline}} = -0.292$, $r_{\texttt{CATD-OD}} = -0.261$, $r_{\texttt{PredictPM++}} = -0.116$). 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 11.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 from the other two methods.

9 CONCLUSION AND FUTURE WORKS

While 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 paper presented an algorithm framework for predicting path travel time from GPS trajectories, under the scenario of 10-15 of GPS-floating cars 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

[3] 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.

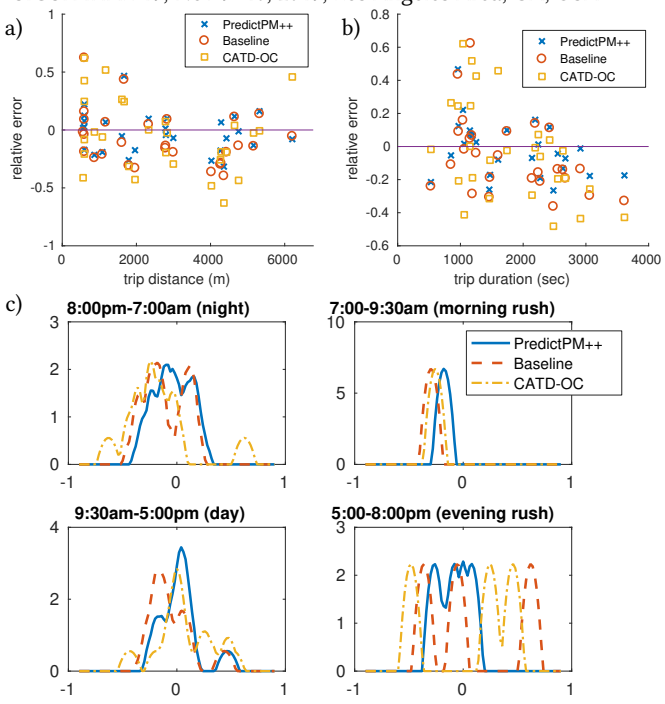

Figure 11: Visualizations of prediction errors for weekday travel time queries in Region 1 using `PredictPM++` (blue), the baseline method (orange) and `CATD-OD` (yellow). a.) Relative prediction error vs. trip distance. b.) Relative error vs. trip duration. c.) Distribution of prediction error for trips that depart within 4 different periods of a day.

relevant pathlets from recent trajectories, then infers the travel time of the query path from the identified pattern and the historical travel time. We experimented on trajectories collected by 10-15 taxis over 11 months and demonstrated 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.

For future works, we will 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 GPS floating cars 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. 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.

REFERENCES

[1] 2015. Open Street Map. www.openstreetmap.org. (2015).
[2] John E Angus. 1994. The probability integral transform and related results. *SIAM review* 36, 4 (1994), 652–654.
[3] Chen Chen, Hao Su, Qixing Huang, Lin Zhang, and Leonidas Guibas. 2013. Pathlet learning for compressing and planning trajectories. In *Proceedings of ACM SIGSPATIAL 2013*. ACM, 392–395.
[4] Hao Chen, Hesham A Rakha, and Catherine C McGhee. 2013. Dynamic Travel Time Prediction using Pattern Recognition. In *ITS World Congress 2013*.
[5] Mei Chen and Steven Chien. 2001. Dynamic freeway travel-time prediction with probe vehicle data: Link based versus path based. *Transportation Research Record: Journal of the Transportation Research Board* 1768 (2001), 157–161.
[6] Jocelyn Chi, Eric Chi, and Richard Baraniuk. 2016. k-POD: A Method for k-Means Clustering of Missing Data. *The American Statistician* 70, 1 (2016), 91–99.
[7] Corrado De Fabritiis, Roberto Ragona, and Gaetano Valenti. 2008. Traffic estimation and prediction based on real time floating car data. In *Proceeds of ITSC 2008*. IEEE, 197–203.
[8] Greg Hamerly and Charles Elkan. 2003. Learning the K in K-Means. In *In Neural Information Processing Systems*. MIT Press, 2003.
[9] Aude Hofleitner and Alexandre Bayen. 2011. Optimal decomposition of travel times measured by probe vehicles using a statistical traffic flow model. In *Proceedings of ITSC 2011*. IEEE, 815–821.
[10] Aude Hofleitner, Ryan Herring, Pieter Abbeel, and Alexandre Bayen. 2012. Learning the dynamics of arterial traffic from probe data using a dynamic Bayesian network. *Intelligent Transportation Systems, IEEE Transactions on* 13, 4 (2012), 1679–1693.
[11] Yijuan Jiang and Xiang Li. 2013. Travel time prediction based on historical trajectory data. *Annals of GIS* 19, 1 (2013), 27–35.
[12] Georgios Kellaris, Nikos Pelekis, and Yannis Theodoridis. 2013. Map-matched trajectory compression. *Journal of Systems and Software* 86, 6 (2013), 1566–1579.
[13] Xiaoguang Niu, Ying Zhu, and Xining Zhang. 2014. DeepSense: a novel learning mechanism for traffic prediction with taxi GPS traces. In *IEEE Global Communications Conference 2014*. IEEE, 2745–2750.
[14] Mahmood Rahmani, Erik Jenelius, and Haris N Koutsopoulos. 2013. Route travel time estimation using low-frequency floating car data. In *Proceedings of ITSC 2013*. IEEE, 2292–2297.
[15] Penghui Sun, Shixiong Xia, Guan Yuan, and Daxing Li. 2016. An Overview of Moving Object Trajectory Compression Algorithms. *Mathematical Problems in Engineering* 2016 (2016).
[16] 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.
[17] Yilun Wang, Yu Zheng, and Yexiang Xue. 2014. Travel time estimation of a path using sparse trajectories. In *Proceedings of ACM SIGKDD 2014*. ACM, 25–34.
[18] Chun-Hsin Wu, Jan-Ming Ho, and Der-Tsai Lee. 2004. Travel-time prediction with support vector regression. *IEEE transactions on intelligent transportation systems* 5, 4 (2004), 276–281.
[19] Xiangxiang Xu, Pei Zhang, and Lin Zhang. 2014. Gotcha: A Mobile Urban Sensing System. In *Proceedings of SenSys 2014*. ACM, 316–317.
[20] 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 ACM SIGSPATIAL 2010*. ACM, 99–108.
[21] Xianyuan Zhan, Samiul Hasan, Satish V. Ukkusuri, and Camille Kamga. 2013. Urban link travel time estimation using large-scale taxi data with partial information. *Transportation Research Part C: Emerging Technologies* (2013), 37–49.
[22] X. Zhan, Y. Zheng, X. Yi, and S. V. Ukkusuri. 2017. Citywide Traffic Volume Estimation Using Trajectory Data. *IEEE Transactions on Knowledge and Data Engineering* 29, 2 (Feb 2017), 272–285. DOI: http://dx.doi.org/10.1109/TKDE.2016.2621104
[23] 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 International Journal of Geo-Information* 5, 11 (2016), 201.