Travel Time Estimation of a Path using Sparse Trajectories

Yilun Wang[1,2,*], Yu Zheng[1,+], Yexiang Xue[1,3,*]
[1]Microsoft Research, No.5 Danling Street, Haidian District, Beijing 100080, China
[2]College of Computer Science, Zhejiang Univeristy
[3]Department of Computer Science, Cornell University
{v-yilwan, yuzheng}@microsoft.com, yexiang@cs.cornell.edu

ABSTRACT
In this paper, we propose a citywide and real-time model for estimating the travel time of any path (represented as a sequence of connected road segments) in real time in a city, based on the GPS trajectories of vehicles received in current time slots and over a period of history as well as map data sources. Though this is a strategically important task in many traffic monitoring and routing systems, the problem has not been well solved yet given the following three challenges. The first is the data sparsity problem, i.e., many road segments may not be traveled by any GPS-equipped vehicles in present time slot. In most cases, we cannot find a trajectory exactly traversing a query path either. Second, for the fragment of a path with trajectories, they are multiple ways of using (or combining) the trajectories to estimate the corresponding travel time. Finding an optimal combination is a challenging problem, subject to a tradeoff between the length of a path and the number of trajectories traversing the path (i.e., support). Third, we need to instantly answer users' queries which may occur in any part of a given city. This calls for an efficient, scalable and effective solution that can enable a citywide and real-time travel time estimation. To address these challenges, we model different drivers' travel times on different road segments in different time slots with a three dimension tensor. Combined with geospatial, temporal and historical contexts learned from trajectories and map data, we fill in the tensor's missing values through a context-aware tensor decomposition approach. We then devise and prove an object function to model the aforementioned tradeoff, with which we find the most optimal concatenation of trajectories for an estimate through a dynamic programming solution. In addition, we propose using frequent trajectory patterns (mined from historical trajectories) to scale down the candidates of concatenation and a suffix-tree-based index to manage the trajectories received in the present time slot. We evaluate our method based on extensive experiments, using GPS trajectories generated by more than 32,000 taxis over a period of two months. The results demonstrate the effectiveness, efficiency and scalability of our method beyond baseline approaches.

Categories and Subject Descriptors
H.2.8 [Database Management]: Database Applications - data mining, Spatial databases and GIS;

Keywords
Travel time estimation; tensor; trajectories; urban computing;

**The paper was done when the first and third authors were interns in Microsoft Research under the supervision of the second author who contributed the main idea and algorithms of this paper.*

KDD'14, August 24–27, 2014, New York, New York, USA.

http://dx.doi.org/10.1145/2623330.2623656

1. INTRODUCTION
Real-time estimation of the travel time of a path, which is represented by a sequence of connected road segments, is of great importance for traffic monitoring [1], finding driving directions [20], ridesharing [13] and taxi dispatching [22]. Existing solutions, e.g., using loop sensors, usually tell people the travel speed of an individual road segment rather than the travel time of an entire path. The latter's value is not a simple summation of the travel time of each individual road segment, as a path also contains road intersections (sometimes with traffic lights) where a driver needs to slow down or wait for a while. Explicitly modeling the time delay at an intersection is not easy [8]. In addition, these methods have limited coverage, as many streets do not have a loop sensor embedded.

An alternative method is to use floating car data (e.g., GPS trajectories of vehicles) to estimate the travel time of a path. For example, as shown in Figure 1, we estimate the travel time of path $r_1 \rightarrow r_2 \rightarrow r_3 \rightarrow r_4$, using four trajectories Tr_1, Tr_2, Tr_3, and Tr_4. Unfortunately, there are three major issues remaining unsolved in existing methods. They are as follows:

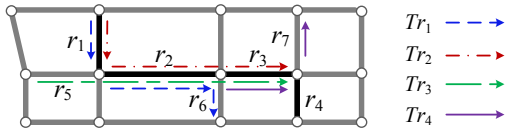


Figure 1. Problem demonstration

1) Data sparsity: For example, r_4 is not traversed by any trajectory in the previous 30 minutes. Using an average of r_4's historical travel times is not accurate enough (since its traffic conditions change over time of day and day of the week). Sometimes, the road may never be traversed by any trajectories (even in history) in our dataset, as in practice we only have the data of a sample of vehicles.

2) Trajectory concatenation: For the sub-path (e.g., $r_1 \rightarrow r_2 \rightarrow r_3$) with trajectories, how to combine these trajectories effectively to achieve an accurate estimate is still a challenging problem. Clearly, there are multiple ways of using the four trajectories shown in Figure 1. For instance, we can calculate the travel time of $r_1 \rightarrow r_2 \rightarrow r_3$ solely based on Tr_2. Or, we can compute the travel time for r_1 (based on Tr_1 and Tr_2), r_2 (based on Tr_1, Tr_2 and Tr_3), and r_3 (using Tr_2, Tr_3 and Tr_4), separately. Later, the travel time of $r_1 \rightarrow r_2 \rightarrow r_3$ can be obtained by summing the travel times of each road segment. We can also use Tr_2 and Tr_3 to estimate the travel time of $r_2 \rightarrow r_3$, then concatenating it with that of r_1; or, do $r_1 \rightarrow r_2$ first based on Tr_1 and Tr_2, then concatenating it with r_3.

Different concatenations have their own advantages and disadvantages, subject to a trade-off between their support and length. The ideal situation is to estimate the travel time of $r_1 \rightarrow r_2 \rightarrow r_3$ using many trajectories like Tr_2 covering the entire path. Such trajectories reflect the traffic conditions of an entire path, including intersections, traffic lights and direction turns, hence, no need to model these complex factors separately and

+ Yu Zheng is the correspondence author of this paper.

explicitly. However, as the length of a path increases, the number of trajectories (i.e., the support) traveling on the path decreases (refer to Figure 10 A) for details). Consequently, the confidence of the travel time (derived from few drivers) decreases. For example, what if Tr_2 is generated by an uncommon driver or in an unusual situation like pedestrians crossing a street? Furthermore, in many cases, we cannot even find a trajectory passing an entire path. On the other hand, using the concatenation of shorter sub-paths can have more occurrences of trajectories on each sub-path (i.e., having a high confidence in the derived travel time for each sub-path). But this results in more fragments, across which the aforementioned complex factors are difficult to model. The more fragments a concatenation contains, the more inaccuracy a path's travel time could involve.

3) *Tradeoff among Scalability, effectiveness and efficiency*: As users can query any path in a city, we need to model the traffic conditions with a city scale, which usually contains tens of thousands of road segments. In the meantime, we have to answer users' query instantly. So, a good solution should be scalable, effective and efficient, all simultaneously. This requirement fails some complex models that work well on a particular road.

In this paper, we propose a model for instant Path Travel Time Estimation (PTTE), based on sparse trajectories generated by a sample of vehicles (e.g., some GPS equipped taxicabs) in the recent time slots as well as in history. Our model is comprised of two major components. One is to estimate the travel time for road segments without being traversed by trajectories through a context-aware tensor decomposition (CATD) approach. The second is to find the most optimal concatenation (OC) of trajectories to estimate a path's travel time using a dynamic programing solution. Our work has three primary contributions:

- *Dealing with the missing values*: We model different drivers' travel times on different road segments in different time slots with a three dimensional tensor. Combined with geospatial, temporal and historical contexts learned from other data sources, we fill in the tensor's missing values through a context-aware tensor decomposition approach. To expedite the inference, we partition a city into disjoint geo-regions and carry out the decomposition for each region in parallel.
- *Optimal concatenation*: We devise and prove an object function that can model the tradeoff between the support and length of a concatenation. Using a dynamic programming solution, we find the most optimal concatenation of trajectories for estimating a path's travel time. In addition, we use frequent trajectory patterns mined in advance to scale down the candidates of concatenation and propose a suffix-tree-based index to manage the recently received trajectories, improving the efficiency of our model.
- *Evaluation*: We evaluate our model with the real trajectories generated by over 32,000 taxis over a period of 2 month on Beijing's road network. The results of extensive experiments demonstrate the advantages of our model. A sample of the data has been released at [25].

The rest of the paper is organized as follows: Section 2 overviews our model. Section 3 elaborates on the method for inferring the travel time of road segments without trajectories. Section 3 introduces the method that searches for the most optimal concatenation. Section 4 presents the experiments and Section 5 summarizes related work. We conclude the paper in Section 6.

2. OVERVIEW

Definition 1: Road Network. A road network RN is comprised of a set of road segments $\{r\}$ connected among each other in a graph format. Each road segment r is a directed edge with two terminal points, a list of intermediate points describing the segment, a length $r.len$, a level $r.lev$ (e.g. a highway or a street), a direction $r.dir$ (e.g. one-way or bi-directional) and the number of lanes $r.n$.

Definition 2: Trajectory. A spatial *trajectory* Tr is a sequence of time-ordered points, $Tr: p_1 \rightarrow p_2 \rightarrow \cdots \rightarrow p_n$, where each point has a geospatial coordinate set and a timestamp, $p = (x, y, t)$.

Definition 3: Path. A path P is represented by a sequence of connected road segments, e.g., $P: r_1 \rightarrow r_2 \rightarrow \cdots \rightarrow r_n$, in an RN.

Definition 4: Trajectory pattern. A trajectory pattern TP is a sequential pattern of road segments with a support over a threshold, calculated by the number of trajectories traversing these road segments. If we set support as 2, $r_1 \rightarrow r_2$ and $r_2 \rightarrow r_3$ in Figure 1 are trajectory patterns, while $r_1 \rightarrow r_2 \rightarrow r_3$ is not eligible.

Definition 5: Concatenation. A path $\boldsymbol{P}$ can be decomposed into different concatenations (||) of its sub-paths, $\boldsymbol{P} = P_1 || \ldots || P_i \ldots || P_j \ \ldots || \ P_n$, $\forall 1 \leq i, j \leq n$, $i \neq j$, $P_i \cap P_j = \emptyset$. For instance, $r_1 \rightarrow r_2 \rightarrow r_3 \rightarrow r_4$ can be formed by $(r_1 \rightarrow r_2 \rightarrow r_3) || r_4$, or $(r_1 \rightarrow r_2) || (r_3 \rightarrow r_4)$, or $r_1 || (r_2 \rightarrow r_3 \rightarrow r_4)$. Thus, the travel time of $\boldsymbol{P}$ can be obtained via the summation of different concatenations, e.g., $t_{\boldsymbol{P}} = t_{r_1 \rightarrow r_2 \rightarrow r_3} + t_{r_4}$, or $t_{\boldsymbol{P}} = t_{r_1 \rightarrow r_2} + t_{r_3 \rightarrow r_4}$, or $t_{\boldsymbol{P}} = t_{r_1} + t_{r_2 \rightarrow r_3 \rightarrow r_4}$.

Definition 6: Travel Time. A driver u's travel time on a road segment r in time slot k is defined as $t_{r,u,k}$. Likewise, $t_{P,u,k}$ denotes u's travel time on path P in time slot k.

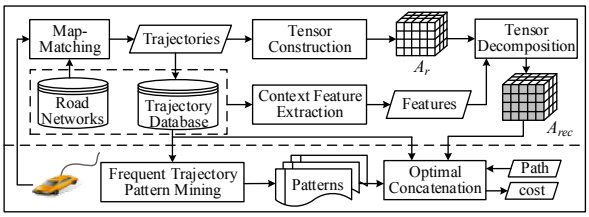


Figure 2. Framework of our model

Figure 2 presents the framework of our model which is comprised of two major parts. In the above part, we project each trajectory received in a current time slot onto a road network, using a map-matching algorithm [21]. The trajectories (combined with road network data) are then used to construct a 3D tensor $\mathcal{A}_r$ where the three dimensions stand for road segments, time slots and drivers, respectively. Each entry is the travel time of a particular driver on a particular road segment in a specific time slot. We partition a day into several time slots based on a certain time interval (e.g., we divide a day into 48 time slots with 30 minutes each in the experiments). Clearly, the tensor is very sparse (i.e., having many entries without values), as a driver can only travel a few road segments in a time slot. To deal with the data sparsity problem, we extract three categories of features, consisting of geospatial, temporal, and historical contexts, from the road network data and trajectories. The first two feature sets are stored in two matrices, respectively, and the historical context is represented by another tensor $\mathcal{A}_h$. The two matrices and $\mathcal{A}_h$ are then factorized with $\mathcal{A}_r$ collaboratively, helping fill $\mathcal{A}_r$'s missing entries in a current time slot (i.e., inferring the travel time of road segments without being traveled by trajectories in the current time slot). The general idea is that road segments with similar contexts could have a similar travel time. The context matrices and tensor reveal the similarity and with a more proportion of non-zero entries than $\mathcal{A}_r$, thereby reducing the factorization error and improving the inference accuracy. After filling

the missing entries in $\mathcal{A}_r$, we obtain the travel time of any driver on any road segment in current time slot (stored in $\mathcal{A}_{rec}$).

In the bottom part, given a query path P, we estimate its travel time in the current time slot, based on $\mathcal{A}_{rec}$, the trajectories received in the time slot and trajectory patterns. Specifically, we devise and prove an objective function that can represent the tradeoff between the length and support of a trajectory pattern. Based on the objective function, we find the most optimal concatenation of trajectories for a path, using a dynamic programing approach. In practice, it is not necessary to try every possible concatenation of a path, as some sub-paths have never been traversed by any trajectory. So, we mine frequent trajectory patterns from historical trajectories in advance and study the concatenation of these existing patterns to estimate the travel time of a path. This reduces the online computational loads significantly, while guaranteeing accuracy in travel time estimation. Note that we are not using the historical travel time of a trajectory pattern. The patterns just provide us with candidate schemes of subpaths for finding an optimal concatenation of a path. Each trajectory pattern's travel time in current time slot is mainly calculated based on the trajectories received in the time slot. If a pattern contains road segments without being traversed by trajectories in the current time slot, we retrieve the inferred time from $\mathcal{A}_{rec}$, according to the driver, road segment and time slot. For instance, two drivers (u_1, u_2) travelled $r_1 \rightarrow r_2$, but nobody traveled r_3 in a pattern $r_1 \rightarrow r_2 \rightarrow r_3$, in current time slot k. That is, $t_{r_1 \rightarrow r_2, u_1, k}$, and $t_{r_1 \rightarrow r_2, u_2, k}$ can be calculated from the present trajectory data, while $t_{r_3, u_1, k}$ and $t_{r_3, u_2, k}$ are unknown. In this case, we retrieve the latter two from $\mathcal{A}_{rec}$, calculating

$$t_{r_1 \rightarrow r_2 \rightarrow r_3, u_1, k} = t_{r_1 \rightarrow r_2, u_1, k} + t_{r_3, u_1, k}, \text{ and}$$
$$t_{r_1 \rightarrow r_2 \rightarrow r_3, u_2, k} = t_{r_1 \rightarrow r_2, u_2, k} + t_{r_3, u_2, k}.$$

With $\mathcal{A}_{rec}$, we can estimate a driver's travel time on a trajectory pattern even if the recently received data is incomplete. The dimension of drivers in $\mathcal{A}_{rec}$ enables us to calculate the variance among different drivers' travel times on a road segment or a sub-path. Intrinsically, different drivers travel the same road segment with different times, majorly depending on the different traffic conditions they experience. Thus, the variance implies the complexity of traffic conditions on a road segment or a sub-path, helping estimate a more accurate travel time of a path (elaborated in Section 4.1). Finally, the travel time of a path is calculated as:

$$T = \sum_{TP \in \Psi} \frac{\sum_{u \in U} t_{TP,u,k}}{|U|}; \qquad (1)$$

Where Ψ is the concatenation of path P, represented by a set of trajectory pattern TPs; U is a collection of drivers traversing (or partially traversing) a TP; k is the current time slot.

3. DEALING WITH MISSING VALUES

3.1 Tensor Building and Feature Extraction

To model the traffic conditions of the current time slot, we construct a tensor $\mathcal{A}_r \in \mathbb{R}^{N \times M \times L}$, with the three dimensions standing for road segments, drivers and time slots, respectively, based on the GPS trajectories received in the most recent L time slots and the road network data. As shown in Figure 3, an entry $\mathcal{A}_r(i, j, k) = c$ denotes the ith road segment is traveled by the jth driver with a time cost c in time slot k (e.g., 2-2:30pm). The last time slot denotes the present time slot, combined with the L-1 time slots right before it to formulate the tensor. Clearly, the tensor is very sparse as a driver can only travel a few road segments in a short time period. If we were able to fill in the missing entries in terms of the values of non-zero entries, we can know the travel time of any driver on any road segment in the present time slot.

A common approach to this problem is to decompose a tensor into the multiplication of a few (low-rank) matrices and a core tensor (or just a few vectors), based on the tensor's non-zero entries. For example, we can decompose $\mathcal{A}_r$ into the multiplication of a core tensor $S \in \mathbb{R}^{d_R \times d_U \times d_T}$ and three matrices, $R \in \mathbb{R}^{N \times d_R}$, $U \in \mathbb{R}^{M \times d_U}$, $T \in \mathbb{R}^{L \times d_T}$, if using a tucker decomposition model. An objective function is defined as Equation 2 to control the errors.

$$\mathcal{L}(S, R, U, T) = \frac{1}{2} \| \mathcal{A}_r - S \times_R R \times_U U \times_T T \|^2 + \frac{\lambda}{2} (\|S\|^2 + \|R\|^2 + \|U\|^2 + \|T\|^2), \qquad (2)$$

where $\|\cdot\|^2$ denotes the l_2 norm and $\frac{\lambda}{2}(\|S\|^2 + \|R\|^2 + \|U\|^2 + \|T\|^2)$ is a regularization of penalties to avoid over-fitting; d_R, d_U, and d_T are usually very small, denoting the number of latent factors. λ is a parameter controlling the contributions of the regularization. Afterwards, we can recover the missing values in $\mathcal{A}_r$ by multiplying decomposed factors as $\mathcal{A}_{rec} = S \times_R R \times_U U \times_T T$.

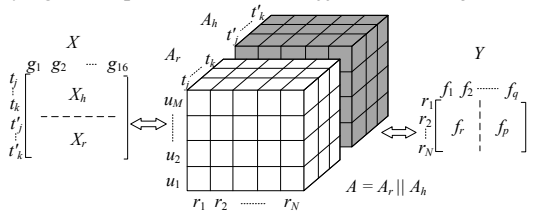


Figure 3. The model dealing with data sparsity

In our problem, however, the tensor is over sparse. For example, if setting 30 minutes as a time slot, only 0.03% entries of $\mathcal{A}_r$ have values. Decomposing $\mathcal{A}_r$ solely based on its own non-zero entries is not accurate enough. To this end, we build another tensor $\mathcal{A}_h$ based on the historical trajectories over a long period of time (e.g. one month). As shown in Figure 3, $\mathcal{A}_h$ has the same structure as $\mathcal{A}_r$, while an entry $\mathcal{A}_h(i, j, k) = c'$ denotes the jth driver's average travel time on the ith road segment in time slot k in the history. Intrinsically, $\mathcal{A}_h$ is much denser than $\mathcal{A}_r$, denoting the historical traffic patterns and drivers' behavior on an entire road network. For instance, using one-month trajectories and setting 30 minutes as a time slot, the non-zero entries of $\mathcal{A}_h$ is about 0.4%. Decomposing $\mathcal{A}_r$ and $\mathcal{A}_h$ together reduces the error of supplementing $\mathcal{A}_r$.

Besides $\mathcal{A}_h$, we also construct another two matrices X and Y to help the decomposition of $\mathcal{A}_r$. Specifically, as illustrated in Figure 4 A), Y stores the geographical features f_r of each road segment, such as $r.len$, $r.lev$, $r.dir$, $r.n$, the number of neighbors (e.g., r_1 has 2 and 3 neighbors) at its terminals, and a tortuosity ratio τ (e.g. $r_1.\tau = r_1.len/d_1$), as well as the distribution of Point of Interests (POIs) f_p around r's terminals. While Y captures the similarity between different road segments in geographic spaces, matrix X (consisting of X_r and X_h) represents the correlation between different time slots in terms of the coarse-grained traffic conditions. More specifically, we partition a city into disjoint and uniform grids (e.g., 4×4 in Figure 4 B), each of which is comprised of many road segments. X_r is built based on the recent trajectory data received from t_i to t_j (e.g., 1pm-3pm), reflecting the present traffic conditions on a road network. An entry of X_r denotes the number of vehicles traversing a particular grid in a particular time slot. A row of X_r represents coarse-grained traffic conditions in a city of a particular time slot. Consequently, the similarity of two different rows indicates the correlation of traffic flows between two time slots. Additionally, in contrast to using the traffic flow on each individual road segment in $\mathcal{A}_r$, X_r can be filled densely, therefore can help reduce the error of decomposing $\mathcal{A}_r$. X_h has the same structure as X_r, storing the historical average number of vehicles traversing a grid from t_i to t_j. In other words, X_r and X_h

respectively correspond to the coarse-grained current and historical traffic conditions in the same span of time of day. In the implementation, we build A_h and X_h of an entire day in advance and retrieve the entries according to current time (and the number of time slots L needed) when constructing X and A. For example, as shown in Figure 4 C), the rows from t_i to t_j will be retrieved from the prebuilt X_h to construct X with X_r.

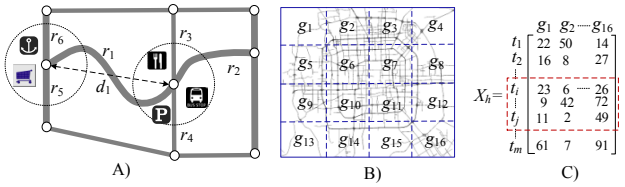


Figure 4. Constructing context matrices

3.2 Tensor Decomposition

To achieve a high accuracy of decomposition, we put together $\mathcal{A}_r$ and $\mathcal{A}_h$(i.e., $\mathcal{A} = \mathcal{A}_r || \mathcal{A}_h$, as shown in Figure 3), decomposing $\mathcal{A}$ with context matrices X and Y collaboratively. The objective function is defined as Equation 3,

$$\mathcal{L}(S,R,U,T,F,G) = \frac{1}{2}\|\mathcal{A}_r - S \times_R R \times_U U \times_T T\|^2 + \frac{\lambda_1}{2}\|X - TG\|^2 + \frac{\lambda_2}{2}\|Y - RF\|^2 + \frac{\lambda_3}{2}(\|S\|^2 + \|R\|^2 + \|U\|^2 + \|T\|^2 + \|F\|^2 + \|G\|^2), \quad (3)$$

where $\mathcal{A} \in \mathbb{R}^{N\times M\times 2L}$ and $X \in \mathbb{R}^{2L\times P}$, P denotes the number of grids; $Y \in \mathbb{R}^{N\times Q}$, Q denotes the dimension of geographical features; $T \in \mathbb{R}^{2L\times d_T}$, $G \in \mathbb{R}^{d_T\times P}$, $R \in \mathbb{R}^{N\times d_R}$ and $F \in \mathbb{R}^{d_R\times Q}$ are low rank latent factor matrices for time slots, grids, roads and geographical features. Later, we can recover $\mathcal{A}$ according to $\mathcal{A}_{rec} = S \times_R R \times_U U \times_T T$. λ_1, λ_2, and λ_3 are parameters controlling the contribution of different parts.

In our model, $\mathcal{A}$ and X shares matrix T, and $\mathcal{A}$ and Y share matrix R. The dense representation of X and Y helps generate a relatively accurate T and R, which reduce the decomposition error of $\mathcal{A}$ in turn. Additionally, the combination of X_r and X_h reveals how current coarse-grained traffic condition deviates from its historical patterns. The information of the deviation is then propagated to $\mathcal{A}$, helping figure out the fine-grained deviation between current traffic conditions and historical traffic patterns on each road segment. So, our model considers both geospatial and temporal correlations. It also incorporates the knowledge from present and historical traffic data. As there is no closed-form solution for finding the most optimal result of Equation 3, we use a numeric method, gradient descent, to find a local optimization, as presented in Figure 5.

Algorithm 1: Tensor Decomposition

Input: tensor $\mathcal{A}$, matrix X, and matrix Y, an error threshold ε
Output: R, U, T, S
1. Initialize $S \in \mathbb{R}^{d_R\times d_U\times d_T}$, $R \in \mathbb{R}^{N\times d_R}$, $U \in \mathbb{R}^{M\times d_U}$, $T \in \mathbb{R}^{2L\times d_T}$, $G \in \mathbb{R}^{d_T\times P}$, $F \in \mathbb{R}^{d_R\times Q}$ with small random values
2. Set η as step size
3. **While** $L_t - L_{t+1} > \varepsilon$
4. **Foreach** $\mathcal{A}_{ijk} \neq 0$
5. $Y_{ijk} = S \times_R R_{i*} \times_U U_{j*} \times_T T_{k*}$;
6. $R_{i*} \leftarrow R_{i*} - \eta\lambda_3 R_{i*} - \eta(Y_{ijk} - \mathcal{A}_{ijk}) \times S \times_U U_{j*} \times_T T_{k*} - \eta\lambda_2(R_{i*} \times F - Y_{i*}) \times F$;
7. $U_{j*} \leftarrow U_{j*} - \eta\lambda_3 U_{j*} - \eta(Y_{ijk} - \mathcal{A}_{ijk}) \times S \times_R R_{i*} \times_T T_{k*}$;
8. $T_{k*} \leftarrow T_{k*} - \eta\lambda_3 T_{k*} - \eta(Y_{ijk} - \mathcal{A}_{ijk}) \times S \times_R R_{i*} \times_U U_{j*} - \eta\lambda_1(T_{k*} \times G - X_{k*}) \times G$;
9. $S \leftarrow S - \eta\lambda_3 S - \eta(Y_{ijk} - \mathcal{A}_{ijk}) \times R_{i*} \otimes U_{j*} \otimes T_{k*}$;
10. $G \leftarrow G - \eta\lambda_3 G - \eta\lambda_1(T_{k*} \times G - X_{k*}) \times T_{k*}$;
11. $F \leftarrow F - \eta\lambda_3 F - \eta\lambda_2(R_{i*} \times F - Y_{i*}) \times R_{i*}$;
12. **Return** R, U, T, S

Figure 5. Algorithm for decomposing a tensor

The Symbol "×" denotes the matrix multiplication; $\times_R$ stands for the tensor-matrix multiplication, where the subscript R stands for the direction, e.g., $H = S \times_R R$ is $H_{ijk} = \sum_{i=1}^{d_R} S_{ijk} \times R_{ij}$; $\otimes$ is the tensor outer product (also called Kronecker product); the entries of the i th row of matrix R are represented as R_{i*}. More specifically, we use an element-wise optimization algorithm (instead of batch decomposition) [10], which updates the factors independently (meaning they can be performed in parallel).

In reality, tensor $\mathcal{A}$ is very large, given hundreds of thousands of road segments and tens of thousands of drivers. Decomposing such a big tensor is very time consuming, therefore reducing the feasibility of our method in providing online services. To address this issue, as illustrated in Figure 6, we partition a city into several disjoint regions, building a tensor for each region based on the data of the region. The matrices X and Y are built in each smaller region accordingly. By setting a proper splitting boundary, we try to keep these small tensors a similar size. As a result, $\mathcal{A}$ is replaced by a few small tensors, which will be factorized in parallel and more efficiently. We validate (in later experiments) that the partition does not compromise the accuracy of the original decomposition when choosing a proper number of partitions.

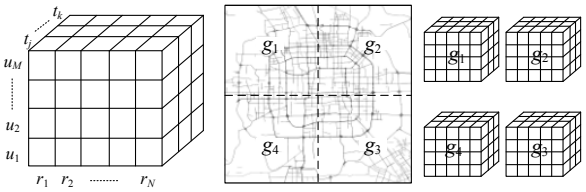


Figure 6. Spatial partition for expediting the tensor decomposition

4. OPTIMAL CONCATENATION (OC)

4.1 Objective Function

Given a path $\boldsymbol{P}$ covered by trajectories, we need to find the best concatenation that results in an accurate travel time estimation. Intuitively, the best decomposition is the one that achieves the lowest empirical risk between the estimate and true travel time $\mu_{\boldsymbol{P}}$. Suppose $\boldsymbol{P}$ is decomposed as $P_1||P_2||\cdots||P_k$, where the estimated travel time is $\bar{t}_{P_1} + \bar{t}_{P_2} + \cdots + \bar{t}_{P_k}$, the squared empirical risk is then wrote as,

$$LSE_{\boldsymbol{P},P_1,P_2,\cdots,P_k} \triangleq E\left(\mu_{\boldsymbol{P}} - \bar{t}_{P_1} - \bar{t}_{P_2} - \cdots - \bar{t}_{P_k}\right)^2, \quad (4)$$

Hence, our problem is to search for the best concatenation which yields the least empirical risk, formally defined as,

$$\text{argmin}_{P_1,P_2,\cdots,P_k} LSE_{\boldsymbol{P},P_1,P_2,\cdots,P_k},$$
$$\text{subject to } P_1||P_2||\cdots||P_k = \boldsymbol{P}. \quad (5)$$

To come up with a computable form of $LSE_{\boldsymbol{P},P_1,P_2,\cdots,P_k}$, we relate $LSE_{\boldsymbol{P},P_1,P_2,\cdots,P_k}$ with $E(\mu_{P_i} - \bar{t}_{P_i})^2$, where μ_{P_i} is the true travel time of sub-path P_i. It is fair to assume if $\boldsymbol{P} = P_1||P_2||\cdots||P_k$, then $\mu_{\boldsymbol{P}} = u_{P_1} + u_{P_2} + \cdots + u_{P_k}$. Hence we have,

$$LSE_{\boldsymbol{P},P_1,P_2,\cdots,P_k} = E\left(\mu_{\boldsymbol{P}} - \bar{t}_{P_1} - \bar{t}_{P_2} - \cdots - \bar{t}_{P_k}\right)^2$$
$$= E\left(\mu_{P_1} + \mu_{P_2} + \cdots \mu_{P_k} - \bar{t}_{P_1} - \bar{t}_{P_2} - \cdots - \bar{t}_{P_k}\right)^2$$
$$= E\left(\sum_{i=1}^{k}\left(\mu_{P_i} - \bar{t}_{P_i}\right)^2 + \sum_{i=1}^{k}\sum_{j=1}^{k}(\mu_{P_i} - \bar{t}_{P_i})(\mu_{P_j} - \bar{t}_{P_j})\right)$$
$$= \sum_{i=1}^{k} E(\mu_{P_i} - \bar{t}_{P_i})^2 + \sum_{i=1}^{k}\sum_{j=1}^{k} E\left((\mu_{P_i} - \bar{t}_{P_i})(\mu_{P_j} - \bar{t}_{P_j})\right)$$

If assuming $\bar{t}_{P_i}$ and $\bar{t}_{P_j}$ are independent, we have $E\left((\mu_{P_i} - \bar{t}_{P_i})(\mu_{P_j} - \bar{t}_{P_j})\right) = E(\mu_{P_i} - \bar{t}_{P_i})E(\mu_{P_j} - \bar{t}_{P_j}) = 0$, Therefore,

$$LSE_{\boldsymbol{P},P_1,P_2,\cdots,P_k} = \sum_{i=1}^{k} E(\mu_{P_i} - \bar{t}_{P_i})^2. \quad (6)$$

Further, $E(\mu_{P_i} - \bar{t}_{P_i})^2 = E(\mu_{P_i} - \frac{1}{n_{P_i}}\sum_{j=1}^{n_{P_i}} t_{P_i,j})^2$

$$= \frac{1}{n_{P_i}^2} E \sum_{j=1}^{n_{P_i}} (\mu_{P_i} - t_{P_i,j})^2 = \frac{1}{n_{P_i}^2} \sum_{j=1}^{n_{P_i}} E(\mu_{P_i} - t_{P_i,j})^2$$

$$= \frac{1}{n_{P_i}} Var(t_{P_i,j}), \quad (7)$$

where n_{P_i} is the number of drivers passing P_i, and $t_{P_i,j}$ denotes the jth driver's travel time on P_i; $Var(t_{P_i,j})$ is the variance of these drivers' travel times. Then, Equation 5 can be represented as:

$$\text{argmin}_{P_1,P_2,\cdots,P_k} \ \sum_{i=1}^{k} \frac{1}{n_{P_i}} Var(t_{P_i,j})$$
$$\text{subject to} \ \ P_1||P_2||\cdots||P_k = \boldsymbol{P} \quad (8)$$

Equation 8 well reflects the aforementioned tradeoff between the support and length of a concatenation. On one hand, it is easier to find more drivers traveling a shorter sub-path. The more the drivers pass a sub-path (i.e. support is higher, n_{P_i} is bigger), the smaller the error of the inferred travel time is. On the other hand, the shorter a sub-path is, the bigger the variance in travel time would be. There are a lot of uncertainties of traveling a short path. E.g., if only traveling one road segment, the travel time will be significantly influenced by a traffic light. As a result, different drivers' travel times could be dramatically different.

4.2 Dynamic Programing Solution

To solve the optimization problem shown in Equation 8, we propose a dynamic programing solution. Suppose a path $\boldsymbol{P}: r_1 \rightarrow r_2 \rightarrow \cdots \rightarrow r_n$, $P' = r_1 \rightarrow r_2 \rightarrow \cdots \rightarrow r_i$, $i < n$, denote $g(P_i) = \frac{1}{n_{P_i}} Var(t_{P_i,j})$, then the optimization problem of P' can be represented as Equation 9.

$$\text{argmin}_{P_1,P_2,\cdots,P_l} \ \sum_{j=1}^{l} g(P_j)$$
$$\text{subject to} \ \ P_1||P_2||\cdots||P_l = P'. \quad (9)$$

Let opt_i be the minimal value of P' to the above problem, then the minimal value of the squared empirical risk function of $\boldsymbol{P}$ is opt_n. Additionally, we have a state transition function as Equation 10.

$$opt_n = \min_{1 \le i < n}(opt_i + g(P_{r_{i+1}||r_{i+2}\cdots||r_n})). \quad (10)$$

Algorithm 2: Query path decomposition

Input: a query path $\boldsymbol{P} = r_1 \rightarrow r_2 \rightarrow \cdots \rightarrow r_n$, a collection of trajectory pattern TPs, a time slot k, trajectories received in k, and tensor $\mathcal{A}_{rec}$
Output: Ψ_n, the most optimal concatenation of path $\boldsymbol{P}$

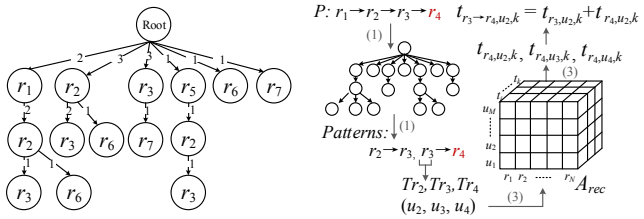
1. $opt_0 \leftarrow 0$, $\Psi_0 \leftarrow \emptyset$;
2. **For** $i = 1$ to n **do**
3. $\quad opt_i \leftarrow +\infty$; $\Psi_i = \emptyset$;
4. $\quad$ **For** $j = i$ down to 1 **do**
5. $\quad\quad P' = r_j \rightarrow r_{j+1} \rightarrow \cdots \rightarrow r_i$; $t_{P',u,k} \leftarrow 0$;
6. $\quad\quad U \leftarrow$ retrieve the drivers traversing (or partially traversing) P' from the trajectory database
7. $\quad\quad$ **Foreach** $u \in U$ **do**
8. $\quad\quad\quad t_{P',u,k} \leftarrow 0$;
9. $\quad\quad\quad$ **Foreach** $r_e \in P'$ not traversed by u's trajectory Tr
10. $\quad\quad\quad\quad t_{P',u,k} \mathrel{+}= (\mathcal{A}_{rec})_{r_e,u,k}$;
11. $\quad\quad\quad t_{P_s',u,k} \leftarrow$ Calculate the time for the rest of P' based on Tr;
12. $\quad\quad\quad t_{P',u,k} \mathrel{+}= t_{P_s',u,k}$;
13. $\quad\quad g(P') = \frac{1}{|U|} Var_{u \in U}(t_{P',u,k})$;
14. $\quad\quad$ **If** $opt_{j-1} + g(P') < opt_i$
15. $\quad\quad\quad opt_i \leftarrow opt_{j-1} + g(P')$;
16. $\quad\quad\quad \Psi_i \leftarrow \Psi_{j-1}||P'$;
17. **Return** Ψ_n;

Figure 7. Algorithm for finding the most optimal concatenation

Using Algorithm 2 shown in Figure 7, we solve this problem with a complexity of $O(n^2 \times m)$, where n is the number of road segments in $\boldsymbol{P}$ and m is the number of drivers passing a segment.

In practice, it is not necessary to check every concatenation of a path, as many sub-paths may not be traversed by any trajectory in the current time slot. To further improve the efficiency of our solution, we mine frequent trajectory patterns from historical trajectories in advance. Then, we just need to check the concatenation of the trajectories patterns. Specifically, we can stop the iteration at Line 4 in algorithm 2 if P' is not a trajectory pattern.

We use a suffix-tree-based algorithm [18] to find the frequent trajectory patterns. Specifically, after being map-matched, a trajectory can be regarded as a string of road segment IDs. By building a suffix tree, where a node denotes a road segment ID, a trajectory is then represented as a path on the tree. For example, the four trajectories shown in Figure 1 can be represented as the tree depicted in Figure 8 A), where $r_1 \rightarrow r_2 \rightarrow r_3$ is the most left path of the tree. $r_2 \rightarrow r_3$ and r_3 are suffixes of $r_1 \rightarrow r_2 \rightarrow r_3$. The number associated with each link stands for the number of the trajectories passing the path (i.e., the support). If setting 2 as a support, we find $r_1 \rightarrow r_2$, $r_2 \rightarrow r_3$, and r_3 are patterns. In reality, the suffix-tree is built based on historical trajectories over a long period of time. As long trajectory patterns are very rare, we set the maximum length of a pattern to 20 road segments.



A) An example of suffix-tree B) Filling in the missing time for a pattern

Figure 8. Mining frequent trajectory patterns and used with tensor

4.3 Working with Tensor $\mathcal{A}_{rec}$

Note that a query path may have some road segments that are not traversed by any trajectory in the current time slot, though these segments may belong to a trajectory pattern (in history). Following the example shown in Figure 1, we demonstrate in Figure 8 B) how $\mathcal{A}_{rec}$ is used with trajectory patterns to help the decomposition of a query path. To estimate the travel time of a query path $P: r_1 \rightarrow r_2 \rightarrow r_3 \rightarrow r_4$ in time slot k, we first search the suffix tree, which was built based on the trajectory data over a long history (not the one shown in the left part of Figure 8 A), for the trajectory patterns that P contains, e.g., $r_2 \rightarrow r_3$ and $r_3 \rightarrow r_4$. To calculate $g(r_3 \rightarrow r_4)$ defined in Equation 9, we need to know the travel time of each driver passing $r_3 \rightarrow r_4$. However, r_4 is not traversed by any trajectory in time slot k. That is $t_{r_3 \rightarrow r_4,u,k}$ is unknown for every driver, though $t_{r_3,u,k}$ can be calculated based on the recently received trajectories, i.e. Tr_2, Tr_3 and Tr_4. To address this issue, we retrieve $t_{r_4,u_2,k}$, $t_{r_4,u_3,k}$, and $t_{r_4,u_4,k}$ from $\mathcal{A}_{rec}$ and calculate $t_{r_3 \rightarrow r_4,u,k}$ for $u = (u_2, u_3, u_4)$, respectively, by Equation 11.

$$t_{r_3 \rightarrow r_4,u,k} = t_{r_3,u,k} + t_{r_4,u,k}, \quad (11)$$

Having $t_{r_3 \rightarrow r_4,u,k}$, we can calculate the most optimal concatenation according to Equation 9, 10 and Algorithm 2. When the supplement of an entry is negative, we resort to the historical average travel time. The dimension of users in tensor $\mathcal{A}_r$ enables us to retrieve a more accurate travel time for a particular driver, resulting in a better estimate of the variance of travel times (as Equation 8). We validate that this is more accurate than just using a historical average of

travel times. After finding the most optimal concatenation, we calculate the travel time of path P by Equation 1, setting Ψ_n as Ψ.

In the implementation, if not building an effective indexing structure, we need to scan a trajectory when calculating the travel time of a path based on the trajectory (i.e., Line 11 of Algorithm 2). This becomes very time consuming if we need to repeat the process many times. To address this issue, we propose an indexing structure to maintain the trajectories received in the current time slot, as shown in Figure 9. The structure looks like the suffix tree we build for mining trajectory patterns. However, each node in the tree stores the ID of the trajectory that traverses the path from the root to the node and the corresponding travel time. For example, Tr_1 and Tr_2 shown in Figure 1 are stored in the index demonstrated in Figure 9, where $t_{r_1 \to r_2 \to r_3}$ stands for the time for traveling path $r_1 \to r_2 \to r_3$.

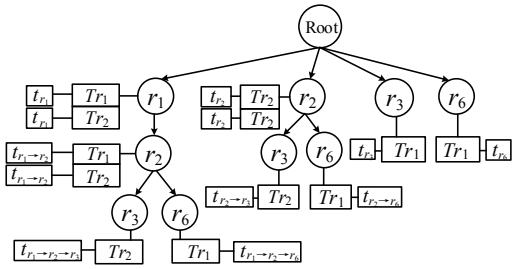


Figure 9. Indexing structure for maintaining recent trajectories

5. EXPERIMENTS

In this section, we evaluate the effectiveness and efficiency of the two major parts of PTTE, i.e., CATD and OC, respectively.

5.1 Settings

5.1.1 Datasets

Taxi Trajectories. We use a GPS trajectory dataset generated by 32,670 taxicabs in Beijing from Sept. 1 to Oct. 31, 2013. The number of GPS points reaches 673,469,757, and the total length of the trajectories is over 26,218,407km. The average sampling rate is 96 seconds per point.

Road networks: We use the road network of Beijing, which is comprised of 148,110 nodes and 196,307 edges. The road network covers a 40×50km spatial range, with a total length (of road segments) of 21,985km.

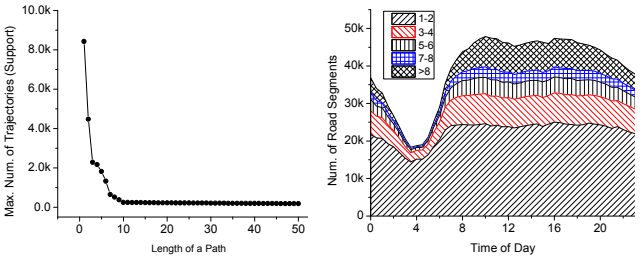
POIs: The dataset consists of 273,165 POIs of Beijing, which are classified into 195 tier two categories. We only chose the top 10 categories that occur around road segments most frequently.

After projecting the trajectories onto the road network, we build $\mathcal{A}_r$ and $\mathcal{A}_h$ to model the recent and historical traffic conditions, respectively. We set four time slots (each time slot is 30 minutes) in the two tensors, as we find 4 have a good tradeoff between effectiveness and efficiency (see Figure 12 A). In the implementation, we remove the road segments that are traveled less than 50 times in two months (i.e., less than 1 time per entire day). The GPS points on such road segments may be noises, or due to the imperfect map-matching algorithm or such places cannot be traveled by vehicles (such as pedestrian streets), and therefore may not be really queried by drivers. Finally, 118,401 road segments are used in our models. We also build geographic and temporal contexts X and Y based on the above mentioned datasets. Table 1 shows the statistics of these tensors and matrices, where the third row means $\mathcal{A}_r$ is partitioned into 25 sub-tensors.

As shown in Figure 10 A), the maximum number of trajectories traversing a path (per day) decreases quickly as the length of the path increases. For instance, a path with 10 road segments is traversed by less than 245 trajectories per day. Figure 10 B) presents the number road segments traveled by taxis with different times. For example, from 8am to 9am, about 25,000 roads segments are traversed by taxis 1-2 times and 8,000 segments (about 4% of Beijing road network) are traveled 3-4 times.

Table 1. Statistics on the data models

	Size	Average non-zero entries
$\mathcal{A}_r$	118,401×32,670×4	0.035%
$\mathcal{A}_h$	118,401×32,670×4	0.4%
$\mathcal{A}_r$/(5×5)	4,736×12,674×4	0.09%
X	8×16	1
Y	118,401×18	1



A) Support of Path W.r.t. it length B) Road segments traversed

Figure 10. Statistics on the trajectory data set

5.1.2 Baseline Methods

We compare PTTE with the following four baseline methods.

1) *Speed-Constraint-based* (SC) method. The travel time of each road segment is computed by the length of a road segment and its speed constraint. The travel time of a path is then a summation of that of each road segment.

2) *Trajectory-based Simple Concatenation* (TSC) method. TSC estimates the travel time of each road segment individually based on the trajectories passing the road segment in the most recent time slot. If a road segment is not covered by any recent trajectory, TSC uses the average historical travel time instead. The travel time of a path is then a summation of that of each road segment.

3) *Optimal Concatenation with Historical Travel Time* (OC+H) method. OC+H uses Algorithm 2 proposed in this paper to find the most optimal concatenation of trajectories to estimate the travel time of a path. For a road segment not covered by any trajectories, this method uses an average historical travel time of the road segment.

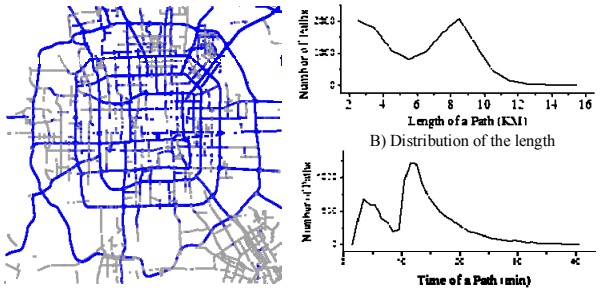
4) *Optimal Concatenation with Nonnegative Matrix Factorization* (OC+MF). This method is similar to OC+H, except for using MF (rather than historical average) to infer the travel time of segments without trajectories. MF is applied to the road-time matrix, which is degraded from tensor $\mathcal{A}$ by averaging the driver dimensions.

Comparing PTTE with the first baseline, we can demonstrate the advantages of using the trajectory data. The second baseline method can reveal the contribution of the proposed optimal concatenation algorithm. Through comparing it with the third and fourth baselines, we can justify why a tensor with the driver dimension is needed in PTTE. Regarding CATD, the first step of PTTE, we study the contribution of using context matrices and historical data, respectively, in filling the missing values of $\mathcal{A}_r$.

5.1.3 Query Paths and Ground Truth

We randomly pick out 50 paths, each of which has been fully traversed by at least two drivers, in each hour of a day, from the taxi trajectory data. We then use these paths as queries and the

average travel time of these trajectories as the ground truth. Once a trajectory is selected as a ground truth, we remove them from the training data. In total, we generate 12,384 queries, whose length ranges from 2KM to 16KM following the distribution as shown in Figure 11 B) and time span follows the distribution depicted in Figure 11 C). The total length of these query paths is 76,412.6km with an effective total time span of 2733.9hours. The queries selected in this way cover different times of day (from 6am to 11pm) and follow the distance distribution of people's true travel patterns in a city's road network. In the experiments, we do not study the query after 11pm and earlier than 6am, as there is almost no traffic at those times. In other words, people can travel as fast as the speed constraint of a road. According to Figure 11 A), these query paths (denoted as blue road segments) also cover the majority of the Beijing road network.



A) Geographical distribution B) Distribution of the length C) Distribution of time length

Figure 11. Distributions of the query paths

We study the mean absolute error (MAE) and mean relative error (MRE) of an estimate according to Equation 12 and 13, respectively, where y_i is an estimate and $\hat{y}_i$ is the ground truth.

$$MAE = \frac{\sum_i |y_i - \hat{y}_i|}{n}, \qquad (12)$$

$$MRE = \frac{\sum_i |y_i - \hat{y}_i|}{\sum_i \hat{y}_i}, \qquad (13)$$

5.2 Results

5.2.1 Performance of CATD

To test the accuracy of estimating the travel time of road segments absent of trajectories, we randomly remove 30% of non-zero entries from the last time slice of tensor $\mathcal{A}_r$. We then infer these entries with our method, using their original values as a ground truth to calculate MAE and RMSE, as shown in Table 2 (λ_1=λ_2=λ_3= 0.01). RMSE is widely used to measure the error of a tensor decomposition, defined as Equation 14.

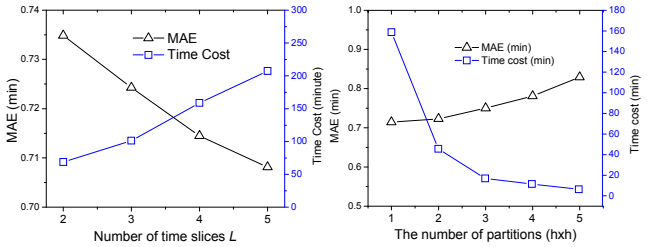
$$RMSE = \sqrt{\frac{\sum_i (y_i - \hat{y}_i)^2}{n}}; \qquad (14)$$

Adding historical trajectories (i.e.., $\mathcal{A}_h$) and context matrices gradually into the tensor decomposition, denoted as TD+H and TD+H+C, respectively, we achieve a clear increase of performance in estimating the missing values.

Table 2. The performance of CATD

	MAE (min)	RMSE
TD	0.747	1.646
$TD + H$	0.732	1.629
$CATD\ (TD + H + C)$	0.714	1.613

We further study the effectiveness and efficiency of CATD changing over the number of time slices in Figure 12 A). Adding more time slices into $\mathcal{A}_r$ and $\mathcal{A}_h$ achieves a more accurate estimate of missing values (i.e., the travel time of road segments without trajectories), as we have richer information of previous traffic conditions. On the other hand, more time slices consume a longer time. We find that formulating a tensor with 4 time slices is a relatively strong tradeoff between effectiveness and efficiency.



A) W.r.t. time slices B) w.r.t. partitions

Figure 12. Performance of CATD

However, the time cost for decomposing such big tensors and matrices (see Table 1) makes our method impractical for answering instant queries. As a consequence, we partition a city into disjointed regions, building individual tensors and matrices, separately and in parallel. Figure 12 B) shows the performance changing over the number of partitions. When dividing Beijing into 5×5 regions, we can finish CATD for each region in 6 minutes. The partition does not compromise the accuracy (only a 0.1min gap) as compared to decomposing the original tensors. Additionally, the tensor decomposition method we adopt can be performed in parallel, further reducing the time cost of CATD.

5.2.2 Overall Performance

Table 3 presents the overall performance of our model PTTE (i.e., CATD+OC) and baseline methods. Besides MAE and MRE, we also present the average error of travel time per km (MAE/L). For example, the average error of PTTE in estimating the travel time of a path is about 0.412min (25 seconds) per kilometer. Clearly, PTTE outperforms all the baselines in terms of the three metrics. Given the queries introduced in Section 5.1.3, on average, the absolute error of the estimated travel time is about 2 minutes per path, which is about 19% of the true travel time. From the results, we can draw the following conclusions. First, using trajectory data (TSC) is much better than that only based on speed constraints of roads. But, simply concatenating the travel time of each individual road segment (TSC) is not an optimal solution. As we mentioned before, the more fragments involved in a concatenation, the more uncertain of travel time for crossing two consecutive fragments occurs is. Second, regarding estimating the travel time of road segments without covered by trajectories, using the historical average travel time (OC+H) is significantly better than using its speed constraint. Using a collaborative filtering model (OC+MF) is slightly better than using an historical average. The PTTE considering the driver dimension is even better than OC+MF.

Table 3. Comparison of different methods

	MAE (min)	MRE	MAE/L (min/km)
SC	8.808	0.665	1.428
TSC	5.244	0.396	0.850
$OC + H$	3.245	0.245	0.526
$OC + MF$	3.061	0.231	0.496
PTTE	**2.545**	**0.192**	**0.412**

Figures 13 and 14 present the performance of SC, TSC, and PTTE changing over time of day, on weekdays and weekends. We do not show that of OC+H and OC+MF, as they almost follow the same trend as (but worse than) PTTE. As depicted in Figure 11, the error of SC increases tremendously during peak traffic hours, around 8-9am and 6-7pm. The complex traffic conditions on roads at these moments deviate the true travel speed far from their speed constraints. When time goes to late night (i.e., no complex traffic

conditions anymore), the error decreases and approaches our PTTE method. This is also the reason that we do not need to perform PTTE on late night time sets, especially after 12am and before 6am. Simply estimating the travel time for each individual road segment and then doing a summation (TSC) is not accurate enough. When there are not enough taxis traveling on Beijing's road network, e.g., after 9pm and before 8am, MAE of TSC is higher than other time slots. With the help of CATD, the MAE of our method only increases slightly after 9am. This demonstrates the value of inferring the missing values.

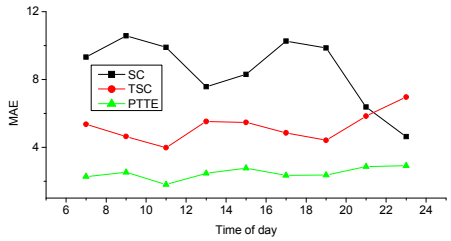


Figure 13. Performance changing over time of day (Weekday)

As most people do not drive to work on weekends, we observe different trends of the performance in Figure 14, in contrast to Figure 13. SC's MAE reaches its peak around 6pm which is the weekend rush hour. The reason why PTTE has the biggest error around 9am on weekends is a tradeoff between the number of taxis and the complexity of traffic conditions in a road network. On weekends, the number of taxis traveling in Beijing's road network is still very small at 9am, while the traffic conditions start becoming complex. As time goes by, more taxis are present on Beijing's road network, alleviating the data sparsity problem and decreasing the MAE. Conversely, before 8am, the traffic condition is still very simple to predict on roads. The travel speed is almost close to the speed limit.

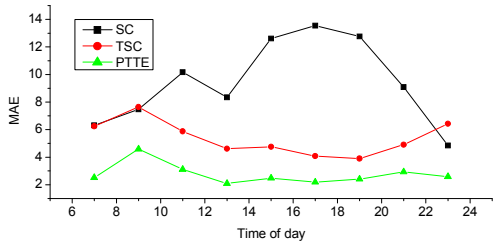


Figure 14. Performance changing over time of day (Weekend)

Figure 15 shows the performance of PTTE changing over the length of a query path. As the length increases, both MAE and MRE decrease. A longer path is more likely to contain more trajectory patterns, which provide more choices for an optimal concatenation. This also echoes the assumption we proposed in the introduction. The shorter a sub-path is the more unstable its travel time could be. In an extreme case, the travel time of a single road segment is terrifically impacted by traffic lights and pedestrians crossing it. So, the travel time varies in time quickly and tremendously, becoming hard to predict.

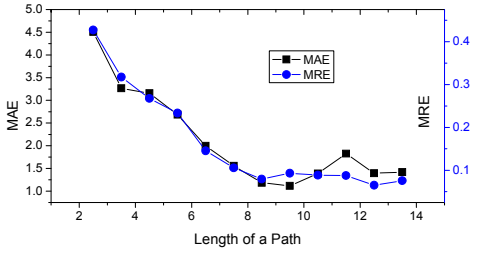


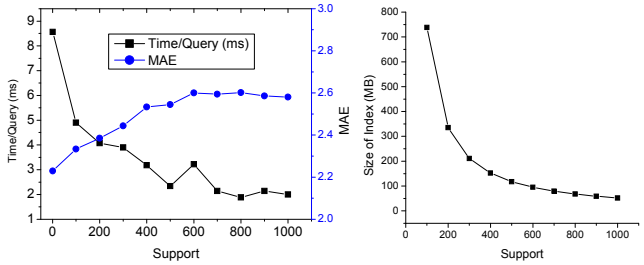
Figure 15. Performance of PTTE changing over the length of a path

Table 4 presents the time and space cost of PTTE's two major components for processing the trajectories received in recent 30 minutes. The first component CATD is only performed once for the entire dataset, while OC is conducted for each query path. All numbers are obtained by only using a single core of a server with 2.80GHz Xeon CPU and 24GB RAM. For example, if we partition a city into 25 regions, building $\mathcal{A}_r$ and $\mathcal{A}_h$ for an individual region needs 44 and 233 seconds, respectively, costing 4.4MB and 14.6 MB of memory, respectively (each tensor contains 4 time slices). Using Algorithm 1 to decompose these tensors with context matrices X and Y needs about 6.31min for each region. We do not list the time for building matrix Y, as it is static and can be built offline. In total, we can infer the travel time on each road segment for each particular driver within 6.4min if using 25 cores in a server. Note that Algorithm 1 is based on an entry-wised decomposition approach, which can be performed in parallel further. If using six 25-core servers, we can finish CATD in about 1 minute. In the optimal concatenation, we can process a query path in 2.3ms. However, the time is much longer if the trajectory patterns and the real-time indexing structure proposed in Figure 9 are not used.

Table 4. Time and memory cost for each step of PTTE

	Components		Time	Memory (MB)
Deal with missing values ($CATD$)	Building matrix X, Y		34ms	9
	Tensor construction	$\mathcal{A}_r$	44ms	4.4
		$\mathcal{A}_h$	233ms	14.6
	Decomposition	5×5	6.31min	116
	Total		6.4min	144
Optimal Concatenation (OC)	Best OC		**2.3ms**	**995**
	w/o trajectory patterns		8.6ms	877
	w/o index		12.2s	714

Figure 16 A) shows the tradeoff between effectiveness (measured by MAE) and efficiency (by time cost) of using trajectory patterns in optimal concatenations, where the horizontal axis denotes the threshold of support. For example, if setting 400 as a threshold, we regard a path as a trajectory pattern if the path has been covered by over 400 trajectories in the two-month dataset. The bigger threshold is the smaller number of trajectory patterns we can obtain. So, the time cost decreases and the MAE increases, as the support increases. Additionally, as depicted in Figure 16 B), setting a smaller support threshold leads to a bigger size of the suffix-tree-based index for trajectory pattern mining. In the implementation, we find 500 is a good tradeoff among MAE, time cost, and index size. Note that without using the indexing structure proposed in Section 4.3, we need about 12 seconds to find the most optimal concatenation for a query path (see Table 4).



A) Time cost and MAE w.r.t. support B) Size of the index w.r.t. support

Figure 16. Performance of OC w.r.t. support of a trajectory pattern

The recovered tensor $\mathcal{A}_{rec}$ helps the optimal concatenation significantly. In the experiments, we test our model with 12,384 query paths, which are comprised of 217,326 road segments in total. When processing the query paths, we access tensor $\mathcal{A}_{rec}$ 1,706,648 times (i.e., 137.8 times per query) to retrieve the missing travel time of a road segment in a trajectory pattern. The travel times of 58,223

road segments (about 26.8% of the road segments in the query paths) are finally retrieved from $\mathcal{A}_{rec}$ for constructing the most optimal concatenation, i.e., 4.7 road segments per path.

We study the performance of PTTE changing over different number of vehicles in Figure 17, aiming to figure out how many GPS equipped vehicles are needed to have accurate results. For example, using 30,000 GPS-equipped vehicles in Beijing is enough to achieving a MRE smaller than 0.23. In other words, if having 1.36 vehicles on every kilometer road in a city, we can achieve a relative error of estimation smaller than 0.23.

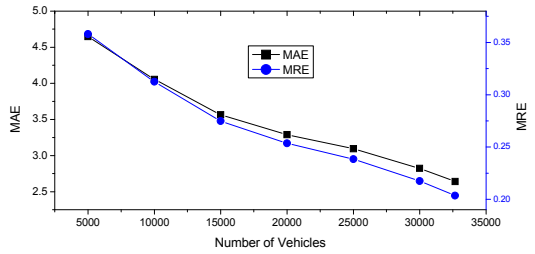


Figure 17. Performance w.r.t the number of taxis

5.2.3 Field Study

Besides using GPS trajectories from taxi drivers, we also send two drivers carrying a GPS logger to test the accuracy of our method from Sept. 1 to Oct. 30. Finally, we collect 114 driving paths with a total length of 999.4km and an effective total time period of 61.6hours. The sampling rate of these GPS trajectories is 5 seconds per point. Table 5 shows the performance of the study, where we observe a better MRE than using taxi drivers' trajectories as a ground truth. The major reason is the length of a path collected in the study is usually long (on average 8.78KM each), where our model has a better accuracy than a shorter path. Additionally, the map-matching for high sampling rate trajectories is more accurate than low sampling rate taxi trajectories, resulting in a more accurate estimation of the ground truth.

Table 5. Performance of the in-the field study

	MAE (min)	MRE	MAE/L (min/km)
SC	18.193	0.561	2.075
TSC	11.300	0.349	1.289
$OC + H$	4.990	0.154	0.569
$OC + MF$	4.052	0.125	0.462
PTTE	**3.771**	**0.116**	**0.430**

6. RELATED WORK

6.1 Road Segment-Based Travel Time

Approaches using Loop Detectors: Estimating travel time based on loop detectors installed on both endpoints of a road segment has been studied intensively over the past few decades. When a vehicle passes through, the time interval for crossing two adjacent loop detectors is recorded, based on which the speed of the vehicle is inferred. [9, 14, 16] use various models to estimate the travel speed on an individual road segment based on the sensor readings from loop detectors, and then convert the speed into a travel time. [19] predicts the travel time of a road segment by applying support vector regression to its historical travel times. As many roads do not have a loop detector buried, this category of research mainly focuses on individual road segments, and therefore is difficult to scale up to an entire city.

Floating-Car-Data Approaches: Learning city transportation using floating car data has gained more attention recently [1, 5, 17, 23, 24]. In these approaches, cars driven in a city serve as dynamic sensors to probe traffic conditions, and their GPS trajectories are used to compute the speed and travel time on road segments. Most methods infer the travel time of an individual road segment without considering the correlation between the traffic conditions on different roads. This reduces the accuracy of an inference in an urban environment where traffic conditions are inter-related.

Some models [2] predict the travel speed of a road segment by considering the traffic patterns of other road segments connected to it. Unfortunately, when scaling up to an entire city, these methods often result in a model with high complexities. Additionally, they do not tackle the data sparsity problem, i.e. many road segments are not traveled by trajectories in the current time slot, which is quite common in reality. The neighboring segments' traffic patterns can be regarded as the local correlation between road segments. However, the correlation between road segments that are not geospatially connected is not considered in these modes. [8] aims to estimate the travel time between two points on a road network using low sampling rate trajectory data. It considers the correlation between different road segments in terms of their historical traffic patterns to infer the travel time on a road segment and the delay at intersections. The model is trained using a Maximum Likelihood Estimation over the collected data in an urban road network.

However, these methods still follow the idea of first estimating the travel time of individual road segments and then summing up the travel times of the road segments belonging to a path. As we mentioned before, it is difficult to explicitly model the complex factors for crossing two road segments, e.g., intersections, direction turns, and traffic lights. Though we also infer the travel time for individual segments, the time is combined with trajectory patterns to formulate a sub-path rather than simply concatenating them one by one. The variance of different users' travel times also captures the complexity of traffic conditions on a road segment or a sub-path. In the meantime, when inferring the travel time of a road segment, we incorporate both spatial correlation between different road segments and the temporal correlation between the traffic conditions of different time slots, as well as the deviation between current traffic conditions and historical traffic patterns.

6.2 Path-Based Travel Time

A possible approach to deal with the weakness of the individual road segment-based methods is to estimate the travel time of a path as a whole based on frequent trajectory patterns. For example, we can mine frequent patterns from historical trajectories [6, 7, 12] in advance, and then use the average travel time of a pattern to represent the travel of the path corresponding to the pattern. Some models can also be built based on the historical data of a path [15] to estimate the future travel time of the path. This approach needs a balance between the coverage of queries that it can answer and the accuracy of the inferred travel time. To be able to answer various query paths, these methods need to select more trajectory patterns by using a small support. However, the travel time derived from a small support is not accurate. Additionally, a path's travel time of current time slot may deviate from its historical average significantly, depending on the real-time traffic conditions. Moreover, many query paths may not be traversed by any trajectories in current time slot as well in the history.

Recent research has started finding more optimal concatenations of road segments to estimate the travel time of a path. A series of research attempts to explicitly calculate the time spent on intersections using an interpolation method [11], or a joint probability model [3], or a dynamic Bayesian network [4]. This could result in a more accurate summation of individual road segments' travel times. However, these methods do not study how

to leverage sub-trajectories to construct an optimal estimation of a path. As we mentioned before, the accuracy of estimating a sub-path's travel time is subject to the tradeoff between its length and the number of trajectories passing the sub-path. In our model, we propose and prove an objective function that can represent the tradeoff. We also consider the variance of different drivers' travel times, resulting in a more accurate travel time estimation of a path.

Based on the trajectories generated by a large number of taxis, [20] builds a landmark graph, where a node (entitled a landmark) is a road segment frequently traveled by taxis and an edge denotes the aggregation of taxis' commutes between two landmarks. The travel time of a path is then approximated by the summation of the travel times between landmarks. Though the proposed landmark graph can also deal with the data sparsity problem, the main goal of [20] is to find the quickest driving path between an origin and a destination; this is different from our problem. Knowing the shortest time for traveling between two points does not mean we can obtain the travel time of any path traversing the two points.

7. CONCLUSION

In this paper, we propose a real-time and citywide model, called PTTE, to estimate the travel time of a path in current time slot in a city's road network, using the GPS trajectories from a sample of vehicles (e.g. taxicabs). Though this is a very important foundation for many traffic monitoring and routing systems, the problem has not been well solved given three challenges: 1) data sparsity, 2) finding an optimal combination of trajectories (i.e., the tradeoff between the length of a sub-path and the number of trajectories passing the sub-path), and 3) the tradeoff between scalability, effectiveness and efficiency. PTTE is comprised of two major components, CATD and OC. The former infers the travel time of a road segment without being traversed by trajectories in the current time slot through a context-aware tensor decomposition approach. The latter searches for the most optimal concatenation of trajectories for a query path using a dynamic programing solution. We evaluate PTTE with extensive experiments based on GPS trajectories generated by over 32,000 taxicabs over a period of two months in Beijing. We test the effectiveness and efficiency of CATD and OC, respectively. First, the results demonstrate the advantages of CATD in accurately filling in the missing values beyond baseline methods, such as using speed constraints, or using a historical average travel time, or using a matrix factorization-based approach. The driver dimension in tensor $\mathcal{A}_r$ helps us calculate the variance of different drivers' travel times on a road segment. The variance indicates the complexity of a road's traffic condition, helping us find the most optimal concatenation for a path. In addition, the geospatial/temporal contexts and historical traffic patterns increase the accuracy of estimating the missing values. Regarding the most optimal concatenation, we devise an objective function which has been proved to be able to model the tradeoff between a sub-path's length and the number of trajectories passing it. Tested by 12,384 query paths, PTTE achieves a mean absolute error of 0.4min per km, which is about 19% of the truth travel time. The results of the in-the-field study have an even smaller estimation error (11.6%). Using the suffix-tree-based indexing structure to manage the trajectories received currently and the trajectory patterns (mined in advance) to scale down the concatenation candidates, on average, we are able to infer the travel time of a path in 2.3ms. The codes and a sample of the data used here have been released at [25].

In the future, we plan to infer the travel time of a path for a particular driver. In addition, we would like to study the impact of other factors, such as weather conditions and air quality, on the travel time estimation of a path.

8. REFERENCES

[1] S. Chawla, Y. Zheng, J. Hu. 2012. Inferring the Root Cause in Road Traffic Anomalies. In *Proc. of IEEE ICDM 2012*.

[2] C. De Fabritiis, R. Ragona, G. Valenti. 2008. Traffic estimation and prediction based on real time floating car data. In *Proc. of IEEE ITSC 2008*.

[3] A. Hofleitner, A. Bayen. 2011. Optimal decomposition of travel times measured by probe vehicles using a statistical traffic flow model. In *Proc. of IEEE ITSC 2011*.

[4] A. Hofleitner, R. Herring, P. Abbeel, A. Bayen. 2012. Learning the dynamics of arterial traffic from probe data using a dynamic Bayesian network. *IEEE Trans. on Intelligent Transportation Systems*, 13(4), 1679-1693.

[5] B. Hull, V. Bychkovsky, Y. Zhang, K. Chen, M. Goraczko, A. Miu, E. Shih, H. Balakrishnan, S. Madden. 2006. CarTel: a distributed mobile sensor computing system. In *Proc. of ACM Sensys 2006*.

[6] J. Han, M. Kamber, J. Pei. 2006. Data mining: concepts and techniques. *Morgan kaufmann*.

[7] J. Han, J. Pei, Y. Yin. 2000. Mining frequent patterns without candidate generation. *ACM SIGMOD Record*, 29(2), 1-12.

[8] E. Jenelius, H. N. Koutsopoulos. 2013. Travel time estimation for urban road networks using low frequency probe vehicle data. *Transportation Research Part B: Methodological*, 53, 64-81.

[9] Z. Jia, C. Chen, B. Coifman, P. Varaiya. 2001. The PeMS algorithms for accurate, real-time estimates of g-factors and speeds from single-loop detectors. *IEEE Trans. on Intelligent Transportation Systems*.

[10] A. Karatzoglou, X. Amatriain, L. Baltrunas, N. Oliver. 2010. Multiverse recommendation: n-dimensional tensor factorization for context-aware collaborative filtering. In *Proc. of ACM Recsys 2010*.

[11] T. V. Larsen. Travel-Time Estimation in Road Networks Using GPS Data. White paper.

[12] W. Luo, H. Tan, L. Chen, L. M. Ni. 2013. Finding time period-based most frequent path in big trajectory data. In *Proc. of ACM SIGMOD 2013*.

[13] S. Ma, Y. Zheng, O. Wolfson. 2013. T-share: A large-scale dynamic taxi ridesharing service. In *Proc. of IEEE ICDE 2013*.

[14] K. F. Pettya, P. Bickelb, J. Jiangc, M. Ostlandb, J. Riceb, Y. Ritovd, F. Schoenbergb. 1998. Accurate estimation of travel times from single-loop detectors. *Transportation Research Part A: Policy and Practice*, 32(1), 1-17.

[15] M. Rahmani, E. Jenelius, H. N. Koutsopoulos. 2013. Route travel time estimation using low-frequency floating car data. In *Proc. of IEEE ITSC 2013*.

[16] J. Rice, E. Van Zwet. 2004. A simple and effective method for predicting travel times on freeways. *IEEE Trans. on Intelligent Transportation Systems*, 5(3), 200-207.

[17] R. Sevlian, R. Rajagopal. 2010. Travel Time Estimation Using Floating Car Data. arXiv preprint arXiv:1012.4249.

[18] R. Song, W. Sun, B. Zheng, Y. Zheng, C. Tu, S. Li. 2014. PRESS: A Novel Framework of Trajectory Compression in Road Networks. In *Proc. of VLDB 2014*.

[19] C. H. Wu, J. M. Ho, D. T. Lee. 2004. Travel-time prediction with support vector regression. *IEEE Trans. on Intelligent Transportation Systems*, 5(4), 276-281.

[20] J. Yuan, Y. Zheng, C. Zhang, W. Xie, X. Xie, G. Sun, Y. Huang. 2010. T-Drive: Driving Directions Based on Taxi Trajectories. In *Proc. of ACM SIGSPATIAL 2010*.

[21] J. Yuan, Y. Zheng, C. Zhang, X. Xie, G. Sun. 2010. An interactive-voting based map matching algorithm. In *Proc. of IEEE MDM 2010*.

[22] N. J. Yuan, Y. Zheng, L. Zhang, X. Xie. 2013. T-finder: A recommender system for finding passengers and vacant taxis. *IEEE Trans. on Knowledge and Data Engineering*, 25(10), 2390-2403.

[23] Y. Zheng, Y. Chen, Q. Li, X. Xie, W. Y. Ma. 2010. Understanding transportation modes based on GPS data for web applications. *ACM Trans. on the Web*, 4(1), 1.

[24] Zheng, Y., Capra, Li, Wolfson, O., Yang, H. 2014. Urban computing: concepts, methodologies, and applications. *ACM Trans. on Intelligent systems and Technology*, 5(3).

[25] Data released: http://research.microsoft.com/apps/pubs/?id=217493