PPQ-Trajectory: Spatio-temporal Quantization for Querying in Large Trajectory Repositories

Shuang Wang
University of Warwick
Coventry, United Kingdom
Shuang.Wang.1@warwick.ac.uk

Hakan Ferhatosmanoglu
University of Warwick
Coventry, United Kingdom
Hakan.F@warwick.ac.uk

ABSTRACT

We present PPQ-trajectory, a spatio-temporal quantization based solution for querying large dynamic trajectory data. PPQ-trajectory includes a partition-wise predictive quantizer (PPQ) that generates an error-bounded codebook with autocorrelation and spatial proximity-based partitions. The codebook is indexed to run approximate and exact spatio-temporal queries over compressed trajectories. PPQ-trajectory includes a coordinate quadtree coding for the codebook with support for exact queries. An incremental temporal partition-based index is utilised to avoid full reconstruction of trajectories during queries. An extensive set of experimental results for spatio-temporal queries on real trajectory datasets is presented. PPQ-trajectory shows significant improvements over the alternatives with respect to several performance measures, including the accuracy of results when the summary is used directly to provide approximate query results, the spatial deviation with which spatio-temporal path queries can be answered when the summary is used as an index, and the time taken to construct the summary. Superior results on the quality of the summary and the compression ratio are also demonstrated.

PVLDB Reference Format:
Shuang Wang and Hakan Ferhatosmanoglu. PPQ-Trajectory: Spatio-temporal Quantization for Querying in Large Trajectory Repositories. PVLDB, 14(2): 215-227, 2021.
doi:10.14778/3425879.3425891

1 INTRODUCTION

With the prevalence of positioning devices and mobile services, massive amounts of location sequences are being generated continuously. Maintaining and querying small-sized representations of raw trajectory data are needed for a wide variety of applications, such as real-time traffic management [32] and intelligent transport systems [4].

Existing trajectory compression methods do not address this need for a number of reasons. First, many of them are defined for edge sequences in a road network [14, 18, 21, 38]. They require pre-processing steps of mapping raw GPS data to the road network structure, followed by transforming the map-matched location data to edge-based sequences. The mapping and transformation processes reduce accuracy and result in limited support for detailed queries. Second, most solutions perform offline compression over full trajectory data, with execution times usually undesirable for online applications. There is a need for scalable online compression. Third, the existing compressed representations can not be directly used to answer spatio-temporal queries without a costly decompression process.

To address these challenges, we present PPQ-trajectory, a spatio-temporal quantization-based solution to generate a compact representation and support a wide range of queries over large trajectory data. An overview of PPQ-trajectory is presented in Figure 1.

The first part of PPQ-trajectory is the partition-wise predictive quantizer (PPQ) that generates an error-bounded summary, consisting of the codebook and prediction coefficients for spatial and autocorrelation-based partitions. The second part is the coordinate quadtree coding (CQC) for the error space caused by the quantization, which enables an accurate reconstruction of the trajectories. These two parts form the summary for the trajectory data, as illustrated in Figure 1. The third part is the temporal partition-based index (TPI) for organizing the quantized spatio-temporal data. Given a query, TPI is used to prune the data space and generate a candidate list of trajectories, whose reconstructed points can be computed from the summary. Overall, PPQ-trajectory generates and uses an indexed summary over raw data sequences to support efficient analysis, ranging from simple queries, such as vehicles passing by a location (x, y) at a given time t, to more complex analytic tasks, such as predicting future positions of entities.

We evaluate PPQ-trajectory with respect to a number of performance measures: the quality of approximate query results, efficiency of exact queries, index building times, and compression ratio. We implemented several baselines, including the widely-used product quantization [19], residual quantization [8], REST [44], which is a recent reference based trajectory compression method, and TrajStore [10], an adaptive storage solution for trajectories.

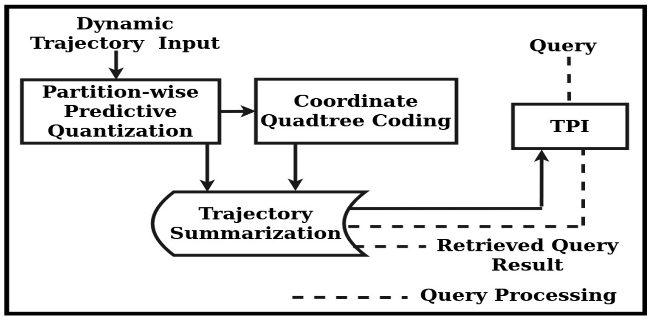


Figure 1: An Overview of PPQ-trajectory

Proceedings of the VLDB Endowment, Vol. 14, No. 2 ISSN 2150-8097.
doi:10.14778/3425879.3425891

This paper makes the following contributions: (1) A spatio-temporal predictive quantizer, PPQ, is designed with an error-bounded codebook for each of the partitions, which are incrementally generated based on spatial and autocorrelation similarity. (2) Utilizing a quadtree structure and a padding process, CQC is developed to encode the relative positions of trajectory points with reconstructed ones for an accurate trajectory reconstruction. A local search strategy is presented to identify exact query results. (3) The temporal partition-based index dynamically reuses parts of the past index to support efficient spatio-temporal queries. (4) PPQ-trajectory answers spatio-temporal queries over raw data, without a full reconstruction of trajectories or accessing all of the candidate trajectories. (5) Experimental results demonstrate significant improvements achieved by PPQ-trajectory. For example, for approximate spatio-temporal queries, it is 3% – 52% more accurate, compared to product quantization, residual quantization, and TrajStore. The mean absolute error (MAE) of PPQ-trajectory is a few or tens of meters, while the alternative approaches' MAE values are orders of magnitude larger for the same size codebook. Significant improvements are also observed for the efficiency, index building times, and compression ratios.

The rest of the paper is organized as follows. The related work is in Section 2. Section 3 presents PPQ. In Section 4, we present CQC and the associated local search strategy. In Section 5, the temporal organization for the quantized data is presented. The effectiveness of PPQ-trajectory is verified via an extensive set of experiments in Section 6. Conclusion is provided in Section 7.

2 RELATED WORK

With the widespread adoption of location-based services, compressing trajectory data has become a prevalent area with high practical relevance [2, 21, 26]. While traditional compression methods aim to reduce the reconstruction error and improve compression ratio, the data management challenge is to design the compression method with the objective of answering queries efficiently, and supporting online querying directly over compressed data.

Road network-constrained trajectory compression has gained significant attention [14, 18, 21, 25, 36, 38]. The common approach is to map raw trajectories to road networks and compress the map-matched trajectories [20]. There is also significant attention on raw trajectory compression. SQUISH and SQUISH-E use a priority queue to remove redundant points [30, 31]. A bounded quadrant system (BQS) is developed in [25], which uses convex hull bounding to achieve trajectory compression. Based on BQS, [26] achieves streaming trajectory compression, and aging history data without overwriting. Another recent method transforms trajectories into vehicle state vector functions and generates an inverted index based on the road segments [4]. This approach requires road segment information and matching each road id with corresponding objects.

The solutions that we included in our experiments are TrajStore [10] and REST [44]. TrajStore aims compression via an adaptive spatial index and clustering the sub-trajectories. It recursively updates the index by merging, splitting or appending. REST is a recent compression-based method which compares trajectories with the sub-trajectories of a reference set. Generating a representative set is challenging especially under changing conditions, where it can fail to represent data from regions that lack enough samples.

Early work in this area applies traditional index structures for trajectory data. For example, STRIPES uses quadtrees to index the predicted positions of moving objects [35]. In [3], an index for trajectories is developed by indexing the coefficients of Chebyshev polynomials that represent trajectories. Most those methods focus on similarity queries and do not address efficient spatio-temporal database queries. Zheng et al., [45] index the reference-based trajectories with IR-tree [24], which is based on R-tree referring to the inverted files for sub-trajectories.

Quantization is a popular method for traditional compression, and for nearest neighbor searches on multi dimensional data, especially for multimedia and computer vision applications [12, 27, 33, 40, 43]. Predictive quantization has been applied for online summarization of multiple one-dimensional data streams [1]. The correlation among consecutive points is employed to predict current points, then the prediction errors are summarized into a smaller number of bits [1]. Product Quantization and Residual Quantization [8, 15, 19] have made significant impact on approximate nearest neighbor searching in computer vision applications. We included these two methods in our performance evaluation. There have been some work to use quantization for encoding trajectories [7], transforming differential trajectory points into strings for compression [29], and retaining information for trajectory prediction [5]. These methods adopt compression but with no particular support for efficient querying over compressed trajectories. Our goal is to quantize dynamic trajectories into an error-bounded and query friendly representation, where there is neither need to fully reconstruct nor traverse the full trajectories. Trajectory data is summarized online, exploiting their large-scale nature, for the purpose of efficient query processing.

3 ONLINE QUANTIZATION IN PPQ-TRAJECTORY

In this section, we present our spatio-temporal quantization based summarization process. The performance measures are the accuracy of results when the summary is used directly to provide approximate query results, the spatial deviation with which spatio-temporal path queries can be answered when the summary is used as an index, and the time taken to construct the summary. The quality of the summary and the compression ratio are also related measures. Table 1 summarizes the notation used throughout the paper. Basic definitions of trajectories and codebooks are as follows.

Definition 3.1. (Trajectory) A trajectory T is a finite sequence of time-stamped positions in the form of $((x_1, y_1, t_1), (x_2, y_2, t_2), ..., (x_n, y_n, t_n))$, where $0 \leq t_i \leq t_j \leq t_n$ with $0 \leq i \leq j \leq n$.

Definition 3.2. (Error-bounded Codebook) Consider a codebook $C = \{C_1, ..., C_n\}$ where the set of trajectory points $\mathcal{T}^i$ is indexed by codeword C_i. For any trajectory point $T_j^t \in \mathcal{T}^i$, if $\left\|T_j^t - C_i\right\|_2 \leq \varepsilon_1$, then the codebook C is bounded with ε_1.

3.1 Error-bounded Predictive Quantization

We first present the error bounded predictive quantizer (E-PQ) to obtain a compact codebook for trajectories. Predictive quantization (PQ) has been successfully applied for one-dimensional data streams

Table 1: Summary of Notation

Variable	Definition
T_i	the i-th trajectory
T_i^t	trajectory point of T_i at time t
T^t	trajectory points at time t
$\widehat{T}_i^t$	reconstructed trajectory point of T_i^t
$\widetilde{T}_i^t$	predicted trajectory point of T_i^t
e_i^t	prediction error of T_i^t, i.e., $T_i^t - \tilde{T}_i$
f	a prediction function
C	error bounded codebook
b_i^t	codeword index for e_i^t
ε_1	spatial deviation threshold
ε_1^M	ε_1 under the geographic coordinate system
ε_p	partition threshold for PPQ
ε_s	partition threshold for constructing index
$\hat{T}_i^{t}{}'$	reconstruction of T_i^t considering CQC

by quantizing the error of the estimate of the sample at time t with previous k samples, i.e., $\widetilde{x}[t] = f(x[t-1], x[t-2], ..., x[t-k])$ [1, 13]. A prediction function f is learned over training data, and a codebook is generated via a vector quantizer [40] on the prediction errors by assigning them to the nearest centroids of their clusters. The range of the error $e[t] = x[t] - \widetilde{x}[t]$ is narrower than the original data which enables the errors to be quantized more effectively than the original data [1].

To estimate the trajectory points using their correlations, we define a prediction function as an extension to the case for one-dimensional streams [1]. For ease of demonstration, we define f as a linear model that predicts T_i^t using the previous k samples. The prediction is computed as:

$$\min_f \sum_{i=1}^{N} \left\| T_i^t - f(T_i^{(t-k:t-1)}) \right\|_2 \tag{1}$$

where T_i^t represents the position (x_t, y_t) of T_i at time t, $T_i^{(t-k:t-1)}$ is the sequence of the trajectory T_i at time interval $[t-k, t-1]$, and f denotes the prediction model.

The prediction error e_i^t is defined as:

$$e_i^t = T_i^t - \widetilde{T}_i^t, \quad \widetilde{T}_i^t = \sum_{j=1}^{k} P_j[t] \widehat{T}_i^{t-j} \tag{2}$$

where $\widetilde{T}_i^t$ is the prediction of T_i^t, $P_j[t]$ is the j-th prediction coefficient of f, and $\widehat{T}_i^{t-k}$ is the reconstruction of T_i^{t-k}.

The prediction errors $\{e_i^t\}$ can be summarized into an error-bounded codebook C as:

$$\begin{aligned} &\min |C| \\ &\text{s.t.} \left\| e_i^t - C(b_i^t) \right\|_2 \le \varepsilon_1, b_i^t \in \{1, ..., V\} \end{aligned} \tag{3}$$

where $C = \{C_1, C_2, ..., C_V\}$ is the error-bounded codebook, V denotes the size of C, b_i^t is the codeword index for e_i^t, and $C(b_i^t)$ denotes the codeword assigned to represent e_i^t. Every codeword $C_i \in C$ is a cluster centroid, obtained by partitioning the data space to facilitate indexing and compression. Equation 3 aims to achieve a minimal error-bounded codebook C for the given ε_1, which is non-convex. For dynamic databases, C needs to be incrementally updated with evolving t values. In order to get the approximate solution, at time $t+1$, if part of the prediction errors $\{e_i^{t+1}\}$ can not satisfy the threshold, the additional codewords are added to update C to guarantee the boundary requirement continuously.

Algorithm 1 Error-bounded Predictive Quantization

```
Input: $\{T_i^t\}$, $\varepsilon_1$
Output: $\{P_j[t]\}$, $C$, $\{b_i^t\}$
1: t = 1
2: while $\{T_i^t\}$ is not empty do
3:     Derive $\{P_j[t]\}$ based on Equation 1
4:     $\widetilde{T}_i^t = \sum_{j=1}^k P_j[t] \widehat{T}_i^{t-j}$
5:     $\{e_i^t\} = \{T_i^t\} - \{\widetilde{T}_i^t\}$
6:     $C, \{b_i^t\} = Incremental\_Quantizer(\{e_i^t\}, C, \varepsilon_1)$
7:     $\hat{T}_i^t = \widetilde{T}_i^t + C(b_i^t)$
8:     t = t + 1
```

The reconstructed T_i^t, $\hat{T}_i^t$, is obtained where:

$$\widehat{T}_i^t = \widetilde{T}_i^t + C(b_i^t) \tag{4}$$

The procedure of quantizing dynamic trajectories is summarized in Algorithm 1. In Line 3, the prediction coefficient $P_j[t]$ can be solved in a standard manner [1, 16]. Line 4 denotes the prediction of the t-th trajectory point by its previous k reconstructed points. For the time $t \le k$, $P_j[t]$ is set to zero. At Line 6, *Incremental_Quantizer* represents the quantization process of Equation 3. E-PQ maps the trajectory data into $\{P_j[t]\}$, C, $\{b_i^t\}$.

3.2 Partition-wise Predictive Quantization

We now present our quantizer that partitions trajectory points and applies E-PQ for each partition. The partition-wise predictive quantization (PPQ) is formulated as:

$$\mathcal{N}^t = \{\mathcal{N}_1^t, ..., \mathcal{N}_q^t\} \tag{5}$$

$$\min \sum_{i \in \mathcal{N}_j^t} \left\| T_i^t - f_j(T_i^{t-k:t-1}) \right\|_2, \mathcal{N}_j^t \in \mathcal{N}^t \tag{6}$$

In Equation 5, N trajectory points $\{T_i^t\}$ are partitioned into q subsets, where $\mathcal{N}_j^t$ denotes the set of trajectory IDs assigned to the j-th partition. $f_j \in \{f_1, ...f_q\}$ is the prediction function for $\mathcal{N}_j^t$. This enables the use of a single prediction function f_j for trajectory points of $\mathcal{N}_j^t$, and is resolved by Equation 6. Via $\{f_1, ...f_q\}$, the correlation among consecutive trajectory points in every $\mathcal{N}_j^t \in \mathcal{N}^t$ is modeled by a specific f_j, then the dynamic range of prediction errors is further narrowed down. When $q = 1$, Equation 1 and Equation 6 become the same. Similarly for the $\{e_i^t\}$ obtained from multiple predictions, they are summarized with Equation 3.

3.2.1 Partitioning for Grouped Modeling. We partition the trajectory points using their spatial and autocorrelation similarities. Tobler's first law of geography indicates "everything is related to everything else, but nearby things are more related than distant things" [39]. Hence, assigning trajectory points based on spatial proximity is a natural approach to be able to use the same f_j to model them. However, as the role of f_j is to capture the correlations between consecutive trajectory points, assigning trajectory points with similar autocorrelations to $\mathcal{N}_j^t$ can enable a more accurate prediction by f_j. In our setting, the correlation between T_i^t and $T_i^{t-k:t-1}$ follows an autoregressive process of order k (AR(k)) [9, 34], where the current trajectory point (T_i^t) is linearly related to the lagged k points ($T_i^{t-k:t-1}$). We derive the parameters of AR(k) as $\{a_i^t\}$ and utilize them to quantity the lag-k autocorrelation. Assigning trajectory points with similar $\{a_i^t\}$ to the same partition $\mathcal{N}_j^t$ allows f_j to more effectively capture the correlations between consecutive trajectory points.

The partitioning process is repeated until all partitions satisfy Equations 7 and 8, for spatial and autocorrelation similarity, respectively. For the spatial proximity, the deviation between any point in $\mathcal{N}_j$ and the centroid of $\mathcal{N}_j$ should be less than ε_p, otherwise, q increases until Equation 7 is satisfied.

$$\left\|T_i^t - centroid_S(\mathcal{N}_j^t)\right\|_2 \leq \varepsilon_p, \text{for all } \mathcal{N}_j^t \in \mathcal{N}^t, i \in \mathcal{N}_j^t \tag{7}$$

Similarly, for the autocorrelation similarity, the partitions satisfy Equation 8, where a_i^t represents the lag-k autocorrelation of T_i^t, and $centroid_A(\mathcal{N}_j^t)$ is the centroid of the autocorrelation of trajectory points in $\mathcal{N}_j$.

$$\left\|a_i^t - centroid_A(\mathcal{N}_j^t)\right\|_2 \leq \varepsilon_p, \text{for all } \mathcal{N}_j^t \in \mathcal{N}^t, i \in \mathcal{N}_j^t \tag{8}$$

The setting of ε_p is based on the size of the region that trajectories span (for spatial proximity), or the magnitude and distribution of autocorrelation coefficients (for autocorrelation similarity).

The computational complexity for q partitions is $O(qmNl)$ as shown in Lemma 1, where N is the number of trajectory points, m denotes the rounds of increasing q to satisfy Equation 7 or 8, and l represents the number of iterations to obtain the given number of partitions by K-means [28]. The complexity is proportional to q.

3.2.2 *Incremental Temporal Partitioning.*

Consider the partitions at time t, i.e., $\mathcal{N}^t = \{\mathcal{N}_1^t, ..., \mathcal{N}_q^t\}$. Instead of performing partitioning from scratch, an incremental partitioning for time $t+1$ is performed with the following steps. First, every trajectory point at time $t+1$, i.e., $\{T_i^{t+1}\}$, is assigned to the same partition as T_i^t. Second, when a partition, $\mathcal{N}_j^{t+1}$, does not satisfy the requirement for ε_p, a new partitioning is performed over trajectory points in $\mathcal{N}_j^{t+1}$ until the resultant partition satisfies the requirement. Third, with $\mathcal{N}_j^{t+1}$ and $\mathcal{N}_{j'}^{t+1}$, if $\left\|centroid_{S/A}(\mathcal{N}_j^{t+1}) - centroid_{S/A}(\mathcal{N}_{j'}^{t+1})\right\|_2 \leq \varepsilon_p$, we merge $\mathcal{N}_j^{t+1\prime}$ to $\mathcal{N}_j^{t+1}$ to avoid too many fragmented partitions. Specifically, for every partition $\mathcal{N}_j^{t+1}$, we only allow merging at most once, as excessive merging might influence the preciseness of partitioning and the quantization performance. If there are N' trajectory points at $t+1$ that do not satisfy the requirement for ε_p, q' new partitions are generated via an m' rounds of checking with Equation 7 or 8, then the computational complexity of the incremental temporal partitioning is $O(q'm'N'l + q'q)$, which is presented in LEMMA 2. q' is only relevant to the distribution of the N' trajectory points. N' gets smaller when the points among the consecutive timestamps are highly similar in autocorrelations or

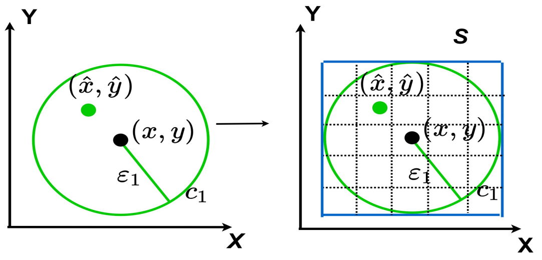


Figure 2: An error space example for the reconstructed trajectory point $(\hat{x}, \hat{y})$

spatially close. In the worst case, when all the N trajectory points at time $t+1$ do not satisfy the q partitions at time t, i.e., $N = N'$, the computational complexity of incremental temporal partitioning becomes $O(qmNl)$.

LEMMA 1. *The computational complexity of partitioning $\{T_i^t\}$ into q partitions (Section 3.2.1) is $O(qmNl)$.*

PROOF. Let q_i ($i \in [1, m]$) be the number of partitions at the i-th round, which increases by a at every round until all the partitions satisfy Equation 7 or 8. Hence, $q = q_m = ma$. For the i-th round, the computational complexity is the same as partitioning N trajectory points into q_i clusters, i.e., $O(q_iNl)$. Then, the overall computational cost is $aNl + ... + maNl = \frac{am(1+m)}{2}Nl$, i.e., $O(qmNl)$. □

LEMMA 2. *The computational complexity of incremental temporal partitioning for $\{T_i^{t+1}\}$ is $O(q'm'N'l + q'q)$.*

PROOF. According to LEMMA 1, the complexity of partitioning N' trajectory points into q' partitions is $O(q'm'N'l)$. For q' new partitions at time $t+1$, in the worst case, there will be $q + ... + (q - (q' - 1)) = \frac{q'(q+(q-(q'-1)))}{2}$ computations to check if a new partition can be merged into the existing q partitions. Hence, the overall complexity is $O(q'm'N'l + q'q)$. □

4 LOCAL CODING IN ERROR-BOUNDED CODEBOOK

With the error-bounded codebook, the reconstructed value $(\hat{x}, \hat{y})$ is guaranteed to be within the circle c_1, as shown in Figure 2. While a small ε_1 is desirable for the accuracy of approximate query results, an excessively small ε_1 would degrade the effectiveness of quantization, both in terms of the efficiency of learning and the size of the codebook. Here, we present the coordinate quadtree coding (CQC), which encodes the spatial deviation between (x, y) and $(\hat{x}, \hat{y})$, to reduce the information loss of the summary.

CQC consists of short binary codes that can be easily restored to the relative position between (x, y) and $(\hat{x}, \hat{y})$ to obtain an accurate trajectory reconstruction $(\hat{x}', \hat{y}')$. The construction of the coordinate quadtree and getting the CQC are independent of the dataset size when ε_1 and g_s are fixed.

Definition 4.1. (Coordinate Quadtree) A coordinate quad-tree is a tree structure in which each internal node has four children nodes, the value of a node is the coordinate of the subspace that

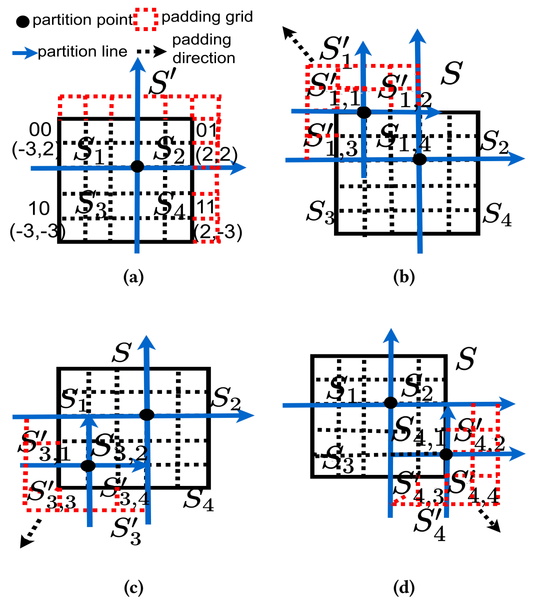


Figure 3: Padding Example

Algorithm 2 Coordinate_QuadTree(c_1, g_s)

```
Output: $Cq$
1: Get the minimum rectangle $S$ covering $c_1$
2: $S$ is split into grids of equal size, $S_{g_s}$.
3: $Cq = \{\}$ # Coordinate quadtree
4: $build\_tree(S_{g_s}, Cq)$.
```

Function build_tree($\{S_{g_s,i}\}, Cq$)

```
Output: $Cq$.
1: for all each sub-region $S_{g_s}$ in $\{S_{g_s,i}\}$ do
2:     $s_x, s_y \leftarrow |S_{g_s}|$ # $s_x$ and $s_y$ are the number of grid cells of $S_{g_s}$ along the $x$- and $y$-axes respectively
3:     if ($s_x, s_y = 1 \& no\_padding(S_{g_s})$) or $s_x, s_y = 0$ then
           continue
4:     $\{\dot{S}_{g_s,i}\} \leftarrow$ partition_padding($S_{g_s}$)
5:     $Cq.append(\{\dot{S}_{g_s,i}\})$.
6:     $build\_tree(\{\dot{S}_{g_s,i}\}, Cq)$
```

the node represents, and the value over the edge is the quadrant that its parent node is located in.

Definition 4.2. (Coordinate Quadtree Coding) Given a coordinate quadtree, the coordinate quadtree coding (CQC) of a node n_q is the values of the edges from the root node to node n_q as well as the quadrant that n_q is located in.

4.1 Coordinate Quadtree Coding

The process of building the coordinate quadtree, which is used as the basis for CQC, is summarized in Algorithm 2. The first step is to get the error space c_1 and find the minimum rectangle S covering

Function *partition_padding*(S_{g_s})

```
Output: $\{S_{g_s,i}\}$.
1: if $\lfloor \frac{s_x}{2} \rfloor \neq \frac{s_x}{2}$ or $\lfloor \frac{s_y}{2} \rfloor \neq \frac{s_y}{2}$ then
2:     $S'_{g_s} \leftarrow$padding($S_{g_s}$)
3:     Partitioning $S'_{g_s}$ into $\{S_{g_s,i}\}$
4: else
5:     Partition $S_{g_s}$ into $\{S_{g_s,i}\}$ # four equal partitions
```

c_1 in Line 1. The second step is to divide S into grid cells of equal size, S_{g_s}, in Line 2, where g_s is the size of a cell. The third stage is to build the coordinate quadtree via Function *build_tree*. Its stopping condition is either when the subspace is empty, or the input subspace is size one and without any padding grid cells, as shown in Line 3. For any S_{g_s}, its partitions are generated by *partition_padding* in Line 4. An example is given in Figure 3. For simplicity, we omit g_s and use S'_i and S_i to demonstrate the example. For function *partition_padding*, if S does not satisfy Line 1, we pad S as S' so that S' can be partitioned into four equal partitions. To avoid conflicts of partitions at different rounds, we design specific padding rules for different quadrants. We have a 5×5 grid S in Figure 3a, S is expanded to a 6×6 grid S' to obtain four equal size subspaces. According to Definition 4.1, the values (-3,2), (2,2), (-3,-3) and (2,-3) represent the size and quadrant for every subspace. The quadrants are encoded as 00, 01, 10 and 11 separately. As shown in Figure 3b, the subspace S_1 at quadrant 00 is expanded towards the upper left as S'_1 which can be further partitioned. Similarly, the subspaces at quadrant 10 and 11, i.e., S_3 and S_4, are padded towards the bottom left and bottom right respectively, as shown in Figure 3c and Figure 3d. However, there is no need to pad S_2 as it can be directly partitioned into four subspaces of equal size following Line 5. The full process of building a coordinate quadtree for 5×5 grids is shown in Figure 4a. The corresponding coordinate quadtree can be observed in Figure 4b. The value in every square corresponds to the coordinate of a subspace it denotes, which is denoted as SC. "X" denotes the empty padding grid. According to Definition 4.2, the CQC for the n_1 in Figure 4 is 001110.

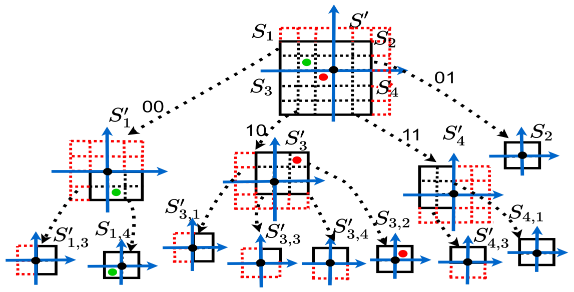
For a CQC, the real coordinate is the sum of the values of the nodes it visits, as shown in Equation 9.

$$c_{cqc} = \sum_{j=1}^{m} \frac{1}{2} SC'_j \tag{9}$$

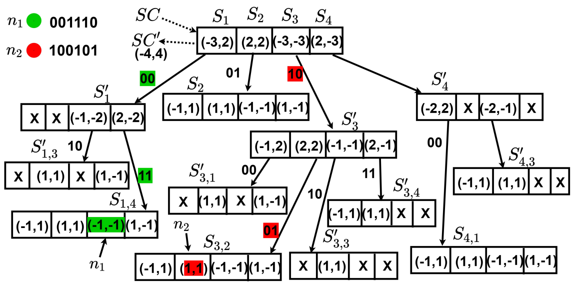
$$SC' = \begin{cases} SC & |x| = |y| = 1 \\ 2\left\lceil \frac{\max(|x|,|y|)}{2} \right\rceil \cdot (\text{sgn}(x), \text{sgn}(y)) & \text{otherwise} \end{cases} \tag{10}$$

where SC' denotes the SC of the padded subspace. SC' is obtained by Equation 10. For example, in Figure 4b, a SC is (-3,2), its SC' is (-4,4) by Equation 10. According to Equation 9, the real coordinate for n_1 is $(\frac{-3}{2}, \frac{1}{2})$.

If S_{g_s} cannot be partitioned into four subspaces with the same number of grid cells, the traditional approach for quadtrees [37] would extend S_{g_s} and maintain the empty grids. However, the padding process guarantees to produce four equally sized partitions. We utilize the coordinate of the subspace, to keep track of the real size of every subspace and the relative displacement of grid cells among the subspaces at different levels, and make sure



(a) Coordinate Quadtree Partitioning Process



(b) Coordinate Quadtree

Figure 4: Coordinate Quadtree Coding Example

one can reversely restore the real position of any grid cell in S_{g_s} by Equation 9–10.

4.2 Trajectory Reconstruction with CQC

When ε_1 and g_s are given, (x, y) is fixed at the center cell of S_{g_s}, i.e, n_2 in Figure 4. Its CQC is represented as cqc_1. The CQC of $(\hat{x}, \hat{y})$ is denoted as cqc_2. With cqc_1 and cqc_2, the reconstructed trajectory point for (x, y) is obtained as:

$$(\hat{x}', \hat{y}') = (\hat{x}, \hat{y}) + g_s \cdot (c_{cqc_1} - c_{cqc_2}) \tag{11}$$

where c_{cqc_1} and c_{cqc_2} are CQC of (x, y) and $(\hat{x}, \hat{y})$, obtained by Equation 9.

As shown in Figure 4b, (x, y) is denoted as n_2, $(\hat{x}, \hat{y})$ is represented as n_1. According to Equation 9, the real coordinates of (x, y) and $(\hat{x}, \hat{y})$ at the S_{g_s} are $c_{cqc_1} = (-\frac{1}{2}, -\frac{1}{2})$ and $c_{cqc_2} = (-\frac{3}{2}, \frac{1}{2})$, respectively. Then, when $(\hat{x}, \hat{y})$ is known, the reconstructed point, $(\hat{x}', \hat{y}')$, can be obtained using Equation 11. Given ε_1 and g_s, a unified and fixed coordinate quadtree is obtained. The coordinate quadtree is stored as a template to recover the position with the given CQC. For any original trajectory points (x, y), they have the same CQC, i.e., cqc_1, as they are always located at the center of the same grid of S_{g_s}. Hence, only the coding of $(\hat{x}, \hat{y})$, i.e., cqc_2, is stored for every trajectory point sample. $(\hat{x}, \hat{y})$ is recovered from the summary produced by our approach. It is easy to prove that deviation has been reduced within $\frac{\sqrt{2}}{2} g_s$ as shown in Lemma 3.

Lemma 3. *Given the grid size g_s, the trajectory point (x, y), the error of the accurate reconstructed trajectory $(\hat{x}', \hat{y}')$ does not exceed $\frac{\sqrt{2}}{2} g_s$, i.e., $0 \le \|(x, y) - (\hat{x}', \hat{y}')\|_2 \le \frac{\sqrt{2}}{2} g_s$.*

Proof. As shown in Figure 2, the continuous subspace is quantized into the grid cell of size g_s. Any points falling into the same grid cell are quantized into the same value, e.g., the center of that grid cell. Hence, the maximum error between the quantized value and the real value is $\frac{\sqrt{2}}{2} g_s$, i.e., half of the length of the grid diagonal. According to Equation 11, the deviation over the quantized values has been kept by CQC. With CQC, the unmeasured error is just the deviation introduced by the quantized process of $(\hat{x}, \hat{y})$, i.e., the maximum is $\frac{\sqrt{2}}{2} g_s$. Hence, we get $0 \le \|(x, y) - (\hat{x}', \hat{y}')\|_2 \le \frac{\sqrt{2}}{2} g_s$. □

5 ONLINE QUERYING OVER QUANTIZED TRAJECTORIES

The last part of PPQ-trajectory is the organization of quantized spatio-temporal data for online querying to obtain the candidate set without the reconstruction of all trajectories. A simple spatio-temporal query example is to search vehicles that travel through location (x, y) at time t, and estimating their next l positions. We focus on two commonly used spatio-temporal queries, as presented later in the paper.

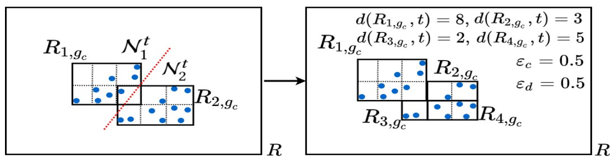
Since the parameters in the system $(\{P_j[t]\}, C, \{b_i^t\}, \text{CQC})$ are enough to reproduce any trajectory, the naive solution is to reconstruct the trajectories from time 1 to t and return the trajectories that match the query conditions. In Section 5.1, we propose a temporal partition-based organization to enable direct access to the relevant trajectory IDs at time t, and get their next l positions in an online manner. In Section 5.2, we illustrate the spatio-temporal query process over quantized trajectories, and a local search strategy motivated by CQC is introduced to achieve the accurate query.

5.1 Temporal Partition-based Index

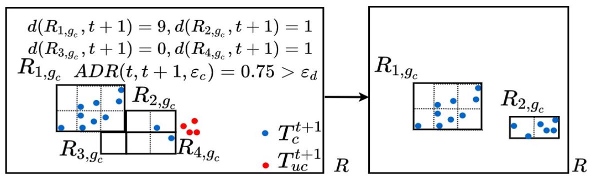
Given the non-uniform nature of trajectory data, a temporal partition-based index (TPI) over the quantized trajectories is constructed to prune the search space. TPI can actually be applied for any of T_i^t, $\hat{T}_i^t{}'$ and $\widehat{T}_i^t$, for simplicity, we use T_i^t to illustrate, which is interchangeable with $\hat{T}_i^t{}'$ and $\widehat{T}_i^t$. The process of constructing the partition-based index (PI), at time t, is described in Algorithm 3. An example for PI at t is given in Figure 5a. In Line 1, the trajectory points at time t, i.e., T^t, are partitioned into q_s subsets following the same principles as Equation 7 while replacing ε_p with ε_s, where ε_s is the partition threshold for region R. Specifically, the setting of ε_s depends on the size of the region R we operate on. For every subset $\mathcal{N}_j^t$, we find the minimum rectangle R_n to cover the trajectory points of $\mathcal{N}_j^t$, as shown in Line 5. For example, points in Figure 5a are split into two partitions, i.e., $\mathcal{N}_1^t$ and $\mathcal{N}_2^t$. The minimum rectangle R_1 is found to cover trajectory points of $\mathcal{N}_1^t$. Note that overlap between R_ns might exist. In Lines 7–8, to avoid duplicate indexes for some points, the overlapping region is removed, then the left polygon is separated into non-overlapping rectangles by the approach in [17], which is denoted by the function *remove_overlap*. As shown in Figure 5a, R_2 overlaps with R_1, the overlapping region is removed, and the left polygon is separated into R_2, R_3 and R_4. As shown in Line 11, we build a grid index [41, 42] for the set $\{R_1, ..., R_n\}$ by Function *grid_index*. Specifically, for every subregion R_j, it is partitioned into cells of g_c [23]. Every trajectory point T_i^t is then mapped to the corresponding grid cell, and its trajectory ID is stored into the grid

Algorithm 3 PI

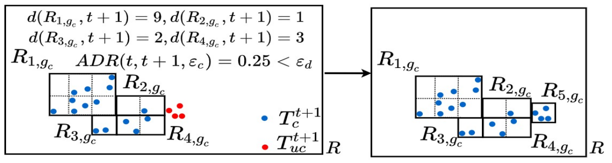
```
Input: $T^t$, partition threshold $\varepsilon_s$, grid size $g_c$
Output: $pi_t$
1: Get $q_s$ partitions $\{\mathcal{N}_1^t, ..., \mathcal{N}_{q_s}^t\}$ #following Equation 7 while
   replace $\varepsilon_p$ with $\varepsilon_s$
2: $region\_list$ = {}, $n$ = 0
3: for each $\mathcal{N}_j^t$ in $\{\mathcal{N}_1^t, ..., \mathcal{N}_{q_s}^t\}$ do
4:     $n = n + 1$
5:     Find the minimum rectangle $R_n$ covering trajectory points
       of $\mathcal{N}_j^t$ .
6:     if $R_n$ overlaps with rectangles in $region\_list$ then
7:         $\{R_n, ..., R_{n+l-1}\}$=$remove\_overlap(R_n)$,
8:         $region\_list$.append($\{R_n, ..., R_{n+l-1}\}$), $n$=$n$+$l$-1.
9:     else
10:        $region\_list$.append($R_n$)
11: $\{R_{1,g_c}, ..., R_{n,g_c}\}$ = $grid\_index(\{R_1, ..., R_n\})$
12: return $pi_t$=$\{R_{1,g_c}, ..., R_{n,g_c}\}$
```



(a) A PI example at t


$d(R_{1,g_c},t+1)=9$, $d(R_{2,g_c},t+1)=1$
$d(R_{3,g_c},t+1)=0$, $d(R_{4,g_c},t+1)=1$
$ADR(t,t+1,\varepsilon_c)=0.75>\varepsilon_d$
R_{1,g_c}
R_{2,g_c}
R_{3,g_c}
R_{4,g_c}
● T_c^{t+1}
● T_{uc}^{t+1}
R

(b) "Re-build" case at $t + 1$


$d(R_{1,g_c},t+1)=9$, $d(R_{2,g_c},t+1)=1$
$d(R_{3,g_c},t+1)=2$, $d(R_{4,g_c},t+1)=3$
$ADR(t,t+1,\varepsilon_c)=0.25<\varepsilon_d$
R_{1,g_c}
R_{2,g_c}
R_{3,g_c}
R_{4,g_c}
R_{5,g_c}
● T_c^{t+1}
● T_{uc}^{t+1}
R

(c) "Insertion" case at $t + 1$

Figure 5: An illustrative example of TPI

cell. To reduce the storage cost, we compress trajectory IDs mapped to the grid cell by delta encoding and Huffman codes, following the approach in the other works [19, 22, 42]. The PI at time t, i.e., pi_t, is returned as shown in Line 12.

We avoid building PI from scratch for every timestamp to efficiently maintain dynamic trajectories. For example, at time t_s, T^{t_s} are indexed by pi_{t_s}. At time t_e, part of pi_{t_s} might be reused for T^{t_e}, as the distributions among consecutive timestamps might not change sharply.

Algorithm 4 TPI

```
Input: Trajectory dataset $T$, density error threshold $\varepsilon_d$, partition
   threshold $\varepsilon_s$, TRD dropping rate threshold $\varepsilon_c$, grid size $g_c$
Output: Time periods $\{period_j\}$ and PIs $\{pi_j\}$
1: $t_s$ = 1, $t_e$ =1, j = 1
2: $pi_j = PI(T^{t_e}, \varepsilon_s, g_c)$
3: $t_e = t_e + 1$.
4: while data input at $t_e$ do
5:     $T^{t_e} = T_c^{t_e} \cup T_{uc}^{t_e}$
6:     if ADR($t_s$,$t_e$,$\varepsilon_c$) > $\varepsilon_d$ then
7:         $period_j$.s = $t_s$, $period_j$.e = $t_e$ -1.
8:         $j = j + 1$, $t_s = t_e$.
9:         $pi_j = PI(T^{t_e}, \varepsilon_s, g_c)$ # Re-build
10:    else if $T_{uc}^{t_e}$ is non-empty then
11:        $pi_j$.append($PI(T_{uc}^{t_e}, \varepsilon_s, g_c)$) #Insertion
12:    $t_e = t_e + 1$
13: return $\{period_j\}$ and $\{pi_j\}$
```

Definition 5.1. (Trajectory Region Density (TRD)) Given T^t and its PI pi_t=$\{R_{1,g_c}, ..., R_{n,g_c}\}$, for R_{i,g_c}, its TRD is $d(R_{i,g_c}, t) = \frac{N_{R_{i,t}}}{|R_{i,g_c}|}$, where $|R_{i,g_c}|$ denotes the size of rectangle R_i, $N_{R_{i,t}}$ is the number of trajectories indexed by R_{i,g_c} at time t.

The definition of TRD quantifies the occupancy rate of subregions, providing the basis of building the temporal index flexibly. According to Definition 5.1, we compute the average dropping rate (ADR) of TRD by Equation 12, to measure reusing the previous index pi_{t_s} or constructing a new index. For example, at time t_s, its index is pi_{t_s} = $\{R_{1,g_c}, ..., R_{n,g_c}\}$. The TRD of subregion R_{i,g_c} at time t_s, i.e., $d(R_{i,g_c}, t_s)$, is obtained. For trajectory points at time t_e, the new TRD $d(R_{i,g_c}, t_e)$ is computed. For R_{i,g_c}, the dropping rate of its TRD from t_s to t_e can be obtained with Equation 13. For R_{i,g_c}, if the dropping rate of its TRD, i.e., $h_1(R_{i,g_c}, t_e, t_s)$, exceeds the threshold ε_c, then it counts for ADR since the occupancy rate of R_{i,g_c} drops too much, which is achieved by Equation 14.

$$ADR(t_s, t_e, \varepsilon_c) = \sum_{i=1}^{n} \frac{\mathrm{h}(\mathrm{h}_1(\mathrm{R}_{i,g_c}, \mathrm{t_e}, \mathrm{t_s}), \varepsilon_c)}{\mathrm{n}} \tag{12}$$

$$h_1(R_{i,g_c}, t_e, t_s) = \frac{d(R_{i,g_c}, t_e) - d(R_{i,g_c}, t_s)}{d(R_{i,g_c}, t_s)} \tag{13}$$

$$h(x, \varepsilon_c) = \begin{cases} 1 & x < 0 \quad and \quad |x| > \varepsilon_c \\ 0 & \text{others} \end{cases} \tag{14}$$

Algorithm 4 presents the TPI. For T^t, the initial index pi_t is obtained by PI (lines 1–3). As mentioned above, Figure 5a shows a PI example at time t. At next timestamp t_e, T^{t_e} is partitioned into two parts, i.e., $T_c^{t_e}$ and $T_{uc}^{t_e}$ (line 5), where $T_c^{t_e}$ is the set covered by pi_j, $T_{uc}^{t_e}$ is the set that are not covered by pi_j. For the covered trajectory set $T_c^{t_e}$, if its ARD exceeds ε_d, the current index pi_j can not index T^{t_e} efficiently. Then a new PI is built for T^{t_e} as shown in Line 6–9, which is denoted as "Re-build" in the experimental study. Otherwise, we only construct the new *PI* for $T_{uc}^{t_e}$, i.e., "Insertion" in the experiments, and according to Line 10–11, the current index, pi_j, is updated. Finally, we obtain a set of time periods $\{period_j\}$ and corresponding PIs $\{pi_j\}$. A larger ε_d would lower the frequency of "Re-build", i.e., a higher tolerance for decreasing of TRD reduces "Re-build"s. A smaller ε_d, the operation of "Re-build" will be more frequent due to the strict constraints for ADR. An example is presented in Figure 5, with $\varepsilon_c = 0.5$, $\varepsilon_d = 0.5$, $|R_{i,g_c}| = 1$ (in Definition 5.1), and $d(R_{i,g_c}, t) = N_{R_{i,t}}$, i.e., the number of nodes in R_{i,g_c} at time t. As shown in Figure 5b, TRD at $t+1$ has changed, i.e., $d(R_{i,g_c}, t+1)$, and according to Equation 12–14, we get $ADR(t, t+1, \varepsilon_c) = 0.75 > \varepsilon_d$, i.e., the average dropping rate of TDR exceeds the threshold, which means the PI at t can not be further reused to index trajectories at time $t+1$, hence a PI is rebuilt. However, in Figure 5c, $ADR(t, t+1, \varepsilon_c) = 0.25 < \varepsilon_d$, then we can hold the PI at t, i.e., $\{R_{1,g_c}, R_{2,g_c}, R_{3,g_c}, R_{4,g_c}\}$, and only build a new PI for trajectory points that are not covered by PI at t, i.e., indexing the uncovered trajectory points T_{uc}^{t+1} in R_{5,g_c}, which is "Insertion".

The merits of TPI are two-fold. First, based on the dynamic trajectory density, spatio-temporal trajectories are indexed as a set of periods, which lowers the frequency of partitioning the spatial region. Second, with TPI, the accuracy of the partitions within a certain time period is guaranteed by the measurement of ARD.

For disk-resident data, the trajectory points within a time period can be written into several pages and the corresponding part of the summary, i.e., ($\{P_j[t]\}$, C, $\{b_i^t\}$, CQC), is assigned to the corresponding page. A lightweight index for the assigned page number is used to record the assigned pages for the trajectory points of $period_j$, and the corresponding summary, i.e., ($period_j$, starting page number, relative page number).

5.2 Spatio-temporal Query Processing

We now present how PPQ-trajectory answers spatio-temporal queries. We illustrate our approach using Spatio-temporal Range Query (STRQ) and Trajectory Path Query (TPQ).

Definition 5.2. (Spatio-temporal Range Query (STRQ)) Given time t and location (x, y), STRQ retrieves trajectories which are located at the grid cell that (x, y) is in at time t.

For STRQ, given a query (x, y, t_1), if $t_1 \in period_j$, the sub-regions of pi_j are obtained, e.g., R_{i,g_c}. (x, y) is mapped to a grid of R_{i,g_c}, then a list of trajectory IDs at time t_1 is returned.

Definition 5.3. (Trajectory Path Query (TPQ)) Given time t, location (x, y) and the path duration l, the trajectory IDs of the STRQ of (x, y, t) are first retrieved, then their sub-trajectories at the time interval [t,t+l] are returned.

For TPQ, given time t, location (x, y) and path duration l, the list of trajectory IDs is first returned from STRQ by searching (x, y, t), the next l positions of the retrieved trajectories are then directly reproduced by the indexed summary.

Local Search using CQC. For a trajectory point (x, y), its accurate reconstructed trajectory point is $(\hat{x}', \hat{y}')$. According to Lemma 3, $(\hat{x}', \hat{y}')$ might be any point within a circle of which radius is $\frac{\sqrt{2}}{2} g_s$ and (x, y) is the center. With the CQC, the deviation has been narrowed to $\frac{\sqrt{2}}{2} g_s$ which is a smaller distance. In order to further improve the accuracy of queries using the summary, we introduce a local search strategy. There are two situations for the local search. (1) When $\frac{\sqrt{2}}{2} g_s > g_c$, as shown in Figure 6a, when all the grid cells g_c that are covered by the circle are scanned, the full actual result

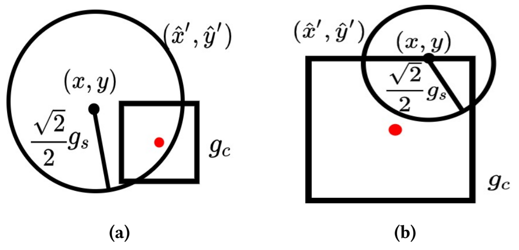


Figure 6: Illustrating the Query Space with CQC

can be retrieved successfully. (2) When $\frac{\sqrt{2}}{2} g_s \le g_c$, as shown in Figure 6b, the worst case is that $(\hat{x}', \hat{y}')$ falls out of the grid cell g_c as (x, y) happens to be adjacent to the border of the grid cell. Hence, in order to guarantee that the actual result is retrieved, the quantized trajectory points of which distance to (x, y) is less than $\frac{\sqrt{2}}{2} g_s$ in the grid cells that are adjacent to the grid cell (x, y) mapped to are all scanned. Specifically, the second case is more general, i.e., g_s is smaller, as the aim of introducing g_s is to further reduce the information loss. For g_c, it serves for the index part, i.e., TPI, hence, it is usually larger.

With the local search, the returned candidate list contains all the trajectory IDs for STRQ, which makes the recall 1. However, the candidate list may include the trajectory IDs of which true position is at the boundary of adjacent cells of g_c while mapped to g_c. The $(\hat{x}', \hat{y}')$ whose distance to (x, y) is larger than $\frac{\sqrt{2}}{2} g_s$ has been filtered out according to LEMMA 3. The precision for STRQ can be improved to be 1 by accessing the original trajectory of the candidate list of which size is relatively small due to the accuracy of $(\hat{x}', \hat{y}')$.

6 EXPERIMENTAL EVALUATION

We conduct a range of experiments to evaluate the effectiveness of PPQ-trajectory and compare it with a variety of alternative approaches. Our methods and all the alternative approaches are implemented in Matlab R2019a. All experiments are executed on a Ubuntu 19.04 with an Intel i5-8500 3.00 GHZ GPU and 31GB RAM.

6.1 Experimental Setting

Datasets. The experiments are performed on the publicly available trajectory datasets, Porto [11] and GeoLife [46]. We select the trajectories with the length being at least 30. The selected Porto dataset contains 1.2 million trajectories, with 74.3M trajectory points and the longest trajectory consisting of 3881 location points. The GeoLife dataset retains 17,932 trajectories with the maximum length of 92,645, containing 24.8M trajectory points.

Compared Methods. We implemented the extended versions of alternative methods from the literature: Product Quantization [19], Residual Quantization [8], REST [44] and TrajStore[10]. We compare with one of the variations of REST, i.e., trajectory redundancy reduction, which was shown to perform best in their work [44]. We also test simple baselines as well as variants of PPQ-trajectory to quantify the improvements of each step.

Product Quantization and Residual Quantization are popularly used for approximate nearest neighbor (ANN) queries. However, they normally do not offer an effective index structure to support pruning for efficient querying. For fairness, we extended these methods with our indexing approach used in PPQ-trajectory.

For REST to work properly, the dataset needs to contain a highly repeating set of patterns between the trajectory set for building the reference set and the matched trajectory. This is not the case for most of the real-world data, including the datasets we use. To be able to compare with REST, we constructed another dataset. We first randomly selected 20,000 trajectories from Porto dataset, then for every trajectory we created four more similar trajectories by down-sampling and adding noise following the procedure in [23]. This process gives a dataset with 100,000 trajectories that is suitable for REST. Specifically, 2,000 trajectories are randomly selected for compression, we name this dataset as sub-Porto, while other trajectories are used to build a reference set for REST.

TrajStore builds an index with the spatial regions and recursively updates the spatial index by merging, splitting or appending. To align with the experimental setting, we implemented TrajStore to be able to get streaming trajectory points as input, and dynamically build the spatial index with time increasing.

We also implemented different variants of PPQ-trajectory to understand the effect of its building blocks. PPQ-S uses the spatial proximity for partitioning, PPQ-A uses the autocorrelations based similarities. PPQ-S-basic and PPQ-A-basic use the quantizers but not CQCs. As other baselines, we include comparisons with E-PQ, and also with a basic version of PPQ-trajectory, by skipping the prediction part, and name it as Q-trajectory.

Parameter Settings. The default quantization deviation threshold is $\varepsilon_1 = 0.001$, which is $\varepsilon_1^M \approx 111$ meters under the geographic coordinate system [6]. In the following experiments, we directly use ε_1^M to describe the comparative study. For the partition threshold ε_p, its setting varies on the autocorrelation and spatialproximity-based partitions. For the spatial proximity-based solution, ε_p defaults to 0.1 for Porto and 5 for Geolife. In the case of autocorrelation similarity, ε_p defaults to be 0.01 for both datasets. For TPI and PI, the grid cell size g_c is set to 100m.g_s defaults to 50m, which denotes the size of the grid cell for CQC. The threshold of the dropping rate of TRD, ε_c is defaulted to be 0.5. The default setting of the threshold of ADR, i.e., ε_d, is 0.5. ε_s defaults to 0.1, which represents the partition threshold for constructing index.

6.2 Query Performance

6.2.1 Spatio-temporal Range Query. The quality of the approximate results for STRQ is evaluated in terms of precision and recall. The precision is the ratio of the correctly retrieved trajectory IDs to the returned candidate list, and the recall is the ratio of the correctly retrieved trajectory IDs to all the trajectory IDs that match the query. We also measure the MAEs of the summaries over the datasets, i.e., the mean absolute errors between the reconstructed trajectory points and the original trajectory points.

For STRQ, we learn C independently for every timestamp guaranteeing the same number of codewords is given to trajectory points at the same time across all methods. We randomly select 10,000 queries. The comparative results are summarized in Table 2. For MAE, the PPQ-trajectory significantly performs better than other methods. As the time increases, PPQ-trajectory is able to gradually quantize a narrower range, whereas, residual and product quantization do not improve over time. The summary process of Trajstore cannot start until the spatial index has been updated with trajectory points of all the timestamps. To ensure fairness, the codewords are assigned in proportion to the number of trajectory points for every spatial cell of TrajStore.

With the same number of bits, PPQ-trajectory obtains significantly higher recall and precision values. For Geolife, autocorrelation-based partitioning (in PPQ-A and PPQ-A-basic) helps to achieve smaller MAEs compared to the spatial proximity based solution (PPQ-S and PPQ-S-basic), while PPQ-S-basic outperforms PPQ-A-basic on Porto dataset. We observe that autocorrelation similarity possesses some advantages upon capturing correlations and obtaining a narrower dynamic range of prediction errors. This result provides a useful insight also for other partitioning tasks and applications of spatio-temporal data, which is discussed in Section 3.2.1. PPQ-S and PPQ-A use CQC, which bounds the error within $\frac{\sqrt{2}}{2}g_s$ as shown in Lemma 3 and reduces the MAEs.

Using the local search strategy in Section 5.2, the precision and recall of PPQ-S and PPQ-A are all 1. TrajStore has smaller MAEs and achieves higher recall and precision compared to the other baselines, as the spatially close trajectory points are finely indexed by the same cell. However, TrajStore makes use of the pre-built spatial index, and the summarization is not efficiently generated with time evolving.

For the Geolife dataset, the MAEs of Q-trajectory, product quantization and residual quantization are extremely large, even around 20,000 meters that are unacceptable for the task of STRQ. Their corresponding precision and recall values are significantly lower than the alternatives. Hence, their precision and recall are marked with "×" in Table 2. The large spatial region spanning of Geolife leads to extremely large MAEs for Q-trajectory, residual quantization and product quantization.

6.2.2 Trajectory Path Query. TPQ involves querying timestamps and trajectory IDs of the STRQ results and reconstructing their next 10–50 trajectory points. According to Definition 5.3, the retrieved sub-trajectories of TPQ depends on the returned trajectory IDs of the STRQ. Different methods might retrieve sub-trajectories for different trajectory IDs, as observed in their different recall and precision values presented in Table 2. For fairness, we select 10,000 same trajectory IDs for all the methods to measure the MAEs of the retrieved sub-trajectories by comparing each to the corresponding original sub-trajectory.

The comparative results are summarized in Table 3. The MAEs for the sub-trajectories increase with the increasing TPQ lengths, because more spatial deviations are accumulated when querying longer sub-trajectories. The PPQ-trajectory and E-PQ significantly perform better than other methods, while the MAEs of E-PQ are smaller than that of PPQ-A-basic and PPQ-S-basic. The MAEs of Q-trajectory, residual quantization and product quantiztion increase significantly with the increasing query length, because their MAEs on the datasets have been extremely large (Table 2). We notice the MAEs of TrajStore over Porto get relatively large with l increasing while its MAE on the full dataset is smaller (Table 2). The codewords are assigned in proportion to the number of trajectory points for

Table 2: Quality of summaries and STRQ evaluation

Dataset	Porto			Geolife		
Performance Measure	MAE(m)	Precision	Recall	MAE(m)	Precision	Recall
PPQ-A	**18.35**	**1.000**	**1.000**	**4.85**	**1.000**	**1.000**
PPQ-A-basic	51.92	0.951	0.948	6.17	0.987	0.987
PPQ-S	23.30	**1.000**	**1.000**	7.89	**1.000**	**1.000**
PPQ-S-basic	44.41	0.944	0.939	14.72	0.976	0.976
E-PQ	76.60	0.931	0.926	15.06	0.962	0.961
Q-trajectory	1752.29	0.425	0.427	29105	x	x
Residual Quantization	868.96	0.675	0.675	22590	x	x
Product Quantization	641.34	0.736	0.725	21228	x	x
TrajStore	152.13	0.917	0.919	617.76	0.8535	0.8547

Table 3: MAE against different lengths of TPQ ($1.0e^3$m)

Dataset	Porto					Geolife				
TPQ length (l)	10	20	30	40	50	10	20	30	40	50
PPQ-A	**0.046**	**0.081**	**0.111**	**0.136**	**0.158**	**0.011**	**0.021**	**0.031**	**0.040**	**0.050**
PPQ-A-basic	0.357	0.657	0.935	1.194	1.437	0.073	0.139	0.205	0.271	0.337
PPQ-S	0.160	0.295	0.402	0.491	0.566	0.019	0.037	0.054	0.070	0.086
PPQ-S-basic	0.338	0.623	0.890	1.140	1.374	0.135	0.257	0.378	0.500	0.621
E-PQ	0.068	0.119	0.162	0.198	0.229	0.031	0.059	0.086	0.113	0.139
Q-Trajectory	24.90	44.55	62.02	77.54	91.75	190.2	360.5	530.0	697.5	861.1
Residual Quantization	3.684	6.641	9.252	11.59	13.66	150.7	288.0	423.7	558.6	692.6
Product Quantization	1.813	3.263	4.518	5.631	6.600	48.78	932.6	1377	1831	2289
TrajStore	5.665	10.32	14.66	18.56	22.04	7.703	14.93	22.15	29.22	36.17

Table 4: Average ratio of trajectories visited and MAE against different sizes of C ($1.0e^{-3}$ | m)

Dataset	Porto					Geolife				
	5bits	6bits	7bits	8bits	9bits	5bits	6bits	7bits	8bits	9bits
PPQ-A	**0.019**	**0.019**	**0.019**	**0.019**	**0.019**	**0.067**	**0.067**	**0.067**	**0.067**	**0.067**
	17.53	18.08	19.16	21.08	23.10	24.45	25.83	26.82	27.48	27.85
PPQ-A-basic	0.046	0.032	0.025	0.024	0.021	0.077	0.077	0.076	0.076	0.076
	62.03	41.46	26.81	17.64	18.55	31.27	12.94	6.180	3.000	1.610
PPQ-S	0.022	0.022	0.022	0.022	0.022	**0.067**	**0.067**	**0.067**	**0.067**	**0.067**
	19.52	19.72	19.86	19.38	19.76	18.93	14.99	6.650	2.970	1.620
PPQ-S-basic	0.039	0.033	0.026	0.023	0.020	0.077	0.077	0.077	0.077	0.077
	64.51	40.97	30.08	21.05	17.88	34.18	15.08	6.220	8.960	1.620
E-PQ	0.112	0.057	0.042	0.028	0.028	0.280	0.215	0.176	0.150	0.149
	118.3	73.00	45.58	29.44	19.16	46.04	30.71	23.98	22.99	22.12
Q-trajectory	0.675	0.488	0.372	0.320	0.293	51.83	23.38	8.741	3.254	1.297
	1008	671.8	438.6	278.4	173.0	6601	3110	1585	827.0	482.0
Residual Quantization	0.502	0.174	0.066	0.030	0.020	62.80	62.30	17.01	3.070	0.504
	639.0	329.0	160.0	74.21	33.29	22244	9373	3235	929.0	308.7
Product Quantization	5.655	5.587	5.286	4.994	4.627	26.70	23.62	18.81	11.44	5.830
	3693	3560	3430	3259	3024	79904	38094	16907	7186	3096

Table 5: Running time against different spatial deviation (s)

Dataset	Porto					Geolife				
spatial deviation (m)	200	400	600	800	1000	200	400	600	800	1000
PPQ-A	802.0	579.3	486.8	445.3	417.8	**706.0**	**557.0**	534.7	**477.5**	**345.6**
PPQ-A-basic	1617	1502	1405	1399	1316	1201	678.9	533.8	525.7	518.1
PPQ-S	**633.4**	**428.5**	**381.2**	**367.8**	**348.1**	799.5	591.1	**518.7**	504.0	445.5
PPQ-S-basic	1663	956.2	547.3	511.7	439.5	1244	1102	980.5	729.5	644.3
E-PQ	6543	3110	1775	1503	1157	647.4	597.9	550.8	497.6	459.0
Q-Trajectory	16027	7530	4347	3100	2789	10698	8387	6168	4692	4152
Residual Quantization	4765	2557	1727	1269	1094	13199	8489	6876	5538	4655
Product Quantization	4883	2870	2753	2667	2353	21113	8728	4181	3354	3073
TrajStore	12826	7448	5908	5355	4870	44588	33058	29261	27835	27063

every spatial cell of TrajStore (Section 6.2.1), hence, a larger spatial cell with a smaller number of trajectory points scattering in will be assigned a smaller number of codewords, then there will be larger deviations of summarizing the trajectory points of this spatial cell, even though the average deviations over all the spatial cells are smaller.

6.2.3 Filtering for Exact Match Queries. We now present the average ratios of trajectories visited when the summary is used as an index for exact match queries. After pruning irrelevant data, only a set of candidates is accessed. We randomly select 10,000 queries, and the average ratios of trajectories visited in their second step are presented. To ensure fairness, we learn C independently for every timestamp guaranteeing the same importance is given to trajectory points at different times which will not influence the filtering ratios for different timestamped queries. Table 4 compares the MAE and ratios of trajectories visited for alternative approaches, varying the size of C from five to nine bits. TrajStore summarizes trajectory points within each cell of the spatial index, while the spatial index is built with the trajectory points of all timestamps. Hence, for TrajStore, we cannot fairly summarize the trajectory points of every timestamp independently with the fixed size C. Hence, the comparison with TrajStore is not considered in this experiment.

PPQ-A performs the best under this performance measure. With the selected queries, it can directly access the trajectories mapped to the adjacent grid cells designed using ε_1 and g_s. The ratios of trajectories visited are the same with different sizes of C due to the accurate reconstructed representation. The same applies to PPQ-S. For other methods, we notice their ratios gradually decrease with the size of C increasing, because the accuracy that C can provide increases, which helps filter more candidate results. Similar performance is observed for Geolife, especially, we observe the same average ratios of trajectories visited for PPQ-S and PPQ-A. The corresponding MAE is also presented in Table 4. MAE decreases with the number of bits increasing for most of the methods. However, the MAEs of PPQ-A and PPQ-S do not strictly decline with the size of C increasing, because their spatial deviation is not fully decided by the quality of C, but also slightly influenced by CQCs.

6.3 Building Time Efficiency

6.3.1 Summary Efficiency. We evaluate the running times of generating the summary for different solutions with spatial deviations as 200m, 400m, 600m, 800m and 1000m. According to Lemma 3, the spatial deviation of PPQ-A and PPQ-S are $\frac{\sqrt{2}}{2}g_s$. In the experiment, we set $\varepsilon_1^M = 2g_s$ for PPQ-A and PPQ-S. For the other methods, the spatial deviation of their summary is simply determined by ε_1^M.

The building time is summarized in Table 5, which gradually decreases as the spatial deviation increases. This is because the quantization process finishes with fewer iterations when the error is larger. The running times of PPQ-trajectory are much smaller than those of Q-trajectory, residual quantization, product quantization, and TrajStore. In our solution, the dynamic range of the prediction errors that needs to be quantized is decreasing with time t evolving, hence its running time is smaller. PPQ-A and PPQ-S are more efficient than PPQ-A-basic and PPQ-S-basic, respectively on both datasets, because for the same spatial deviation, the setting of ε_1^M for PPQ-A and PPQ-S is larger than that of PPQ-A-basic and PPQ-S-basic, which needs fewer iterations to obtain the summary. For Porto dataset, the running time of our solution is up to 25 times faster than residual quantization, product quantization, Q-trajectory, and TrajStore, while E-PQ is faster than PPQ-A-basic when the spatial deviation is larger. The running time of E-PQ is comparatively high on Porto, which is even larger than residual quantization. It shows the E-PQ cannot work as efficiently as our solutions on the large datasets, because E-PQ executes one prediction on the whole datasets, the prediction errors will be larger and need

Table 6: Number of codewords in C against different spatial deviation ($\times 10^4$)

Dataset	Porto(m)					Geolife(m)				
	200	400	600	800	1000	200	400	600	800	1000
PPQ-A	**0.283**	0.172	0.113	**0.088**	**0.069**	**0.375**	**0.264**	**0.218**	**0.181**	**0.156**
PPQ-A-basic	0.651	0.371	0.279	0.227	0.183	0.560	0.422	0.349	0.324	0.288
PPQ-S	0.284	**0.148**	**0.112**	0.093	0.082	0.487	0.311	0.242	0.208	0.183
PPQ-S-basic	0.877	0.415	0.265	0.198	0.162	0.927	0.605	0.475	0.390	0.344
E-PQ	3.182	1.457	0.892	0.648	0.500	0.804	0.530	0.401	0.325	0.280
Q-trajectory	16.37	7.689	4.667	3.275	2.501	29.66	16.02	11.16	8.646	7.157
Residual Quantization	5.329	2.510	1.566	1.120	0.864	29.74	16.05	11.23	8.688	7.179
Product Quantization	5.175	2.444	1.527	1.092	0.845	29.24	15.72	10.95	8.453	6.982
TrajStore	7.617	3.620	2.287	1.589	1.173	35.64	18.75	12.71	9.539	7.724

Table 7: Statistics of TPI on different ε_c

	Index Size(MB)		Time Cost		No.Periods		No.Insertions	
ε_c	Porto	Geolife	Porto	Geolife	Porto	Geolife	Porto	Geolife
0.2	863.1	250.0	1346	7003	1245	14627	**4367**	**71448**
0.4	860.1	241.6	544	3792	656	10100	7207	89492
0.6	859.4	237.6	458	3028	485	7117	7198	95308
0.8	**859.1**	**237.3**	**418**	**2935**	**421**	**6876**	6637	101187

Table 8: Statistics of TPI on different ε_d

	Index Size(MB)		Time Cost		No.Periods		No.Insertions	
ε_d	Porto	Geolife	Porto	Geolife	Porto	Geolife	Porto	Geolife
0.2	862.0	249.2	1252	6535	1136	13958	**4457**	**55951**
0.4	860.0	238.2	497	4445	625	7953	5716	66400
0.6	859.9	236.5	480	3145	355	5670	6613	88033
0.8	**857.4**	**235.1**	**465**	**2848**	**245**	**3567**	7326	90554

more iterations to satisfy the spatial deviation requirement. For Geolife dataset, the running time of our solution is 4-78 times faster than residual quantization, product quantization, Q-trajectory, and TrajStore, while E-PQ is slightly faster than PPQ-trajectory for some spatial deviations. The running time of residual quantization, product quantization and Q-trajectory drops quickly with the spatial deviation increasing. However, their running time is extremely high when the spatial deviation is 200m, as the time span of Geolife is relatively large, which needs even more iterations to summarize.

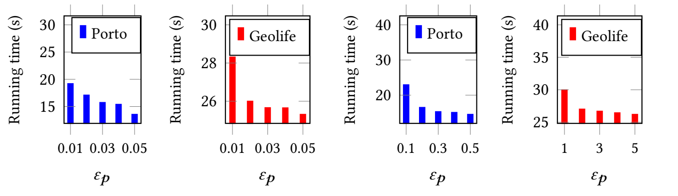
The running time of TrajStore is extremely high for all spatial deviations. Its running time includes both the time of building the spatial index and compression for every spatial cell, because TrajStore depends on the spatial index to conduct the summarization, and the process of building the index is time-consuming due to the frequent merging, splitting, and appending operations.

6.3.2 Dynamic Data Organization. In this section, we analyze the proposed partition-based index (PI) and temporal PI (TPI) on $\{T_i\}$ with different ε_c and ε_d, in terms of building time cost, the number of partitioned time periods,"Insertion" and "Re-build".

Table 7 reports that as ε_c increases, the index size gradually decreases, since a higher ε_c provides a higher tolerance of reusing the previous structure by performing "insertions". Similar results are observed for ε_d in Table 8, i.e., a higher ε_d allows a PI reused for more timestamps.

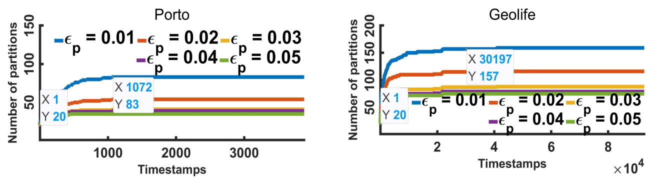
6.3.3 Temporal Partitioning Efficiency. In this section, we evaluate the efficiency of the incremental temporal partitioning (Section 3.2.2), and analyze how the number of partitions change with time, with respect to different ε_p values.

Figure 7 illustrates that the running time of the temporal partitioning component reduces as ε_p increases, since a smaller number

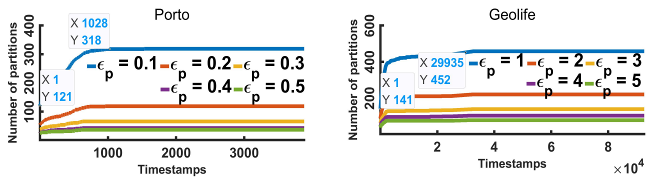


(a) PPQ-A (b) PPQ-S

Figure 7: Temporal partitioning running time against different ε_p



(a) PPQ-A (b) PPQ-A



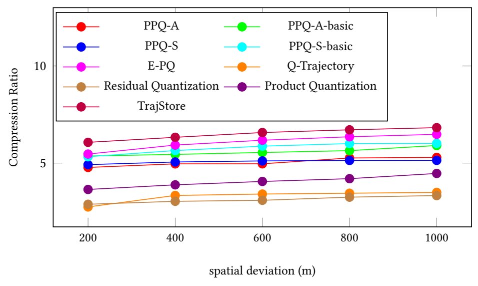
(c) PPQ-S (d) PPQ-S

Figure 8: Number of partitions q against different ε_p

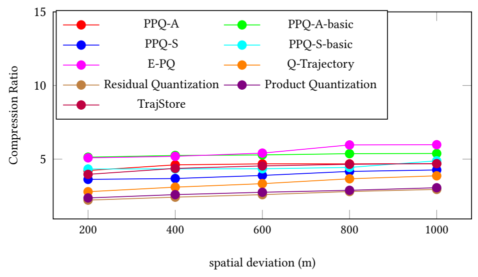
of partitions is produced when ε_p gets larger. In Figure 8, we also present the number of partitions that is maintained will gradually get stable with time increasing. For example, in Figure 8a, we get the maximum number of partitions, 83, on Porto dataset at $t = 1072$.

6.4 Compression Ratio

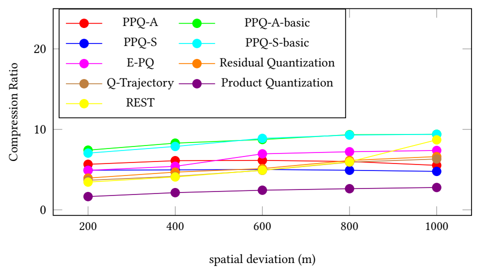
In this section, the compression ratios of different methods are measured for different values of spatial deviation, following the same parameters setting as Section 6.3.1. The comparative results are presented in Figure 9. For the Porto dataset, our solution outperforms Q-trajectory, residual quantization, and product quantization. The compression ratios of PPQ-A-basic and PPQ-S-basic slightly outperform PPQ-A and PPQ-S, respectively, because PPQ-A and PPQ-S need additional space to store CQC. The size of codebooks of E-PQ and TrajStore is up to 11 and 27 times larger, respectively, than PPQ-trajectory (Table 6), however, they achieve compression ratios that are higher than PPQ-trajectory, because we need additional space for multiple partitions, prediction coefficients $\{P_j[t]\}$ as well as CQC. Residual quantization and product quantization produce smaller sizes of codebooks compared to TrajStore, however, their compression ratios are 35%–51% smaller than that of TrajStore, because they need more space to store additional codeword indexes for restoring trajectory points from their summary. For Geolife dataset, PPQ-A-basic outperforms most of the alternatives in terms



(a) Porto



(b) Geolife



(c) sub-Porto

Figure 9: Compression ratio against different spatial deviation

Table 9: Disk-based index performance

	Index Size(MB)		No.I/Os		Response Time(s)		Building Time(s)	
Dataset	Porto	Geolife	Porto	Geolife	Porto	Geolife	Porto	Geolife
TPI	**857.4**	235.1	1225	2230	24	285	**465**	**2848**
PI	870.5	271.9	**338**	**301**	**18**	**121**	1572	32009
TrajStore	**857.4**	**233.5**	13803	35233	147	378	4244	24372

of the compression ratio, including Q-trajectory, residual quantization, product quantization and TrajStore. However, E-PQ produces compression ratios that are slightly higher than PPQ-trajectory when the spatial deviation is larger than 600m.

As mentioned in the experimental setting, REST have certain assumptions that do not hold for the general case we focus in this work. Hence, we only investigate the compression ratio on the sub-Porto dataset that is suitable for REST. The compression ratios for different spatial deviations are shown in Figure 9c. The comparative results with respect to different spatial deviations are presented in Figure 9c. When the spatial deviation is 200–600m, the compression ratios of PPQ-A-basic and PPQ-S-basic are two times that of REST. The gap decreases as the spatial deviation increases. REST's compression ratio depends on the correlation between the compressed trajectory and the reference set, the compressed trajectory cannot always be matched well with the offline learned reference set, which directly influences the compression ratio. However, PPQ-trajectory is able to flexibly extend codewords when the compressed trajectory can not be matched well with the existing codebook.

6.5 Further Comparison with TrajStore

In this section, we provide disk-based comparisons of temporal partition-based index (TPI) with TrajStore in terms of the index size, query response times, the number of I/Os during queries, and building times. The index of TrajStore is built on the raw trajectory points, for fairness, we align disk-based TPI with TrajStore to directly build index over the raw trajectory points in accordance with the end of Section 5.1. We followed the same process in the TrajStore[10], bounding the data on disk and setting the page size as 1MB. We randomly select 10,000 spatio-temporal queries and sort them in the order of their starting times. The parameters for TPI are ε_d=0.8 and ε_c=0.5.

These experimental results are presented in Table 9. The index size and building time of PI are larger than that of TPI and TrajStore for both datasets, while its response time and I/Os are the smallest among the three methods on both datasets. For Porto, the size of TPI is the same as TrajStore, while the index size of TrajStore for Geolife is slightly smaller than that of TPI. However, TPI continuously outperforms TrajStore in terms of the number of I/Os and query response times. Given a query, TPI can quickly target the relevant $period_j$ and filter out more irrelevant trajectory points in terms of time. However, for TrajStore, the quadtree-based index structure is shared by all timestamps, and a spatial cell can include trajectory points of a large time range, which might be stored on different pages. Hence, given a spatio-temporal query, for TrajStore several pages are likely to be visited, this is why the number of I/Os for TrajStore is over the number of queries we evaluate on.

Building of TPI is more efficient than TrajStore. TrajStore recursively updates the spatial index by merging, splitting or appending with trajectory points updated. However, when we get updates for TPI, it is only relevant to the trajectory points within a smaller range of timestamps, i.e., a $period_j$.

7 CONCLUSIONS

We presented PPQ-trajectory which generates and maintains an error-bounded summary for large-scale trajectory data analytics. A partition-wise predictive quantizer (PPQ) for spatio-temporal data is designed, which involves a spatial proximity and autocorrelation based partitioning, followed by a local coding. A temporally-quantized data organization is developed to process spatio-temporal queries efficiently. The query performances, building times, and compression capabilities of PPQ-trajectory significantly outperform other solutions in most of the experiments. As a future work, the quantization based approach can be enhanced to consider dynamic traffic conditions, and utilize machine learning to more accurately predict trajectory points and generate a more compact summary.

REFERENCES

[1] Fatih Altiparmak, Ertem Tuncel, and Hakan Ferhatosmanoglu. 2007. Incremental maintenance of online summaries over multiple streams. *IEEE Transactions on Knowledge and Data Engineering* 20, 2 (2007), 216–229.

[2] Richard E Bellman and Stuart E Dreyfus. 2015. *Applied dynamic programming.* Vol. 2050. Princeton university press.

[3] Yuhan Cai and Raymond Ng. 2004. Indexing spatio-temporal trajectories with Chebyshev polynomials. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data.* 599–610.

[4] Zhi Cai, Fujie Ren, Juncheng Chen, and Zhiming Ding. 2017. Vector-based trajectory storage and query for intelligent transport system. *IEEE Transactions on Intelligent Transportation Systems* 19, 5 (2017), 1508–1519.

[5] Addison Chan and Frederick WB Li. 2012. Utilizing massive spatiotemporal samples for efficient and accurate trajectory prediction. *IEEE Transactions on Mobile Computing* 12, 12 (2012), 2346–2359.

[6] Kang-Tsung Chang. 2008. *Introduction to geographic information systems.* Vol. 4. McGraw-Hill Boston.

[7] Minjie Chen, Mantao Xu, and Pasi Franti. 2012. Compression of GPS trajectories. In *2012 Data Compression Conference.* IEEE, 62–71.

[8] Yongjian Chen, Tao Guan, and Cheng Wang. 2010. Approximate nearest neighbor search by residual vector quantization. *Sensors* 10, 12 (2010), 11259–11273.

[9] Jonathan D Cryer and Natalie Kellet. 1991. *Time series analysis.* Springer.

[10] Philippe Cudre-Mauroux, Eugene Wu, and Samuel Madden. 2010. Trajstore: An adaptive storage system for very large trajectory data sets. In *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010).* IEEE, 109–120.

[11] ECML-PKDD. 2015. *Taxi Service Trajectory Prediction Challenge 2015.* http://www.geolink.pt/ecmlpkdd2015-challenge/

[12] Hakan Ferhatosmanoglu, Ertem Tuncel, Divyakant Agrawal, and Amr El Abbadi. 2000. Vector approximation based indexing for non-uniform high dimensional data sets. In *Proceedings of the ninth international conference on Information and knowledge management.* 202–209.

[13] Alyson K Fletcher, Sundeep Rangan, Vivek K Goyal, and Kannan Ramchandran. 2007. Robust predictive quantization: Analysis and design via convex optimization. *IEEE Journal of selected topics in signal processing* 1, 4 (2007), 618–632.

[14] Stefan Funke, Tobias Rupp, André Nusser, and Sabine Storandt. 2019. PATHFINDER: storage and indexing of massive trajectory sets. In *Proceedings of the 16th International Symposium on Spatial and Temporal Databases.* 90–99.

[15] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. 2013. Optimized product quantization for approximate nearest neighbor search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2946–2953.

[16] Allen Gersho and Robert M Gray. 2012. *Vector quantization and signal compression.* Vol. 159. Springer Science & Business Media.

[17] Kevin Gourley and Douglas Green. 1983. A polygon-to-rectangle conversion algorithm. *IEEE Computer Graphics and Applications* 1 (1983), 31–36.

[18] Yunheng Han, Weiwei Sun, and Baihua Zheng. 2017. COMPRESS: A comprehensive framework of trajectory compression in road networks. *ACM Transactions on Database Systems (TODS)* 42, 2 (2017), 11.

[19] Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2010. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence* 33, 1 (2010), 117–128.

[20] Georgios Kellaris, Nikos Pelekis, and Yannis Theodoridis. 2013. Map-matched trajectory compression. *Journal of Systems and Software* 86, 6 (2013), 1566–1579.

[21] Satoshi Koide, Yukihiro Tadokoro, Takayoshi Yoshimura, Chuan Xiao, and Yoshiharu Ishikawa. 2018. Enhanced Indexing and Querying of Trajectories in Road Networks via String Algorithms. *ACM Transactions on Spatial Algorithms and Systems (TSAS)* 4, 1 (2018), 3.

[22] Daniel Lemire and Leonid Boytsov. 2015. Decoding billions of integers per second through vectorization. *Software: Practice and Experience* 45, 1 (2015), 1–29.

[23] Xiucheng Li, Kaiqi Zhao, Gao Cong, Christian S Jensen, and Wei Wei. 2018. Deep representation learning for trajectory similarity computation. In *2018 IEEE 34th International Conference on Data Engineering (ICDE).* IEEE, 617–628.

[24] Zhisheng Li, Ken CK Lee, Baihua Zheng, Wang-Chien Lee, Dik Lee, and Xufa Wang. 2010. Ir-tree: An efficient index for geographic document search. *IEEE Transactions on Knowledge and Data Engineering* 23, 4 (2010), 585–599.

[25] Jiajun Liu, Kun Zhao, Philipp Sommer, Shuo Shang, Brano Kusy, and Raja Jurdak. 2015. Bounded quadrant system: Error-bounded trajectory compression on the go. In *2015 IEEE 31st International Conference on Data Engineering.* IEEE, 987–998.

[26] Jiajun Liu, Kun Zhao, Philipp Sommer, Shuo Shang, Brano Kusy, Jae-Gil Lee, and Raja Jurdak. 2016. A novel framework for online amnesic trajectory compression in resource-constrained environments. *IEEE Transactions on Knowledge and Data Engineering* 28, 11 (2016), 2827–2841.

[27] Xiaoyan Liu and Hakan Ferhatosmanoglu. 2003. Efficient k-NN search on streaming data series. In *International Symposium on Spatial and Temporal Databases.* Springer, 83–101.

[28] Stuart Lloyd. 1982. Least squares quantization in PCM. *IEEE transactions on information theory* 28, 2 (1982), 129–137.

[29] Chengjiao Lv, Feng Chen, Yongzhi Xu, Junping Song, and Pin Lv. 2015. A trajectory compression algorithm based on non-uniform quantization. In *2015 12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD).* IEEE, 2469–2474.

[30] Jonathan Muckell, Jeong-Hyon Hwang, Vikram Patil, Catherine T Lawson, Fan Ping, and SS Ravi. 2011. SQUISH: an online approach for GPS trajectory compression. In *Proceedings of the 2nd International Conference on Computing for Geospatial Research & Applications.* 1–8.

[31] Jonathan Muckell, Paul W Olsen, Jeong-Hyon Hwang, Catherine T Lawson, and SS Ravi. 2014. Compression of trajectory data: a comprehensive evaluation and new approach. *GeoInformatica* 18, 3 (2014), 435–460.

[32] Zhaolong Ning, Jun Huang, and Xiaojie Wang. 2019. Vehicular fog computing: Enabling real-time traffic management for smart cities. *IEEE Wireless Communications* 26, 1 (2019), 87–93.

[33] Mohammad Norouzi and David J Fleet. 2013. Cartesian k-means. In *Proceedings of the IEEE Conference on computer Vision and Pattern Recognition.* 3017–3024.

[34] Athanasios Papoulis and S Unnikrishna Pillai. 2002. *Probability, random variables, and stochastic processes.* Tata McGraw-Hill Education.

[35] Jignesh M Patel, Yun Chen, and V Prasad Chakka. 2004. STRIPES: an efficient index for predicted trajectories. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data.* 635–646.

[36] Iulian Sandu Popa, Karine Zeitouni, Vincent Oria, and Ahmed Kharrat. 2015. Spatio-temporal compression of trajectories in road networks. *GeoInformatica* 19, 1 (2015), 117–145.

[37] Hanan Samet. 1984. The quadtree and related hierarchical data structures. *ACM Computing Surveys (CSUR)* 16, 2 (1984), 187–260.

[38] Renchu Song, Weiwei Sun, Baihua Zheng, and Yu Zheng. 2014. PRESS: A novel framework of trajectory compression in road networks. *Proceedings of the VLDB Endowment* 7, 9 (2014), 661–672.

[39] Waldo R Tobler. 1979. Cellular geography. In *Philosophy in geography.* Springer, 379–386.

[40] Ertem Tuncel, Hakan Ferhatosmanoglu, and Kenneth Rose. 2002. VQ-index: An index structure for similarity searching in multimedia databases. In *Proceedings of the tenth ACM international conference on Multimedia.* ACM, 543–552.

[41] Sheng Wang, Zhifeng Bao, J Shane Culpepper, Timos Sellis, Mark Sanderson, and Xiaolin Qin. 2017. Answering top-k exemplar trajectory queries. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE).* IEEE, 597–608.

[42] Sheng Wang, Zhifeng Bao, J Shane Culpepper, Zizhe Xie, Qizhi Liu, and Xiaolin Qin. 2018. Torch: A Search Engine for Trajectory Data.. In *SIGIR.* 535–544.

[43] Roger Weber, Hans-Jörg Schek, and Stephen Blott. 1998. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *VLDB*, Vol. 98. 194–205.

[44] Yan Zhao, Shuo Shang, Yu Wang, Bolong Zheng, Quoc Viet Hung Nguyen, and Kai Zheng. 2018. Rest: A reference-based framework for spatio-temporal trajectory compression. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining.* ACM, 2797–2806.

[45] Kai Zheng, Yan Zhao, Defu Lian, Bolong Zheng, Guanfeng Liu, and Xiaofang Zhou. 2019. Reference-based framework for spatio-temporal trajectory compression and query processing. *IEEE Transactions on Knowledge and Data Engineering* (2019).

[46] Yu Zheng, Lizhu Zhang, Xing Xie, and Wei-Ying Ma. 2009. Mining interesting locations and travel sequences from GPS trajectories. In *Proceedings of the 18th international conference on World wide web.* ACM, 791–800.