Prediction-based Online Trajectory Compression

Arlei Silva †, Ramya Raghavendra ∗, Mudhakar Srivatsa ∗, Ambuj K. Singh †
†Computer Science Department, University of California, Santa Barbara, CA, USA
∗IBM T.J. Watson Research Center, Yorktown Heights, NY, USA
{arlei,ambuj}@cs.ucsb.edu,{rraghav,msrivats}@us.ibm.com

ABSTRACT
Recent spatio-temporal data applications, such as car-sharing and smart cities, impose new challenges regarding the scalability and timeliness of data processing systems. Trajectory compression is a promising approach for scaling up spatio-temporal databases. However, existing techniques fail to address the online setting, in which a compressed version of a trajectory stream has to be maintained over time. In this paper, we introduce ONTRAC, a new framework for map-matched online trajectory compression. ONTRAC learns prediction models for suppressing updates to a trajectory database using training data. Two prediction schemes are proposed, one for road segments via a Markov model and another for travel-times by combining Quadratic Programming and Expectation Maximization. Experiments show that ONTRAC outperforms the state-of-the-art offline technique even when long update delays (4 min.) are allowed and achieves up to 21 times higher compression ratio for travel-times. Moreover, our approach increases database scalability by up to one order of magnitude.

1. INTRODUCTION
Spatio-temporal data processing has become an important challenge in data mining research due to the proliferation of data sources and Geographical Information Systems (GIS). A key driver of recent advances in this field is the need for improving transportation in rapidly growing cities. For instance, vehicle tracking [4], taxi-dispatching [18], parking space location [25], routing [27], and similar applications [11] have set new requirements regarding the scalability, timeliness, and modeling for spatio-temporal data analytics.

Online Trajectory compression is a promising approach for enabling the processing of trajectory data at a scale that meets the demands of modern applications. However, existing work [5, 16, 23] fails to address the online setting, in which an accurate compressed version of a trajectory stream has to be maintained as new updates arrive. For instance, car-sharing apps, such as Uber and Lyft, have to handle online trajectory data at a massive scale in order to perform user-driver assignment, improve routing algorithms and protect both drivers and passengers’ safety. Moreover, megacities like Beijing are investing in urban computing as means to reduce congestions and air pollution. An important aspect shared by all these applications is that trajectories can be map-matched, since vehicles move mostly over a road structure. Existing work has shown that we can make use of such knowledge to improve trajectory data processing [5].

Figure 1 illustrates an online trajectory compression application scenario based on the so called smart cities. Cab trajectory data is collected using GPS devices and stored in a trajectory DBMS in the cloud. A cab updates its trajectory in the database by sending its (lat-long) coordinates every minute. Recent GPS updates are map-matched to the road network structure and then processed by an online trajectory compression framework, which inserts a compressed version of the trajectory into the database. Several applications query the DBMS for different purposes, including a cab-dispatching app, the cab company, and a smart city control center. Each query is augmented by the trajectory compression framework as means to become compatible with the compressed data schema. The retrieved results are then uncompressed by the framework before they are sent to the applications. For instance, the smart city control center might compare recent cab speeds against historical data to detect traffic jams. In large cities, with tens of thousands of cabs, managing these trajectory updates becomes a challenge.

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. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright 20XX ACM X-XXXXXX-XX-X/XX/XX ...$15.00.
The natural solution to the online trajectory compression problem is to extend state-of-the-art offline schemes using a caching mechanism with bounded delays. If delays are short (e.g., in the order of one minute), the trajectory database might still answer queries in an "almost online" fashion. However, our experiments show that the compression ratio achieved by an offline approach degrades drastically as the size of the accepted delay window decreases.

In this paper, we introduce a new framework for online trajectory compression called ONTRAC (ONLine TRAjectory Compression). ONTRAC’s design started from the following open question: Can we apply trajectory prediction as means to suppress online updates to a large trajectory database? Our approach decomposes trajectories into two components — spatial and temporal trajectory information — for which different prediction algorithms are proposed. Accurate predictions in both of these dimensions translate into an effective online trajectory compression. We show one example of a real cab trajectory compressed using ONTRAC in Figure 1. The compression ratio achieved for this trajectory is 19.6 (i.e., 95% of the updates are suppressed).

The motivation for ONTRAC comes from information theory, where, dictionary-based compression (e.g., Huffman coding) requires knowledge of the complete string to be compressed but prediction-based schemes (e.g., prediction by partial matching) suppresses new symbols using a prediction model. However, in typical trajectory compression applications, compressed data is stored into a database with query processing capabilities. Information theory is mostly focused on bit string representations that cannot be processed by existing database technology.

ONTRAC compresses spatial trajectory information, i.e., the sequence of road segments visited by an object, by predicting segment updates using a Markov model. For temporal compression, ONTRAC combines Quadratic Programming (QP) and Expectation-maximization (EM) to learn a prediction model for travel-times. Our approach handles the sparseness and noise of GPS updates, a problem we call temporal trajectory inference, by computing maximum-likelihood estimates for missing temporal trajectory information. Such estimates are based on a Gaussian model for travel-times that is fitted using a convex QP formulation. We illustrate how data compressed using ONTRAC can be efficiently queried for a typical historical query that asks where a given vehicle was at a given instant of time.

We summarize our main contributions as follows:

- We introduce the online map-matched trajectory compression problem in databases, which consists of minimizing the number of updates in a trajectory stream.
- We present ONTRAC, an online trajectory compression framework that supports lossless spatial compression and error-bounded temporal compression.
- We demonstrate the effectiveness of ONTRAC using real datasets. While our approach is fully online, the baseline requires long update delays (4 min.) to achieve the same spatial compression ratio. Moreover, ONTRAC obtains up to 21 times higher compression ratio for temporal trajectory information. We also show how ONTRAC’s compression increases database scalability by up to one order of magnitude.

2. **ONLINE TRAJECTORY COMPRESSION**

A road network is a directed graph $G(V, E)$ where $V$ is a set of road segments and $(s_i, s_j) \in E$ iff an object can navigate from $s_i$ to $s_j$ without passing through any other segment in $V$. A GPS trajectory is a sequence of triples $((x_1, y_1, t_1), (x_2, y_2, t_2), ...)$, where $x_i$ is a longitude, $y_i$ is a latitude and $t_i$ is a timestamp. GPS trajectories can describe any movement on Earth’s surface. However, this paper assumes that trajectories are constrained to the road network $G$, thus we can represent a GPS trajectory as a map-matched trajectory $< (s_1, t_1), ..., (s_m, t_m) >$, where $s_i$ in $V$ and $t_i$ is the time when $s_i$ was traversed.

Since GPS updates are sparse, map-matched trajectories have missing temporal information. An update $(s_i, t_i)$ for which the timestamp $t_i$ is unknown is called a missing update. The temporal trajectory inference problem consists of inferring such information based on the remaining updates.

**Definition 1. Temporal Trajectory Inference Problem:** Given a map-matched trajectory $T = < (s_1, t_1), (s_2, t_2), ... >$ $< (s_m, t_m) >$ with missing temporal information, compute estimates for the timestamps $t_i$ of the missing updates $(s_i, t_i)$.

A map-matched trajectory stream is a sequence of updates $(o_i, s_i, t_i)$ ordered by $t_i$, where $o_i$ is an object, $s_i$ is a road segment and $t_i$ is a timestamp. Nevertheless, for sake of simplicity, our algorithms will be described in terms of aggregated trajectories. Each object $o_i$ has an associated trajectory $T_i = < (s_1, t_1), ..., (s_m, t_m) >$ that aggregates all its updates $(o_i, s_1, t_1), ..., (o_i, s_m, t_m)$. Notice that these trajectories might have missing temporal information. A trajectory stream $D$ is a set of trajectories $\{T_1, ..., T_n\}$ that are processed in an online manner (i.e., one update at a time).
is a sequence of road segments \(< s_1, \ldots, s_m >\) and the temporal information \((T.temporal)\) is a sequence \(< (d_1, t_1), \ldots, (d_m, t_m) >\) where \(d_i\) is a distance and \(t_i\) is a timestamp. Such a decomposition is motivated by the fact that spatial and temporal trajectory information have different properties, while the spatial information is discrete (road segments), the temporal is continuous (travel-times).

Sections 3.1 and 3.2 describe ONTRAC’s spatial and temporal compression strategies, respectively. Although the same general approach is applied for both cases — learning a model to predict future updates based on past data — the problem becomes more challenging when we move from the spatial to the temporal domain. Section 3.3 discusses how to query trajectory data compressed using ONTRAC.

Proofs for all the theorems presented in this section can be found in the Appendix.

### 3.1 Spatial Compression

ONTRAC compresses the spatial information from trajectories using a Markov model learned during a training phase. The model is applied to predict the next update (i.e. road segment) for a given trajectory given its past updates. Compressed trajectories contain those segments that are not predicted by the model. This approach leads to a lossless compression, since any trajectory can be fully recovered as a combination of its compressed form and the model.

A \(k\)-order Markov trajectory model \(\Psi\) for predicting the most likely next segment update \(s\) for a given spatial trajectory \(S\) is learned from training data \(D_t\):

\[
\Psi(S, D_t) = \arg \max_{s} P(s|S')
\]

where \(S'\) is the longest prefix of \(s\) in \(D_t\) such that \(|S'| \leq k\) and \(P(s|S')\) is the probability of \(s\) to happen given \(S'\) in \(D_t\). A Trie data structure is applied as a representation for \(\Psi\). Each node \(nd\) in \(\Psi\) corresponds to a sequence of road segments and has a vector \(count\) that keeps track of the frequency of next segments and a variable \(pred\) set to the most frequent next segment for the corresponding sequence.

Algorithm \(\mathbf{6}\) describes the algorithm \((\text{Spatial-training})\) for spatial compression. It receives the road network \(G\), the training trajectories \(D_t\), and the order \(k\) as parameters and returns a \(k\)-order Markov model \(\Psi\) for trajectory compression. For each trajectory \(T\) in \(D_t\), the algorithm extracts its spatial information \(S\) (line 3). A sliding window \(W\) of size up to \(k+1\) is moved along \(S\) (lines 4 and 5) and the segments in \(W\) are added to \(\Psi\) in reverse order (lines 6-12). For every size \(w \in [1, |W| - 1]\) suffix of \(W\), the next segment \(W[w+1]\) has its count \((nd.count(W[w+1]))\) updated (line 9). The prediction \(nd.pred\) associated to the node \(nd\) is also updated if needed (line 10).

Figure 4: Markov model learned from the training trajectories \(D_t = \{a_1, a_2, a_3\}\) in our running example \((k = 2)\).

The Markov model \(\Psi\) learned in the training phase is app-
Algorithm 1 Spatial-training

Require: Road network $G$, training data $D_t$, order $k$
Ensure: Markov chain model $Ψ$
1: $Ψ ←$ empty tree
2: for Trajectory $T ∈ D_t$ do
3: $S ← T$.spatial
4: for $s ∈ [1,|S|]$ do
5: Window $W ← S[\text{max}(1, s − k) : s]$
6: Node $nd ← Ψ$.root
7: for $w ∈ [|W| − 1 : 1]$ do
8: $nd ← nd.W[w]$
9: $nd$.count$[W[w + 1]] ← nd$.count$[W[w + 1]] + 1$
10: $nd$.pred$ ← \text{max}, nd$.count$[p]$
11: end for
12: end for
13: end for

Algorithm 2 Spatial-compression

Require: Trajectory $T$, model $Ψ$, order $k$
Ensure: Compressed spatial trajectory $S'$
1: $S ← T$.spatial
2: $S' ← S[1]$ >
3: for $s ∈ [2,|S|]$ do
4: Window $W ← S[\text{max}(1, s − k) : s − 1]$
5: $b ← |W|$
6: Node $nd ← Ψ$.root$[|W|]$
7: while $\text{null} ND b > 1$ do
8: $b ← b − 1$
9: $nd ← nd.W[b]$
10: end while
11: if $S'[s] \neq nd$.pred then
12: $S' ← S'[\text{max}(1, s − k) : s]$ >
13: end if
14: end for

applied for online compression of new trajectories, as described in Algorithm 2 (Spatial-compression). Besides $Ψ$, the algorithm receives a trajectory $T$ and the order $k$ as parameters and returns the spatial information of $T$ in compressed form $S'$. Similar to Algorithm 1, the compression algorithm extracts the spatial information $S$ from $T$ (line 1). The first segment $(S[1])$ is added to $S'$ (line 2), and, for each further segment $S'[s] (2 ≤ s ≤ |S|)$ is compared against the prediction from $Ψ$ using a sliding window $W$ of size up to $k$ past segments (lines 3-14). Predictions are based on the maximum matching suffix of $W$ in $Ψ$. In this case, the prediction is incorrect, the new update $S'[s]$ is added to $S'$ (line 12). We use the symbol ‘∪’ to denote spatial concatenation.

Figure 4 shows the Markov model $Ψ$ learned from the training trajectories $D_t = \{o_1, o_2, o_3\}$ in our example (Table 3), and for $k = 2$. Each node is represented as a pair $s_i : s_j$, where $s_i$ is a segment for matching and $s_j$ is a prediction. For instance, in the case the longest matching sequence of segments for a new update is $s_2.1$, the update is predicted as $s_2.3$. The compressed spatial information for the trajectory $o_4$ in our example is $< s_{1,2}, s_{2,1}, s_{3,4} >$, which gives a compression ratio of 2. Segments $s_2.3$ and $s_3.4$ were suppressed from the database, since they are predicted by the model.

Both the spatial-training and the spatial-compression algorithms are efficient in terms of space and time complexities. The size of the Markov model $Ψ$ is $O(\text{deg}^k)$, where $\text{deg}$ is the maximum in-degree of a node in $E$, and spatial-training takes time $O(|D_t||T|k)$, where $|T|$ is the maximum size of a trajectory. The time taken to compress a single trajectory using spatial-compression is $O(|T|k)$.

We define a new concept of spatial update block entropy ($h_k$) as the average number of database updates required to represent a new trajectory update $s_{i+k}$ given that the previous $k$ updates $S' = < s_{i+1}, s_{i+2}, \ldots, s_{i+k} >$ are known. As usual, we assume that trajectories are infinite and are generated from a stationary process (index $i$ is dropped).

$$h_k = \sum_{S' ∈ D} P(S)(1 − δ(s_{i+k}, Ψ(S', D)))$$
$$= \sum_{S' ∈ D} P(S) − \sum_{S' ∈ D} P(S' ∪ Ψ(S', D))$$
$$= 1 − \sum_{S' ∈ D} P(S' ∪ Ψ(S', D))$$

where $S' = < s_1 \ldots s_k >$, $S = S' ∪ < s_{i+k+1} >$, and $δ$ is the Kronecker delta function.

We apply the definition of $h_k$ to address an important question: How does the road network topology affect the spatial compression? We assume a random-walk trajectory model. In other words, given that the object is at a road segment $s_i$, it will choose any of the segments $s_j$ such that $(s_i, s_j) ∈ E$ uniformly at random. Similar to existing random-walk based algorithms [21], we add some connections to allow the trajectories to continue regardless of the segment they reach. Under this assumption, we can isolate the effect of the road network topology from the particular distribution of the trajectories in the data.

Theorem 1. Given a road network $G$ and a trajectory database $D$ composed of random trajectories:

$$h_k = 1 − \sum_{s_i ∈ V} \frac{π(s_i)}{\text{deg}^k(s_i)}$$

where $π(s_i)$ is the PageRank [21] of $s_i$, and $\text{deg}^k(s_i)$ is the out-degree of $s_i$ in $G$.

Theorem 1 enables the comparison of two road networks (e.g. different cities), quantifying the hardness of compressing trajectories on them. For instance, it follows directly from this theorem that compressing trajectories in a fully connected road network is hard because the entropy $h_k$ for random trajectories approaches 1. However, objects of relevance are not expected to move at random but rather follow mobility patterns [19]. Therefore, random trajectories should be more difficult to compress than actual ones.

3.2 Temporal Compression

The temporal compression algorithm applied by ONTRAC is also based on a prediction model learned using training data. However, instead of predicting road segments, here the goal is to predict a vehicle travel-times. Whenever a new update matches the one predicted by the model within an accepted error $λ$, the update is suppressed from the database.

We start by defining a generative Gaussian model for travel-times (Section 3.2.1) based on some intuitive assumptions regarding vehicle trajectories. In Section 3.2.2, we show how missing trajectory information can be inferred by fitting our Gaussian model via Maximum-likelihood Estimation, which leads to a convex Quadratic Programming formulation. This formulation is part of a learning algorithm (Section 3.2.3) that estimates travel-time distributions for road segments based on Expectation-Maximization. Finally, we describe how the learnt travel-time model is applied for trajectory compression in Section 3.2.4.
3.2.1 Generative Gaussian Model for Travel-times

In order to predict temporal trajectory updates, ONTRAC assumes a generative Gaussian model for travel-times. Similar to existing approaches, the distributions involved in the model are Gaussians \([15, 12]\). Such a model describes how the temporal information of GPS updates for a given trajectory is generated according to the expected travel-times for road segments and also taking into account the smoothness of vehicle speeds between segments. Figure 5 provides a high-level description of ONTRAC travel-time model, which will be described in the remainder of this section.

As discussed in Section 2, temporal trajectory information is based on GPS updates, which are sparse and noisy. We use \(t_i\) to represent total travel-times and \(t'_i\) for relative times (i.e. \(t'_i = t_i - t_{i-1}\)). Segments and times are indexed by pairs \((j, i)\), where \(j\) and \(i\) are associated to the GPS update and segment/time, respectively. Given a sequence \(< (s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k) >\), with missing temporal information, \(t_k = t_0 + \sum_{i=1}^{k} t'_i + \epsilon (\epsilon \sim N(0, \sigma^2_k))\), the GPS temporal error \(\sigma^*_k\), associated to \(t'_k\), can be estimated as:

\[
\sigma^*_k = \sigma^* (t_k - t_0) / \sum_{i=1}^{k} |s_i|
\]

where \(\sigma^*\) is the known GPS spatial error.

This error estimate follows from the speed formula \(v = d/t\). Notice that GPS updates provide only the total time taken to traverse a sequence of segments but the travel-time \(t'_i\) for each segment is missing. A simplistic approach is to assume that vehicles move at constant speed between updates (i.e. \(t'_j = [s_j|t_k - t_0| / \sum_{i=1}^{k} |s_i|\)). However, due to the sparsity of GPS data (see Table 1), such estimates are highly inaccurate. ONTRAC addresses this issue by learning travel-time distributions for segments from training data while enforcing a smoothness constraint on vehicle speeds.

ONTRAC assigns a travel-time distribution \(N(\phi_i, \omega_i^2)\) to each segment \(s_i \in E\). Vehicles traversing \(s_i\) will draw their travel-time \(t'_i\) from its distribution, as shown in Figure 5. Therefore, the total travel-time is such that:

\[
\sum_{i=1}^{k} t'_i \sim N(\sum_{i=1}^{k} \phi_i, \sum_{i=1}^{k} \omega_i^2)
\]

One important aspect of real trajectories is that physical constraints enforce speeds to be smooth (i.e. similar on nearby segments). We take smoothness into account in our model by adding a parameter \(\Delta\), which controls variations in vehicle speeds through adjacent road segments as follows:

\[
t'_i \bigg/ |s_i| \sim N\left( t'_{i+1} \bigg/ |s_{i+1}|, \Delta^2 \right)
\]

In the next section, we propose an algorithm for inferring travel-times given a sequence of GPS updates and model parameters. This inference step is applied by ONTRAC as means to predict temporal trajectories in the online compression. We use the notation \((\Phi, \Omega)\) to denote the union of the parameters \((\phi_i, \omega_i)\) for all the road segments in \(G\).

3.2.2 Temporal Trajectory Inference

Assuming that travel-times are generated according to our Gaussian model with known parameters (GPS error \(\sigma^*\), travel-time distributions \((\Phi, \Omega)\) and speed smoothness \(\Delta\), ONTRAC estimates missing travel-times \(t'_i\) for segments in a given trajectory from its GPS updates by means of Maximum-likelihood Estimation \([8]\). Here, we propose a Quadratic Programming (QP) formulation for maximizing the likelihood of a trajectory. This inference step can be performed efficiently because the resulting QP is convex \([8]\).

Without loss of generality, we will assume that every GPS update covers \(k\) road segments. Let \(T = < (s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k) >\) be a trajectory with \(n\) GPS updates and some missing temporal information. The sequence \(T' = < t'_1, \ldots, t'_k >\) contains the estimated travel-times for \(T\). Under the assumption of independence between the Gaussian trajectory model components, the likelihood of travel-times \(T'\), \(P(T')\), for temporal trajectory \(T\) given the parameters \(\sigma^*\), \((\Phi, \Omega)\), and \(\Delta\) is computed as:

\[
\prod_{i=1}^{n} P(t'_i|\phi_i, \omega_i) \prod_{i=1}^{n} P(t'_i|t'_{i-1}, \Delta) \prod_{j=1}^{n} P\left( \sum_{i=j}^{j+k} t'_i|\sigma^*\right)
\]

where \(t'_j = t_{j+k} - t_{j-1+k}\) is the travel-time in between updates \((s_{j-1+k}, t_{j-1+k})\) and \((s_{j+k}, t_{j+k})\).

Our goal is to identify the travel-times \(T'_{\text{max}}\) that maximize \(P(T')\). However, devising an analytical formulation for \(T'_{\text{max}}\) has shown to be a quite elusive task. Instead, ONTRAC makes use of the assumption that the several distributions involved in our model are Gaussians, which enables an expansion of Expression 9 as a Quadratic Programming (QP) instance, as shown in Theorem 2:

**Theorem 2.** The following Quadratic Programming formulation is the maximum-likelihood estimate for travel-times \(T'\) for a trajectory \(T\) assuming the trajectory Gaussian model.

\[
T'_{\text{max}} = \arg \min_{\mathbf{x}} \frac{1}{2} \mathbf{x}^T \mathbf{Q} \mathbf{x} + c^T \mathbf{x}
\]

s.t. \(\mathbf{x} \geq 0\), where:

\[
\mathbf{Q} = Q_1 + Q_2 + Q_3, c = c_1 + c_2
\]

\[
Q_1 = \begin{pmatrix}
\frac{1}{\sigma^2_1} & 0 & \ldots & 0 \\
0 & \frac{1}{\sigma^2_2} & \ldots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \ldots & \frac{1}{\sigma^2_{n,k}}
\end{pmatrix}
\]

\[
\mathbf{Q} = \begin{pmatrix}
p_1 & p_2 & \ldots & p_{n-k} \\
p_2 & p_3 & \ldots & p_{n-k-1} \\
\vdots & \vdots & \ddots & \vdots \\
p_{n-k} & \ldots & \ldots & p_{n-k}
\end{pmatrix}
\]

\[
c = \begin{pmatrix}
c_1 \\
c_2 \\
\vdots \\
c_{n-k}
\end{pmatrix}
\]
Theorem 2 supports the use of the existing theory on Quadratic Optimization to compute maximum-likelihood estimates for vehicle travel-times based on Equation 3. However, general QP is NP-hard and an algorithm that is exponential on the number of road segments would prevent the application of our compression scheme to large trajectories. We show that our QP formulation is convex, and thus can be solved in polynomial time.

**Theorem 3.** The Quadratic Programming problem described in Theorem 2 is convex.

We show how Theorems 2 and 3 are applied by ONTRAC using our running example (see Table 3). Consider trajectory $\mathcal{G}_3$ and assume $\phi_{2,3} = 6, \phi_{1,4} = 7, \phi_{3,2} = 1, \omega_{2,3} = 2, \omega_{3,1} = 2, \Delta = 2.5$ and $\sigma_{1,5} = 3$. Then the matrix $Q$ and vector $c$ are computed as follows:

$$Q = \begin{pmatrix}
1 & 0 & 0 & \cdots & 0 \\
0 & .01 & .005 & 0 & \cdots \\
0 & .05 & .005 & 0 & \cdots \\
\cdots & \cdots & \cdots & \cdots & \cdots \\
0 & 0 & 0 & \cdots & 1
\end{pmatrix},
$$

$$c = \begin{pmatrix}
\phi_{1,1} \\
\phi_{2,1} \\
\phi_{3,1} \\
\cdots \\
\phi_{n,k}
\end{pmatrix},$$

for each trajectory are latent variables. The algorithm starts with an initial setting for the model $(\Phi^0, \Omega^0)$. In the expectation step (e-step), expected values for the latent variables given the parameters are computed using Quadratic Programming (line 5). Next, in the maximization (m-step), the model $(\Phi^t, \Omega^t)$ is updated to maximize the likelihood of the data given the latent variables computed in the e-step (lines 7-12). After a finite number of iterations, the algorithm converges to a local optima, as shown in Theorem 4.

**Theorem 4.** The Temporal-training estimate for the model $(\Phi, \Omega)$ converges to a local optima.

Temporal-training runs in $O(q(|D|t^3 + |V|^3))$ time, where $q$ is the number of iterations, $|T|$ is the maximum trajectory length and $V$ is the set of segments. Solving a QP instance takes $O(|T|^3)$ time and the algorithm solves $n$ QP instances in the e-step, updating the travel-time distribution parameters for each segment $s_i \in V$ in the m-step.

**3.2.4 Compression**

ONTRAC applies the travel-time model $(\Phi, \Omega)$ learned by the Temporal-training algorithm to predict travel-times. Updates are suppressed if the model predictions are within a user-defined accepted time error $\lambda$ from the actual travel-times. Therefore, ONTRAC’s temporal compression is lossy.

Update travel-times are computed by fusing the model $(\Phi, \Omega)$ and the GPS update information. Given our Gaussian assumptions, the problem consists of fusing the means of two Gaussians with known variance as a weighted mean:

$$\frac{\hat{t}\sigma_i^2 + \bar{t}_i\hat{w}}{\hat{w} + \sigma_i^2}$$

where $\hat{t}$ is the model prediction with variance $\sigma_i^2$ and $\bar{t}_i$ is the GPS time update with variance $\hat{w}^2$.

For missing updates, ONTRAC assumes that travel-times $t_j$ are proportional to the estimates from the model $(\Phi, \Omega)$:

$$\frac{(\hat{t}\sigma_i^2 + \bar{t}_i\hat{w})\phi_j}{(\hat{w} + \sigma_i^2)t_j}$$

As a consequence, by keeping the model and the compressed version of the non-missing updates, ONTRAC guarantees the recovery of the missing ones within error $\lambda$.

Algorithm 4 describes ONTRAC’s temporal compression algorithm. It receives a trajectory $T$, the prediction model $(\Phi, \Omega)$, the GPS error $\sigma^*$, and the time error $\lambda$ as parameters.
Algorithm 4 Temporal-compression

Require: Trajectory $T$, travel-time model $(\Psi, \Phi)$, GPS error $\sigma^*$, accepted error $\lambda$
Ensure: Compressed temporal trajectory $D$

1: $D \leftarrow \emptyset$; $d \leftarrow 0$
2: $\tau \leftarrow t_0$; $\hat{\tau} \leftarrow 0$; $\hat{\omega} \leftarrow 0$; $\hat{\sigma} \leftarrow 0$
3: for updates $i \in [1, |T|]$ do
4: \quad $t \leftarrow t + \phi_i$; $\hat{\omega} \leftarrow \hat{\omega} + \omega_i^2$; $d \leftarrow d + |s_i|$\n5: \quad if segment $t_i$ is non-missing then
6: \quad \quad $t^* \leftarrow t^* + \frac{\hat{\tau}^2 - \tau^2}{4 \hat{\sigma}}$
7: \quad \quad $\tau \leftarrow \tau + t$
8: \quad \quad if error $|\tau - t^*| > \lambda$ then
9: \quad \quad \quad Add update $D \leftarrow D \cup \{d, t^*\}$; $\tau \leftarrow t$
10: \quad end if
11: \quad $\hat{\omega} \leftarrow 0$; $\hat{t} \leftarrow 0$
12: end if
13: end for

Figure 6: Example of temporal compression for trajectory $o_4$ and $\lambda = 5$.

and returns a compressed version $D$ of $T$, which is initialized as empty (line 2). Also, both the actual total travel-time $t^*$ and the model estimate $\tau$ are initialized with the starting time of the trajectory (lines 2-3). For a non-missing update in $T$ the model travel-time prediction $t$ is fused with the GPS time $t_0$ in order to compute the actual travel-time $t^*$ (line 6). If the predicted total time $\tau$ is within $\lambda$-error from the actual total time $t^*$, the update is suppressed from $D$. Otherwise, the update is added $D$ and the predicted total travel-time $\tau$ is corrected to the actual one (lines 9-10).

Figure 6 illustrates ONTRAC’s temporal compression for trajectory $o_4$, time error $\lambda = 5$ and assuming time $\phi_{1, j} = 10$ for any segment $s_j$ in our running example. Two updates, $(s_{2, 1, 15})$ and $(s_{2, 3, ?})$, are suppressed based on the prediction model model. We assume the GPS update and fused times are the same in our example for the sake of simplicity.

The time complexity of Temporal-compression for given a trajectory $T$ is linear with the size of the trajectory ($O(|T|)$). Moreover, the algorithm guarantees the upper bound on the time error $\lambda$ for any update in the compressed database.

3.3 Querying

We describe how ONTRAC supports a type of query that returns the location of an object at a particular time.

Definition 3. where($o_i, t$) query: Given a database $D$, an object $o_i$, and a time $t$, return the road segment $s_j$ where $o_i$ is located at time $t$ (i.e. $s_j, t \in c(t_1)$).

For instance, where($o_{1, 4}, 10$) = $s_{1, 2}$. Notice that, different from the full data, ONTRAC’s compressed trajectories cannot be queried without information from the compression model. In particular, because spatial information is compressed based on prefix matching (see Equation [1]), recovering a trajectory requires contextual information that might also have been compressed. A simplistic approach to address this problem is performing a full reconstruction of a compressed trajectory $T$ at query time, which might incur a significant overhead on query performance for long trajectories. Instead, ONTRAC supports the recovery of a partial decompression $\hat{T}$, which is a sub-trajectory of the decompressed version of $T$ that covers a query time $t$.

The idea behind partial trajectory decompression is building a query context that goes far enough backwards in time to make the reconstruction unique. We use the notation $T[< s_{p} \rightarrow s_{q}]$ to represent all the possible sub-trajectories in the form $< s_{p}, s_{p+1} \ldots s_{q} >$ where $s_{q} = \Psi(< s_{p}, s_{p+1} \ldots s_{q-1} >)$. Moreover, $\tilde{T}[\Psi(< s_{p} \rightarrow s_{q} >)]$ gives the corresponding set of next segment predictions for sub-trajectories $T[< s_{p} \rightarrow s_{q}]$ in $\Psi$. Once there is a next segment prediction, the trajectory can be uniquely reconstructed.

Algorithm 5 shows the partial trajectory decompression algorithm. It receives a compressed trajectory $T$, the spatial ($\Psi$) and temporal ($\Phi$) compression models, and the query time $t$. As a result, it returns $\hat{T}$ which is a partial decompression of $T$. The algorithm starts building a context $< s_{p}, s_{q} >$ that precedes the time $t$ and is long enough to cover a prefix of order $k$ (line 1). The segment $s_{p}$ is moved backwards in $T$ until it leads to a unique segment prediction (lines 2-4). The trajectory $t$ is then decompressed using the spatial and temporal model $(\Psi, \Phi)$ starting from $s_{p}$ (lines 5-8). We use the symbol ‘<’ to denote ‘the lastest before’.

For instance, consider the compressed trajectory $< (s_{1, 1, 5}), (s_{1, 2, 8}), (s_{2, 1, 2, 5}), (s_{2, 3, 30}) >$, a time $t = 31$ for which a query requires a partial reconstruction, and the spatial prediction model $\Psi$ from Figure 3 with order $k = 2$. The algorithm will build a context $< s_{1, 2} \rightarrow s_{2, 3} >$ and compute the possible partial reconstructions $\Psi(T[s_{1, 2} \rightarrow s_{2, 3}]) = \{s_{3, 4}\}$. Since this reconstruction is unique, the trajectory can be partially reconstructed using $< s_{1, 2} \rightarrow s_{2, 1} >$. Otherwise, it would move backwards along the compressed trajectory, adding $s_{0, 1}$ to the context.

The Partial-decompression algorithm has time complexity $O(|T|\deg(k))$, where $|T|$ is the maximum size of a decompressed trajectory, $\deg$ is the maximum degree of a segment in $E$, and $k$ is the order of the spatial compression model $\Psi$. This is the same complexity of the querying procedure using the partial decompression scheme.

4. EXPERIMENTAL RESULTS

We evaluate ONTRAC online trajectory compression using real-world and synthetic datasets. Sections [1] and [2] show that our approach achieves high compression ratios, outperforming the baseline, in terms of spatial and temporal compression, respectively. Section [3] demonstrates how ONTRAC improves database scalability in trajectory data.
management with a low overhead over the query time.

Two datasets combining GPS taxi traces and road network data from OpenStreetMap are applied in the experimental evaluation of ONTRAC. The first one is the San Francisco Cab dataset (SF) [22], which consists of GPS traces from 500 cabs over 30 days in San Francisco. The second dataset (Beijing) [27] contains GPS traces generated by 10,357 cabs during a week-long period in Beijing. Table 1 shows the number of vertices and edges, the number of GPS updates, and the sampling rate (average time between two consecutive GPS updates) for the datasets.

Baseline: We extend PRESS [23] to the online compression setting as a baseline for ONTRAC. A maximum delay is added as a parameter for both spatial and temporal compression. Updates can be compressed as in batch mode within the time delay and once the delay is reached, the trajectory database is updated. For temporal compression, we considered the maximum delay to be equal to λ. Moreover, we replace PRESS’s Huffman codes generated for spatial trajectories by integer identifiers in order to add database support for PRESS compressed data.

Implementation: The algorithms are implemented in C++ and are publicly available. Map-matching is performed using our implementation of a state-of-the-art algorithm [20]. Querying and storage are implemented on top of PostgresQL and its geographic extension PostGIS. Quadratic programming is implemented using CPLEX. The training step of our temporal compression algorithm (Algorithm 3) was parallelized using pthreads.

Experimental Setting: Experiments were run on a 2.67 GHz 8-core Intel i7 with 196GB RAM. We set the GPS error σ to 5 meters by default and the smoothness parameter Δ was estimated from the speed variance in the GPS updates.

Additional Experiments: We present additional experimental results in the Appendix, including those for synthetic data and also entropy evaluation for road networks and spatial trajectory information.

4.1 Spatial Compression

ONTRAC’s spatial compression is evaluated in terms of compression ratio (in number of database updates), compression time and training time in Figure 7. We compare our approach against PRESS when different delays (in minutes) are allowed. For each dataset, 90% of the trajectories are used for training and 10% are used for compression.

In Figures 7a and 7b, we show the compression results of ONTRAC and PRESS with varying delays. ONTRAC achieves compression ratios of 13.6 and 10.4 for SF and Beijing, respectively. Even when delays are up to 4 minutes, ONTRAC still achieves better compression results than the baseline.

In Figures 7c and 7d, we evaluate our approach in terms of training and compression time. Results show that ONTRAC achieves similar or better performance than the baseline approach for most of the datasets, being able to learn its compression model using up to 10 million updates in a time in the order of minutes. Moreover, ONTRAC can compress up to a million trajectories in time in the order of seconds.

4.2 Temporal Compression

In this section, we evaluate ONTRAC and the baseline technique in terms of temporal compression. Our main goal is to assess how many updates to the database can be suppressed by applying ONTRAC’s prediction scheme. As in the last section, we divide the datasets into training and test folds with 90% and 10% of the trajectories, respectively. Moreover, we vary the accepted error λ from .5 to 8 minutes as means to analyze the trade-off between compression accuracy and compression ratio in the lossy compression.

The relationship between the compression error (λ) and the compression ratio for ONTRAC and PRESS is shown in Figures 8a and 8b. ONTRAC outperforms PRESS for all the datasets and different values of λ. For instance, ONTRAC achieves from 16 (λ = 0.5) to 21 (λ = 8) times higher compression for SF dataset and from 8 (λ = 8) to 14 (λ = 0.5) higher compression for Beijing.

In Figures 8c and 8d, we evaluate ONTRAC and PRESS compression with regard to training and compression time. PRESS performs line simplification, which requires no training. On the other hand, ONTRAC’s Temporal-training is an EM algorithm and thus performs several iterations to compute its prediction model. Results show that our training algorithm requires approximately 3 hours for training using 12 million trajectories and 8 cores. Because ONTRAC’s

<table>
<thead>
<tr>
<th>name</th>
<th>#vertices</th>
<th>#edges</th>
<th>#updates</th>
<th>samp. rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>SF</td>
<td>78,847</td>
<td>181,598</td>
<td>7.7M</td>
<td>2.27m</td>
</tr>
<tr>
<td>Beijing</td>
<td>461,710</td>
<td>904,144</td>
<td>12.6M</td>
<td>6.86m</td>
</tr>
</tbody>
</table>

Table 1: Dataset statistics.
ONTRAC achieves up to 3 times higher compression ratio than PRESS. Our approach is efficient in terms of compression time, significantly outperforming the baseline. Efficient compression is of particular interest in the online trajectory compression setting.

Figures 8 and 9 show the database scalability using ONTRAC (\( \lambda = 1 \text{ min.} \)) and without any compression (FULL). ONTRAC increases the number of inserts/second by factors of 11.0 and 11.7 for SF and Beijing datasets, respectively. We also evaluate the impact of ONTRAC over querying performance, for complete (ONTRAC-DC) and partial (ONTRAC-PT) trajectory decompression, in Figures 8(c,d) and 9(c,d). ONTRAC-DC achieves poor query performance due to the cost of full decompression. Such an issue was the main motivation for the partial decompression approach (ONTRAC-PT), which incurs limited overheads of 38% and 25% compared to the FULL setting for SF and Beijing, respectively.

5. RELATED WORK

Our work is focused on map-matched trajectory compression, a problem introduced in [3]. The proposed approach, called Nonmaterialized Trajectory Construction (NTC), represents trajectories in a linear reference system (distance, time) and finds a minimum sequence of line segments that covers the original trajectory within a user-defined error. Map-matched trajectory compression (MMTC) [16] compresses a trajectory by replacing some of its parts by shorter sub-trajectories (i.e. those with fewer road segments).

In a recent work [23], the authors proposed decomposing trajectories into spatial and temporal components for which different compression schemes were presented. The resulting approach, called PRESS, applies shortest paths and Huffman coding to compress the spatial information and function simplification (similar to [5]) for the temporal information. Experiments show that it outperforms both NTC and MMTC in terms of compression ratio and query performance. However, PRESS relies on a batch model, an aspect that limits its application in online settings, such as taxi dispatching [18] and traffic monitoring [11], where large volumes of trajectories have to be processed in real-time in order to support high-quality services. Moreover, PRESS compressed data cannot be directly processed by existing relational databases, which offer many capabilities (e.g. indexing) that are critical for trajectory data management. These limitations are the motivation for our work.

Similar to PRESS, ONTRAC also employs different compression schemes for spatial and temporal information. More specifically, its spatial compression is inspired by Prediction by Partial Matching (PPM) [6], a state-of-art statistical compression scheme. PPM enables the creation of compact
bit string representations for symbols that are more likely to occur given a sequence of previous symbols. In ONTRAC, we apply a Markov model to suppress the most likely update for an object from the database based on the road segments it has traversed recently. Similar models have been recently applied for the inference of uncertain trajectories [1] and for answering predictive queries on road networks [13].

ONTRAC’s temporal compression exploits Quadratic Programming (QP) [3] and Expectation Maximization (EM) [7] in order to learn a travel-time model to predict future updates. While there is an extensive literature on travel-time inference [14][15][17][26][24][27], the online compression setting has important properties that motivate a new approach. For instance, the inference has to be made at update time, which imposes higher performance constraints. Moreover, online compression requires shorter term predictions since travel-times will be in the order of minutes.

6. CONCLUSION

We have described ONTRAC, a new framework for online map-matched trajectory compression. ONTRAC applies prediction models for spatial and temporal trajectory information as means to suppress updates to a trajectory database. In particular, ONTRAC’s spatial compression predicts road segments via a Markov model and its temporal compression scheme learns a travel-time model by combining Quadratic Programming and Expectation Maximization. Experiments have shown that our approach achieves high compression ratios in terms of spatial and temporal compression for real datasets. ONTRAC outperforms the baseline method even when 4 minute delays in updates are allowed and achieves up to 21 times higher compression for travel-times. Finally, we have shown that ONTRAC can increase database scalability by up to one order of magnitude with small overhead over querying performance.

This paper opens promising directions for future research. Adding support to other, more complex queries, such as nearest-neighbors and range queries, using partial reconstruction would increase the applicability of our framework. Also, a context-based travel-time prediction, similar to our spatial compression scheme, might improve ONTRAC’s compression power. Finally, online EM [19] can support efficient online updates of ONTRAC’s temporal compression model.

7. REFERENCES


APPENDIX

A. PROOFS OF THEOREMS
Theorem 1
Proof. Given the random-walk trajectory assumption:

\[ h_k = 1 - \sum_{S' \in \mathcal{D}} P(S' \cup \Psi(S', \mathcal{D})) \]
\[ = 1 - \sum_{S' \in \mathcal{D}} P(s_k \cup \Psi(S', \mathcal{D})) \]
\[ = h_1 \]

We know that \( P(s_i) \) is the PageRank \( \pi(s_i) \) of \( s_i \) in \( G \). Finally, again using the random-walk assumption:

\[ P(s_i \cup s_j) = \pi(s_i) \frac{1}{d_{e_{ij}}(s_i)} \forall (s_i, s_j) \in E \]

\[ \square \]

Our goal is to identify the travel-times \( T_{\text{max}}' \) that maximize \( P(T') \). However, devising an analytical formulation for \( T_{\text{max}}' \) has shown to be a quite elusive task. Instead, ON-TRAC makes use of the assumption that the several distributions involved in our model are Gaussians, which enables an expansion of Expression 3 as a Quadratic Programming (QP) instance, as shown in Theorem 2.

Theorem 2
Proof. From maximum-likelihood theory, we know that:

\[ T_{\text{max}}' = \max_{T'} P(T') = \min_{\mathcal{T}_q} \{- \log P(T')\} \]

Also, from the Gaussian distribution definition:

\[ N(x; \mu, \sigma) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \]

\[ \log N(x; \mu, \sigma) = -\log \sigma - \frac{(x-\mu)^2}{2\sigma^2} - \frac{1}{2} \log 2\pi \]

After removing some constants, we can write the logarithms of the probabilities in Equation 3 as follows:

\[ \log P(t' | \phi_i, \omega_i) = -\frac{(t'_i - \phi_i)^2}{2\omega_i^2} \]

(6)

\[ \log P(t' | t'_{i-1}, \Delta) = -\frac{1}{2\Delta^2} \left| t'_i - t'_{i-1} \right|^2 \]

(7)

And \( \log P(T') \) can be computed as:

\[ \sum_{(n,k)} \log P(t'_i | \phi_i, \omega_i) + \sum_{(n,k)} \log P(t'_i | t'_{i-1}) + \sum_{j=1}^n \log P(\sum_{i=j(k)} t'_i) \]

where some parameters are omitted for conciseness. Expanding Equation 6 and dropping some constants:

\[ \sum_{(n,k)} \log P(t'_i | \phi_i, \omega_i) = -\sum_{(n,k)} \frac{t'^2_i}{2\omega_i^2} + \sum_{i=0}^n \frac{t' \phi_i}{\omega_i^2} \]

We represent times in vector form \( x \) (\( x[i] = t'_i \)) and take the \( \frac{1}{2} \) factor in the QP notation into account, which leads to \( Q_1 \) and \( c_1 \). Doing the same for Equations 7 and 8.

\[ \sum_{j=1}^n \log P(t'_i | t'_{i-1}, \Delta) = -\frac{t'^2_i}{2\Delta^2 |t'_{i-1}|^2} - \frac{t'^2_{n,k}}{2\Delta^2 |t'_{n,k}|^2} \]

Theorem 3
Proof. A sufficient condition for the convexity of the formulation is \( Q \) to be positive-definite (i.e. \( xQx^T > 0 \) for any non-zero vector \( x \)). From the definition of \( Q \):

\[ xQx^T = (xQ_1 + Q_2 + Q_3)x^T \]

\[ = xQ_1x^T + xQ_2x^T + xQ_3x^T \]

\( Q_1 \) is positive-definite because it is a diagonal matrix with positive entries, its eigenvalues are the diagonal. \( Q_2 \) is positive-definite due to the fact that it is a block diagonal matrix in which each \( 2 \times 2 \) block is itself a positive-definite matrix. \( Q_3 \) is also a block diagonal matrix where each block is constant and thus positive-semidefinite, the same holding for \( Q_3 \) (i.e. \( xQ_3x^T > 0 \), for any non-zero \( x \)).

Theorem 4
Proof. We will prove the convergence of the algorithm based on how new estimate (\( \Phi^T, \Omega^T \)) is related to the one computed in the previous iteration, (\( \Phi^{t-1}, \Omega^{t-1} \)):

\[ (\Phi^T, \Omega^T) = \arg \max_{\Phi, \Omega} Q(\Phi, \Omega, \Phi^{t-1}, \Omega^{t-1}) \]

(9)

where (replacing \( t'_i \) by \( t_i \) for conciseness):

\[ Q = \sum_{T \in \mathcal{T}_i, n=1}^{||T||} \int_{t_i}^1 P(t_i | \Phi^{t-1}, \Omega^{t-1}) \log P(t_i | \Phi, \Omega) dt_i \]

We maximize \( Q \) by computing \( \Phi \) and \( \Omega \) such that \( \frac{dQ}{d\Phi} = 0 \) and \( \frac{dQ}{d\Omega} = 0 \) (dropping some constants):

\[ \frac{dQ}{d\Phi} = -\sum_{T \in \mathcal{T}_i} \int_{t_i}^1 P(t_i | \Phi, \Omega^T) \frac{d}{d\Phi} (t_i - \phi_i)^2 dt_i \]

\[ = -\sum_{T \in \mathcal{T}_i} \int_{t_i}^1 (t_i - \phi_i)^2 dt_i \]

And

\[ \frac{dQ}{d\Omega} = \sum_{T \in \mathcal{T}_i} \int_{t_i}^1 P(t_i | \Phi, \Omega^T) \frac{d}{d\Omega} \omega_i dt_i \]

By solving the derivative and eliminating some multiplicative terms, we get an equivalent expression:

\[ \frac{dQ}{d\Omega} = \sum_{T \in \mathcal{T}_i} (t_i - \phi_i)^2 - T \omega^2 \]

\[ \frac{dQ'}{dT} = \sum_{j=1}^{||T||} (t_i - \phi_i)^2 - T \omega^2 \]
Table 2: Synthetic dataset statistics.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Lines</th>
<th>Polygons</th>
<th>Updates</th>
<th>Compression Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syn-I</td>
<td>78,847</td>
<td>181,598</td>
<td>3.3M</td>
<td>2m</td>
</tr>
<tr>
<td>Syn-II</td>
<td>78,847</td>
<td>181,598</td>
<td>0.7M</td>
<td>10m</td>
</tr>
</tbody>
</table>

This proves that lines 10 and 11 of Temporal-training cannot decrease the likelihood of the trajectories and thus the algorithm is guaranteed to converge.

### B. ADDITIONAL EXPERIMENTS FOR REAL AND SYNTHETIC DATA

#### B.1 Synthetic Data

Experiments were also performed using two synthetic datasets (Syn-I and Syn-II) generated using the SF road network and synthetic trajectories. Each trajectory was created by selecting a random starting and destination segments. Road segments in a trajectory are those in the shortest path between the starting point and destination. Travel-times follow segment distributions as defined in our trajectory model (see Section 3.2.1) with Gaussian distributed speeds from \( N(15, 10) \) (in m/s). Destinations are selected according to an exponential popularity distribution \( \alpha e^{-\alpha x} \), with \( \alpha = 1 \) in Syn-I and \( \alpha = 10^{-4} \) in Syn-II. Therefore, while Syn-I has a few very popular destinations, Syn-II has many almost equally popular ones.

#### B.2 Network and Trajectory Entropies

We characterize the SF and Beijing datasets in terms of spatial update block entropy, as defined in Section 3.1. Figure 10a shows the entropy of the networks (see Theorem 1) for random-walk trajectories and compares them against a line and a complete graph. Both road networks are sparse and the results show that more than half of the trajectory updates can be saved in the extreme case of random trajectories. SF has an entropy 17% larger than Beijing.

Figure 10b compares the trajectories from datasets in terms of entropy for varying order \( k \). As expected, the entropy decreases as \( k \) grows, but differences between entropies at order 1 and order 6 are small (within 0.03). Trajectories in the Beijing dataset present larger entropy than those in SF. Also, by design, trajectories in Syn-II have higher entropy than those in Syn-I. Based on these results, we fix ONTRAC’s spatial compression order to 2 in the remaining of the experiments discussed in this section.

#### B.3 Spatial Compression

For Syn-I, where most of the trajectories share the same destination, our approach achieves a compression ratio of 54. However, in Syn-II, where destination popularity is almost uniformly distributed, its compression ratio decreases to 16.

#### B.4 Temporal Compression

Figures 12a-12d provide experimental evidence of the convergence of our approach (see Theorem 3 for a formal proof). The number of iterations \( q \) is set to 5 in all experiments discussed in this section, which has shown to produce a good compromise between training time and compression ratio.

We notice different trends in the compression ratio for the real and synthetic datasets, which are due to the shorter length of the synthetic trajectories (i.e. shortest paths between a starting and end point) compared to the real ones. In these cases, ONTRAC quickly approaches to its maximum compression ratio.

#### B.5 Scalability and Querying

Regarding the synthetic datasets, ONTRAC achieves 50 and 13 more inserts/second than the baseline, respectively. These numbers are consistent with the compression ratios presented in Sections 4.1 and 4.2. For synthetic datasets, trajectories are typically short, and thus there is no overhead in decompressing the complete trajectories.
Figure 11: Spatial compression ratio (a,b), training and compression times (c,d) for ONTRAC and PRESS using Syn-I (a,c), and Syn-II (b,d) datasets.

Figure 12: Temporal compression convergence using the SF (a), Beijing (b), Syn-I (c), and Syn-II (d) datasets.

Figure 13: Temporal compression ratio (a,b), and training and compression times (c,d) for ONTRAC and PRESS using the Syn-I (a,c), and Syn-II (b,d) datasets.

Figure 14: Inserts (a,b) and queries/second (c,d) when applying ONTRAC with complete (ONTRAC-DC) and partial decompression (ONTRAC-PT) and with no compression (FULL) using Syn-I (a,c) and Syn-II (b,d) datasets.