A Spatiotemporal Trajectory Data Index Based on the Hilbert Curve Code

YuHao Wu 1, Xuefeng Cao 1,*, Zipeng An

1. Institute of Surveying and Mapping, Information Engineering University, Zhengzhou 450001, China;
* Correspondence: 514536575@qq.com;

Abstract: Massive trajectory data have been accumulated with the rapid development of global positioning technology and the popularisation of intelligent mobile terminal. However, the generation of massive data does not necessarily lead to the increase in effective data. An index method meeting the efficient management requirements of spatiotemporal trajectory data needs to be designed for the efficient spatiotemporal analysis and calculation of data. This paper proposes a spatiotemporal index method based on the Hilbert curve code to solve this problem. Firstly, the method constructs a multi-scale spatiotemporal grid model covering the whole world by dividing the three-dimensional space composed of time, latitude and longitude. Secondly, the grid cell codes are designed based on the Hilbert curve for hierarchical organisation of the trajectory data. Finally, the corresponding query process based on the code index is proposed in accordance with the different spatiotemporal query requirements of trajectory data. The comparison experiments show that the proposed method is more efficient than the existing spatiotemporal index method and can effectively support the management of massive multi-scale trajectory data.

Keywords: trajectory data; spatiotemporal index; Hilbert curve; spatiotemporal query; grid

1. Introduction

In recent years, with the rapid development and use of position sensing equipment, such as global positioning system (GPS) and Beidou, the collection, storage, analysis and excavation of the spatiotemporal trajectory data of moving objects have become a reality. Spatiotemporal trajectory data comprise a series of spatiotemporal trajectory sampling points, in which each sampling point records in detail the position, speed, direction, current moment and other relevant properties of the current moving object. This information provides data support for in-depth exploration of the spatiotemporal evolution process of moving objects and provide referential and utilised knowledge for location-based service (LBS), crowd management, urban planning and other relevant application fields [1–4].

As typical spatiotemporal data, trajectory data are characterised by large data size, fast update speed, continuous monotonicity in time and spatial continuity in space. The massive growth of trajectory data does not necessarily represent an inevitable increase in the amount of available data. The massive data of scattered organisations may contradict the efficient analysis and application of data. How to effectively organise and analyse trajectory data is an urgent problem to be solved. The common solution is to design and construct a suitable spatiotemporal index. The spatiotemporal index will need to meet the following three core requirements to solve the above problems [5]: (1) the equivalent index of time and space. The existing spatiotemporal indexes mostly treat time and space separately. When performing spatiotemporal data query, the index order of time-first space or space-first time is often adopted, resulting in high complexity of spatiotemporal query. Constructing time and space equivalent indexes can reduce the complexity of spatiotemporal query and improve the efficiency of data query. (2) Index update costs. The location of the moving object changes relatively frequently, resulting in a high frequency of index updates. Thus, updates in the index structure should be cost-effective. (3) Support for various types of spatiotemporal query operations. The spatiotemporal index should be suitable to many
types of spatiotemporal query operations to achieve the complex analysis of spatiotemporal trajectory data.

We propose a spatiotemporal trajectory data indexing method based on the Hilbert curve coding to solve the above problems. Firstly, this method extends Geohash's spatial splitting idea and constructs a global multi-scale spatiotemporal mesh model by splitting the three-dimensional (3D) space composed of time and latitude and longitude. Then, a spatiotemporal coding index method based on Hilbert curve is designed for the hierarchical nested subdivision and multi-scale spatiotemporal grid. Finally, for the different query requirements, the corresponding query steps are put forward in accordance with the proposed encoding index method.

2. Related work

2.1. Spatiotemporal index method

The existing spatiotemporal index technologies supporting trajectory data can be mainly divided into three categories:

(1) Multi-dimensional index method containing augmented temporal dimension: The basic idea of multi-dimensional index is to add time dimension to traditional spatial indexes (such as R-tree index). Common representative methods include the 3DR-tree [6], TB-tree [7], STR-tree [8], etc. The advantage of this kind of index is that the index structure can be adaptively adjusted in accordance with the data distribution. One of the disadvantages is the use of a tree index, which introduces excessively redundant space. Another disadvantage is that the build efficiency of the indexes can easily deteriorate [9].

(2) Multi-version structure index method: This index first constructs a spatial index, such as R-tree, on different timestamps and then organises the spatial indexes of each timestamp by using a 1D index such as B-tree. The purpose is to save unchanging data in each space index to save storage space. Common methods include the MR-tree [10], HR-tree [11] and MV3R-tree [12]. When classifying trajectory data, the index first considers time, followed by space. Therefore, the multi-version structure index features the advantage of high time query efficiency [13]. However, each spatial index has to be updated constantly given the frequent changes in the trajectory data, translating into the overall high maintenance cost of the indexes.

(3) Index method based on spatial partitioning: The spatial partitioning index classifies trajectory data into a pre-divided grid and then constructs a time index for each grid. Common methods include SETI [14], MTSB tree [15], CSE, GeoMesa [16] and so on. The core of this type of index is the dimensionality reduction of 2D geospatial to 1D coding. This kind of index prioritises space after time. Given that this type of index adopts fixed geographical grids divided by rules, it possesses the advantages of high construction efficiency, low update cost and high efficiency in large-scale spatiotemporal data query [17–18]. The disadvantage is that the time and space range need to be preset.

2.2. Geohash

In the spatial partitioning index, Geohash is a widely used global spatial meshing coding method proposed by Gustavo Niemeyer in 2013 and published on the Wikipedia page [19].

Geohash is a geocoding format that alternates the earth’s surface along the longitude and latitude and uses a binary number (Geohash code) to represent the non-overlapping meshes formed. After dividing the world twice, four Geohash codes (00, 01, 10 and 11) with a code length of 2 are formed, corresponding to four grids. Considering the mesh coded as 11 as an example, the longitude range is [0°,180°], and the latitude range is [0°,90°]. If this grid is further divided twice, four Geohash codes with a code length of 4 will be formed (1100, 1101, 1110 and 1111).

Geohash coding presents several features [20]:

- Uniqueness. Each cell grid has a corresponding globally unique code.
• One-dimensionality. Geohash uses a 1D number or string to represent a 2D space region.
• Recursive. The Geohash subordinate unit mesh is divided by the upper unit mesh, which is essentially a spatial Z-curve fill. The four Geohash meshes in a ‘Z’ shape belong to the same parent mesh, and their binary codes have the same prefix.

Geohash effectively converts 2D longitude and latitude information into 1D strings, which can be used as spatial information identifiers in relational databases. Geohash has been shown to perform well in many spatial data indexing problems [21]. However, given the lack of time information in Geohash code, additional indexes must be built for time dimension in the management of spatiotemporal data to increase the operational complexity of spatiotemporal data and reduce the operational efficiency.

3. Materials and methods

3.1. Spatiotemporal grid model

The core idea of Geohash is to build a global longitude and latitude grid model and design unique codes for all grid cells, achieving dimensionality reduction from 2D to 1D coding. Given the lack of time information, the Geohash code cannot be directly used for the management of trajectory data. This paper extends the time dimension on the basis of Geohash to solve this problem. Firstly, we construct a spatiotemporal mesh model by dividing the 3D space composed of time and latitude and longitude. Subsequently, all grids are coded using the Hilbert curve to achieve the dimensionality reduction from 3D spatiotemporal to 1D coding.

Considering the convenience of data processing in time dimension, the time unit obtained by division should be an integer unit as much as possible. Therefore, using the method in [22] to transform the time unit obtained by the splitting into integer units, this paper expands 1 year to 16 months, 1 month to 32 days, 1 day to 32 h, 1 h to 64 min and 1 min to 64 s. Thus, the time unit obtained after the division is still an integer unit, which is beneficial to the improvement of data processing.

Different from the monotonically increasing time dimension, the latitude and longitude in real space are divided into west longitude, east longitude, south latitude and north latitude. In mathematical calculations, positive and negative signs must be added to distinguish values that are not conducive to the time dimension integrated. The latitude and longitude in real space are extended to maintain the monotonously increasing nature of the spatiotemporal range: define latitude \( \lambda \in (0°, 180°) \), that is, add 90° to the original geographic latitude value; define the longitude \( \varphi \in (0°, 360°) \), that is, add the original longitude value to 180°.

By expanding the spatiotemporal dimension, this paper sets the spatiotemporal range to \( U = \{ \varphi \in (0°, 360°), \lambda \in (0°, 180°), t \in (0\text{mon}, 16\text{mon}) \} \) and the octal tree partitioning of 3D space composed of time and space (Figure 1). The hierarchical process is as follows:

1. The initial spatiotemporal grid of the 0th layer includes the ranges of longitude \( \varphi \in (0°, 360°) \), latitude \( \lambda \in (0°, 180°) \) and time \( t \in (0 \text{months}, 16 \text{months}) \).
2. The first layer mesh is obtained by the octree segmentation of the 0th layer mesh, that is, the three dimensions are separately split into half. Finally, the resolution of the eight subgrids is \( (180°, 90°, 8 \text{mons}) \).
3. The second layer mesh is obtained by recursive octree decomposition of the first layer of the mesh, and the final resolution of 64 subgrids is \( (90°, 45°, 4 \text{mons}) \). Given the expansion in the time dimension, the 64 grids in the hierarchy contain 16 virtual grids that also have no realistic meaning in the actual data production; the data falling into the virtual grid do not appear. At the rate of the grid split, the virtual grid is no longer a recursively split.
4. Step 3 is repeated to obtain layers 3–26.
3.2. Hilbert curve code

The space-filling curve is a 1D curve that can cover a specified area by recursion [23]. The commonly used space-filling curves include the Hilbert, Peano, Sierpinski and Morton (Z) curves. The basic idea of using a space-filling curve to create a split code to divide the data space into a number of grid cells in accordance with a certain rule. Each grid cell contains several data points, and the space-filling curve passes through all the grid cells once. Each grid unit is assigned a unique number based on the order in which the space-filling curve passes through the grid cells, and these numbers represent the codes of the grid cells on the curve [24].

The space-filling curve code primarily aims to maintain spatiotemporal data and the spatiotemporal proximity of the code. That is, the data with similar time and space are similar in coding, and vice versa. Various operations, such as querying and retrieving spatiotemporal data, usually involve a large amount of data, and good proximity allows data distribution over a small number of curve segments. As the number of curve segments is considerably less than the number of data, the data access intensity is substantially reduced, which shows a significant effect on the improvement of data operation performance. To obtain the best clustering effect, several scholars have verified the Hilbert curve in a large number of experiments [25–26]. Therefore, the Hilbert curve is used to fill the spatiotemporal grid cells, and the coupled 1D spatiotemporal index is established to improve the retrieval efficiency of spatiotemporal data.

The Hilbert curve belongs to the Peano curve family, which was first proposed by the Italian mathematician Peano [27]. This curve features the characteristics of self-similarity and recursion. The high-level curve can be constructed recursively from the low-level curve. The 3D m-order Hilbert curve is denoted by $H_m^3$, and the 3D one-order Hilbert curve $H_1^3$ is used as an example. The process of recursively generating the 3D two-order Hilbert curve $H_2^3$ can be composed of a series of coordinate transformation instructions. The coordinate transformation consists of two forms, namely, the exchange ‘↔’ and the reversal ‘′’. Figure 2(b) shows the result of exchanging $t$ and $\phi$, whereas Figure 2(c) displays the result of $\lambda$ reversal.

![Figure 2](image-url)
The result obtained by passing $H^3_1$ through the coordinate transformation $G[i]$ is denoted as $H^3_1[i]$, and $H^3_2$ is composed of eight $H^3_1[i]$ connected end to end. Table 1 shows the coordinate transformation $G[i]$ formed at each position; $G \in \{G[0], G[1], G[2], \ldots, G[7]\}$ is called a gene list of the Hilbert curve [28].

| Coordinate transformation | $G[0]$ | $G[1]$ | $G[2]$ | $G[3]$ | $G[4]$ | $G[5]$ | $G[6]$ | $G[7]$ |
|---------------------------|--------|--------|--------|--------|--------|--------|--------|--------|
| Exchange                  | $t \leftrightarrow \phi$ | $\lambda \leftrightarrow \phi$ | -      | $t \leftrightarrow \phi$ | $t \leftrightarrow \phi$ | -      | $\lambda \leftrightarrow \phi$ | $t \leftrightarrow \phi$ |
| Reversal                  | -      | -      | -      | $t', \phi'$ | -      | -      | $\lambda', \phi'$ | $t', \phi'$ |

According to Hilbert’s recursiveness and self-similarity, $H^3_2$ can be obtained by transforming each node in $H^3_2$ in accordance with the gene list. Thus, the Hilbert curve of any level can also be obtained. The $H^3_2$ can traverse eight nodes in the computational domain, $H^3_2$ can traverse 64 nodes in the computational domain, $H^m_3$ can traverse $8^m$ nodes in the computational domain. The Hilbert curve is structurally compatible with the spatiotemporal splitting octree. The Hilbert curve of different levels can be used to traverse all the spatiotemporal grid elements obtained by the octree branching, and the spatiotemporal grid elements of each level are encoded in accordance with the curve filling order.

Eight nodes are present on the $H^3_2$ curve, and each node can be assigned an identification code in the form of $[e][8]$. (where $[i][j]$ represents the level, and $[i][j]$ denotes an octal number). The $H^3_2$ Hilbert curve is obtained by subdividing the node grid space of the $H^3_2$ Hilbert curve. The identifier of the node is an identification code $[h^3_2][8]$, which consists of two parts. The first part is the one-order binary identification code of the node $[e][8]$, and the second part is its subdivision code $[e'][8]$, i.e. $[h^3_2][8] = [e][8] \oplus [e'][8]$ ( means shifting one bit to the left and adding), as shown in Figure 2(d).

Combined with the recursiveness, continuity and monotonicity of the Hilbert curve, the $8^m$ nodes on the $H^m_3$ curve can be represented by an m-bit octal identification code $[h^m_3][8] = [e][8] \oplus [e][8] \oplus \ldots \oplus [e][8]$. For a given spatiotemporal grid, the subdivision code that belongs to each layer in the Hilbert curve can be solved first, and the subdivision codes are then spliced to obtain the final Hilbert curve code. The subdivision code of each level is related to the subspace in which the spatial coordinates $(i,j,k)$ fall in each hierarchical process. $r_\phi \in (0,1), r_\lambda \in (0,1), r_t \in (0,1)$ are used to identify the subspace to which the spatial coordinates $(i,j,k)$ belong; 0 and 1 represent the left and right subsegments of a certain dimension, respectively. Table 2 shows the mapping relationship between $(r_\phi, r_\lambda, r_t)$ and binary encoding. The mapping relationship $f$ is actually the order of the nodes on the $H^3_2$ curve.
The Hilbert curve binary coding algorithm corresponding to the m-layer spatiotemporal grid element \((i,j,k)\) to which the spatiotemporal coordinate points \((\varphi, \lambda, t)\) belong is given as follows:

**Algorithm 1: Hilbert coding algorithm–Hilbert Index\((i,j,k)\)**

Description: Calculate the \(H^3_m\) code \([h^3_m][8]\) of spatiotemporal coordinate points \((\varphi, \lambda, t)\)

Input: Point \(p(\varphi, \lambda, t) = \{(\varphi, \lambda, t) \in (0^\circ, 360^\circ), \lambda \in (0^\circ, 180^\circ), t \in (0, 1\text{ year})\}\)

Output: \([h^3_m][8]\)

1. \((\varphi, \lambda, t) \rightarrow (i,j,k)
2. for \(r = 1\) to \(m\)
3. \(r_\varphi = (i \& 2^{m-1}) \geq 0; r_\lambda = (j \& 2^{m-1}) \geq 0; r_t = (k \& 2^{m-1}) \geq 0;
4. \([\varepsilon][8] = f(r_\varphi, r_\lambda, r_t)
5. \([h^3_m][8] = \oplus = [\varepsilon][8]
6. \varphi, \lambda, t \leftarrow G[i]
7. end for
8. output \([h^3_m][8]\)

### 3.3. Code index framework

The proposed code has three characteristics: (1) Global space–time uniqueness. The grid cells in the space–time grid model contain one and only one code corresponding to them. (2) Coding recursion. The code of different levels of the same space–time grid unit has the same prefix. The longer the code length, the smaller the space–time range and the higher the precision are. (3) Coding one-dimensionality. Space and time are represented by a 1D code, and the dimensionality reduction processing is adopted to improve the query speed.

The spatiotemporal trajectory data are transformed into a Hilbert curve code, and the generated code mapping relationship is stored in a relational database to construct the index framework shown in **Figure 3**. The code index framework of this paper is mainly divided into two modules: user query and storage management. The user converts the query requirements to the code query statement through the user query module. The storage management module is mainly used to store spatiotemporal trajectory data sets and generate the Hilbert-coded indexes.

Hilbert-coded indexes are generated for each spatiotemporal trajectory data imported into the storage management module. Each trajectory data may be determined in accordance with the space–time unit in which it falls, and multiple spatiotemporal trajectory data points may be present in each space–time unit. The item of the code index is the ID of each spatiotemporal trajectory data, and one code corresponds to multiple IDs. When new data are added, the correspondence between the code and the ID must be recalculated. Then, the data are inputted into the database. When a record needs to be deleted, the corresponding code is first calculated, and the ID of the record is deleted in the ID set corresponding to the code. Then, the record is deleted in the database. Given that time and space are divided at the same time in the construction of the space–time grid model, the code index considers both time and space in the query process, and the query complexity is lower than that of the index method considering the time-first space or space-first time. The specific query steps are described in Chapter 3.

### Table 2. Coded mapping relationship \(f\).

| Subdivision | \((0,0,0)\) | \((0,0,1)\) | \((1,0,1)\) | \((1,0,0)\) | \((1,1,0)\) | \((1,1,1)\) | \((0,1,1)\) | \((0,1,0)\) |
|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|
| code        | 0           | 1           | 2           | 3           | 4           | 5           | 6           | 7           |

The Hilbert curve binary coding algorithm corresponding to the m-layer spatiotemporal grid element \((i,j,k)\) to which the spatiotemporal coordinate points \((\varphi, \lambda, t)\) belong is given as follows:
4. Spatiotemporal data query

The data queries involved in the operation of trajectory data can be roughly divided into spatiotemporal data point query, range query and K-nearest neighbour (KNN) query. The spatiotemporal data query can be converted into the query of the code through the corresponding steps. With the one-dimensionality and spatiotemporal proximity of the code, the number of the data IO can be reduced, thus improving the efficiency of the data query.

4.1. Spatiotemporal point query

The spatiotemporal point query uses the precise latitude, longitude and time information as the input. The purpose is to query the data in the database with the same latitude, longitude and time information. Given that each data record in the database corresponds to one code, the query range can be reduced to several code ranges, and the number of full table traversal is reduced to optimise the query efficiency.

The specific steps of the space–time point query are as follows:

1. The set of spatiotemporal trajectory points \( P = \{p_i(\text{lon}_i, \text{lat}_i, \text{time}_i)|1 \leq i \leq n\} \) to be queried is entered.

2. Coarse screening. A corresponding code \( \text{code}_i \) is generated for all points \( p_i \) in the spatiotemporal trajectory point set \( P \) to form a code set \( \text{CodeSet} = \{\text{code}_i|1 \leq i \leq n\} \). All the records belonging to \( \text{CodeSet} \) from the database are filtered, and the initial screening set \( P' = \{p_j(\text{lon}_j, \text{lat}_j, \text{time}_j)|1 \leq j \leq n \leq m\} \) is formed.

3. Fine screening. For point \( p_j \) in the initial screening set \( P' \), whether or not \( p_j = p_i \) exists is assessed. If it exists, \( p_j \) is added to the result point set \( \text{PointQuerySet} = \{p_r(\text{lon}_r, \text{lat}_r, \text{time}_r)|1 \leq r \leq n\} \).

4. Output result point set as \( \text{PointQuerySet} = \{p_r(\text{lon}_r, \text{lat}_r, \text{time}_r)|1 \leq r \leq n\} \).

4.2. Spatiotemporal range query

The spatiotemporal query considers the spatial query area and time interval as the input. The goal is to find all the data in the specified query range. For example, if the input query condition is \((\text{minlon}, \text{minlat}, \text{mintime}, \text{maxlon}, \text{maxlat}, \text{maxtime})\), then all the points \( P(\text{lon}, \text{lat}, \text{time}) \) that meet the following conditions are obtained:
Similar to the space–time point query method, the process of using the code index for spatiotemporal range query is mainly divided into two stages. The first stage is the coarse screening, in which the code set corresponding to the input query range is filtered out. The second stage is the fine screening, in which all the data corresponding to the specified query range are finely filtered. In the 3D space–time domain, the spatiotemporal range query can be expressed as querying all the data inside a cuboid, as shown in Figure 4, where the coordinates of the lower left corner of the cuboid are $(P\_{\text{min}}, \text{minlon}, \text{minlat}, \text{mintime})$, and the coordinates of the upper right corner are $(P\_{\text{max}}, \text{maxlon}, \text{maxlat}, \text{maxtime})$. During the query process, as shown in Figure 5, on the basis of the relation between the grid cell and the query range cuboid, the grid cell can be divided into two types: the grid cell completely contained by the query range cuboid and the grid cell with parts that are not intersected with the query range cuboid. For the grid cells that are completely contained in the query range cuboid, according to the geometric relationship, all the data in the grid cell are in the query range cuboid. For a grid cell with parts which are not intersected with the query range cuboid, given the impossibility of determining whether all the contained data are within the query range cuboid, a fine screening step is required.

![Figure 4. Spatiotemporal range query.](image)

![Figure 5. Grid cell classification.](image)

The specific steps of the spatiotemporal range query are as follows:

1. Determination of the grid code level used. Before performing a spatiotemporal range query, the spatiotemporal code level that is suitable to the range is determined. If the level is notably high, the code range is extremely large, which reduces the query efficiency. If the level is evidently low, the code index cannot filter out high amounts of data that are not in the query range. Therefore, the algorithm for setting the code level is as follows:

   1. The span of the query range in longitude, latitude and time is calculated:
\[
\Delta \text{lon} = \text{maxlon} - \text{minlon} \\
\Delta \text{lat} = \text{maxlat} - \text{minlat} \\
\Delta \text{time} = \text{maxtime} - \text{mintime}
\]

(2) The level corresponding to longitude, latitude and time calculated:

\[
\begin{align*}
m_{\text{lon}} &= \left\lceil \frac{\log(N_e/\Delta \text{lon})}{\log(2)} \right\rceil \\
m_{\text{lat}} &= \left\lceil \frac{\log(N_j/\Delta \text{lat})}{\log(2)} \right\rceil \\
m_{\text{time}} &= \left\lceil \frac{\log(N_{\text{time}}/\Delta \text{time})}{\log(2)} \right\rceil
\end{align*}
\]

(3) The code level is set to be higher than the minimum level in the 3D levels by one level. Thus, the code range to be queried is not extremely large when more grid cells are completely contained in the query range cuboid:

\[
m = \max(m_{\text{lon}}, m_{\text{lat}}, m_{\text{time}}) + 1
\]

(2) Coarse screening. The grid cell codes that intersect with the query range cuboid are filtered out to form the code set \(\text{CodeSet} = \{\text{code}_1, \text{code}_2, \text{code}_3, \ldots\}\). The code in the set \(\text{CodeSet}\) is classified according to whether it is completely contained in the rectangular range of the query range. The grid cell codes that are completely included in the query range cuboid constitute the set \(\text{CodeSet}_{\text{contains}}\), and the grid cell codes which partially intersect the query range cuboid constitute the set \(\text{CodeSet}_{\text{notcontains}}\).

(3) Fine screening. The data in the code set \(\text{CodeSet}_{\text{contains}}\) are directly added to the result set \(\text{RangeQuerySet}\) without fine filtering. For the data in the code set \(\text{CodeSet}_{\text{notcontains}}\), fine filtering is performed according to Formula (1), and the data conforming to this equation are added to the result set \(\text{RangeQuerySet}\).

4.3. Spatiotemporal KNN query

The spatiotemporal KNN query aims to search for the spatial nearest K points in the time range \((\text{mintime}, \text{maxtime})\) from the target point \(P(\text{lon}, \text{lat}, \text{time})\). Based on the proposed code index method, the steps of the spatiotemporal KNN query are as follows:

(1) Calculation of the code level \(m = \left\lceil \frac{\log(N_{\text{time}}/(\text{maxtime} - \text{mintime}))}{\log(2)} \right\rceil\).

(2) Calculation of the grid code \([h^3_m]_8\), where the target point \(P(\text{lon}, \text{lat}, \text{time})\) is located at the level \(m\).

(3) All \(n\) points encoded as \([h^3_m]_8\) are grouped into a point set \(P_n\), and the spatial distance of each point in \(P_n\) from the target point is calculated according to Equation (6).

\[
S_i = \text{distance}(P, P_i) = \sqrt{(\text{lon} - \text{lon}_i)^2 + (\text{lat} - \text{lat}_i)^2} \quad i = (1, 2, 3, \ldots, n)
\]

(4) By sorting in ascending order in accordance with the spatial distance from the target point, the top \(K\) points in the point set \(P_n\) are filtered out and added to the query result set \(\text{KNNQuertSet}\). If the size of the query result set \(\text{KNNQuertSet}\) fails to reach \(K\), the query range is expanded once, and the newly queried point is added to the point set \(P_n\). Then, sorting and filtering are performed again. If the size of the result set \(\text{KNNQuertSet}\) still fails to reach \(K\), then the extension is performed again until the size of the set \(\text{KNNQuertSet}\) reaches \(K\) (Figure 6).
5. Experiment and analysis

To verify the performance of the encoding index method in this paper, we use C++ based on MySQL database to implement the index method based on the Hilbert curve code. The trajectory data originate from the T-Driver project of Microsoft Research Asia [29]. The data are collected from the taxi GPS equipment in Beijing Real Road Network and include the trajectory data of 10,357 taxis over a week. In this paper, 1 million trajectory data records are selected to build a trajectory database. Each field in the database table records the ID, licence plate number, code, longitude, latitude, time and other attributes of each trajectory data (Table 3). The hardware environment of this experiment includes a CPU Intel Core i7-7700K (dual core 4.2 GHz), with 64 Gb memory and 7200 r/min hard disk.

| Field     | Types   | length | Precision | Description            |
|-----------|---------|--------|-----------|------------------------|
| ID        | Int     | 6      | 6         | Identification code    |
| Taxi ID   | String  | 10     | 6         | Number plate           |
| Code      | String  | 30     | 6         | Code                   |
| Lat       | Double  | 6      | 6         | Longitude              |
| Lon       | Double  | 6      | 6         | Latitude               |
| Time      | Double  | 6      | 6         | Time                   |
| Attributes| JSON    | 30     | 6         | Whether to carry passengers, speed… |

5.1. Verification of the correctness of the subdivision coding method

This paper designed the following experiments with the following steps to verify the correctness of spatiotemporal meshing and coding methods:

Firstly, all the data in the database are generated to form a set of spatiotemporal points \( P = \{ p_i(\varphi_i, \lambda_i, t_i) | 1 \leq i \leq n \} \). Then, in accordance with the spatiotemporal grid coding method of this paper, for the point \( p_i \) in the point set \( P \), the Hilbert curve code \([h_m^3][8]\) is generated under the level \( m (1 \leq m \leq 26) \). Finally, in accordance with the decoding method, the code \([h_m^3][8]\) are decoded to obtain the spatial coordinates \((i, j, k)\) of the grid cells corresponding to the coordinate point \( p_i \) at the level \( m (1 \leq m \leq 26) \). In each dimension, the distance between \( p_i \) and the vertex closest to the origin in the grid unit \((i, j, k)\) is calculated. The distances in the three dimensions are recorded as \( d_\varphi, d_\lambda \) and \( d_t \). In the case where the encoding and decoding algorithms are correct, \( d_\varphi, d_\lambda \) and \( d_t \) should respectively satisfy the following conditions:

\[
\begin{align*}
    d_\varphi & \leq \Delta\varphi \\
    d_\lambda & \leq \Delta\lambda \\
    d_t & \leq \Delta t
\end{align*}
\]  (5)
where $\Delta \varphi, \Delta \lambda$ and $\Delta t$ are the grid scales in each dimension at level m. Considering the spatiotemporal coordinate point $p(150^\circ0'0'',110^\circ0'0'',6\text{ – 1}\text{ 00: 00: 00})$ as an example, the correctness verification experiment process is as follows:

In accordance with the encoding and decoding algorithms of this paper, the code of each hexadecimal code and the coordinates of the vertices are calculated separately. The results of several hierarchical experiments are shown in Table 4. After calculation, the distance between the vertex coordinates and $p$ in each dimension is smaller than the grid scale, consistent with the correctness condition of the split code.

**Table 4.** Encoding and decoding results.

| Level | Code       | Vertex coordinates        | Grid scale       | Distance in each dimension |
|-------|------------|---------------------------|------------------|----------------------------|
| 5     | 72502      | Longitude: 146°15'0''     | Latitude: 106°52'30'' | Time: 6 – 100: 00: 00      |
|       |            | Longitude: 11°15'0''     | Latitude: 5°37'30'' | Time: 16day                |
|       |            | Longitude: 3°45'0''     | Latitude: 3°07'30'' | Time: 0day                 |
| 15    | 725021525121525 | Longitude: 149°59'46.28164'' | Latitude: 109°59'42.4219'' | Time: 6 – 100: 00: 00      |
|       |            | Longitude: 0°0'39.5508'' | Latitude: 0°1'9.7754'' | Time: 32min                |
|       |            | Longitude: 0°0'20.4492'' | Latitude: 0°0'17.5781'' | Time: 0min                 |
| 26    | 72502152512152512152 | Longitude: 149°59'59.9871'' | Latitude: 109°59'59.9987'' | Time: 6 – 100: 00: 00      |
|       |            | Longitude: 0°0'0.0193119'' | Latitude: 0°0'0.00965595'' | Time: 1s                   |
|       |            | Longitude: 0°0'0.0129''  | Latitude: 0°0'0.0013''  | Time: 0s                   |

Given the analysis in Table 4, the higher the code level is, the smaller the grid granularity is, and the higher the resolution is, the more accurate spatiotemporal information can be identified. In addition, different hierarchical codes have the same prefix, indicating a good parent–child relationship.

### 5.2. Code construction efficiency

For all the data in the database, 12–26 levels of codes were generated, and the average time required to construct a code is statistically compared. The experimental results are shown in Figure 7. As the code level increases, the time required for code construction increases. The reason is that as the level increases, the grid resolution increases, which prolongs the coding construction time. However, each datum takes less than 45 µs, and the cost of code construction is negligible in data operations.
5.3. Spatiotemporal range query efficiency comparison

The proposed algorithm is compared with the existing Geohash-coded index algorithm to verify the performance of the spatiotemporal range query algorithm based on the proposed code index. During the experiment, random queries are performed based on the different size query windows. Each query window is repeated 500 times, and the average time required for the two query methods is counted. The results are shown in Figure 8. The space ranges of the query window are $0.01^\circ \times 0.01^\circ$, $0.02^\circ \times 0.02^\circ$, $0.03^\circ \times 0.03^\circ$ and $0.04^\circ \times 0.04^\circ$, and the time ranges are 1, 2, 3 and 4 h, respectively.

![Figure 7. Coding construction efficiency.](image)
Figure 8. Spatiotemporal range query time for different query windows.

A comprehensive analysis of experimental results shows the increasing query time required by both algorithms as the query window expands. Table 5 shows the number of records in the data set obtained by the query.

Based on the comparison of the experimental results of different query windows, the spatiotemporal range query algorithm based on the proposed code index is more efficient than the existing Geohash-coded index algorithm. The reason is that the algorithm in this paper codes time, longitude and latitude information. In such case, on the one hand, the range of spatiotemporal query can be narrowed. On the other hand, considerable data can be directly obtained by coarse screening of the 1D code and without the fine screening of multi-dimensional requirement calculation, thus reducing the complexity of spatiotemporal query and improving the query rate. In the Geohash-coded index algorithm, given that the Geohash code only encodes the latitude and longitude information, when the spatiotemporal range query is performed, the Geohash code can only roughly filter out the data that conform to the space constraint. All the data obtained from the coarse screening still need to be filtered out by the time index, resulting in increased complexity of the spatiotemporal query and lower query rate.

Table 5. Amount of trajectory data in the query result.

| Spatial query range | 1h  | 2h  | 3h  | 4h  |
|---------------------|-----|-----|-----|-----|
| 0.01 x 0.01         | 860 | 1938| 2034| 2085|
| 0.02 x 0.02         | 2017| 4267| 5595| 5844|
| 0.03 x 0.03         | 3417| 7175| 9702| 10139|
| 0.04 x 0.04         | 3895| 7943| 11462| 11968|

5.4 Comparison of spatiotemporal KNN query efficiency

The proposed algorithm is compared with the Geohash algorithm and the traversal query algorithm to verify the efficiency of the proposed coded index in the spatiotemporal KNN query. The nearest neighbour number is set as $K = (10,100,1000,10000,20000)$, and the experimental results are shown in Figure 9.

Figure 9. Spatiotemporal KNN query time with different K values.
The analysis of Figure 9 shows that the query efficiency of the proposed algorithm and the Geohash algorithm is higher than that of the traversal query algorithm. The reason is that the traversal query algorithm needs to calculate the spatial distance of all the records in the database, whereas the proposed algorithm and Geohash algorithm can reduce the required calculation times by the code index method. Compared with the Geohash algorithm, the proposed algorithm can directly filter the time, longitude and latitude information through the code index. The number of records in the filter result set obtained is less than that of the Geohash algorithm, which results in fewer computation times and improved query efficiency. Compared with the query time of the proposed algorithm at different K values, the query time required for K=10, 100, 1000 is small. The reason is that the grid cell neighbourhood expansion is not performed at this time, and the number of calculations is small. When K=10000, 20000, the 1st and 2nd neighbourhood expansions are performed, respectively. The calculation times increase, thus resulting in the increased query time.

6. Conclusions
For large-scale spatiotemporal trajectory data, this paper designed a trajectory data encoding index method based on the Hilbert curve and described in detail the spatiotemporal subdivision method, coding design and index structure. In accordance with the different data query requirements, different data query processes are designed based on the code index method. Experiments focusing on the correctness of subdivision coding, the efficiency of code index construction and data query of the code index are designed to verify the performance of the proposed code index method. The experimental results reveal that the code designed in this paper features self-similarity and recursion, which can better reflect the process and results of segmentation. Compared with the Geohash code index method, the proposed code operates faster in multiple data query operations and is more suitable for the code index of massive multi-scale trajectory data. The future work will focus on the flexible configuration of the spatiotemporal grid model domain, which makes the split-encoding index method more suitable for more complex spatiotemporal data management.

References
[1] Yue Y, Wang H D, Hu B, et al. 2012 Exploratory calibration of a spatial interaction model using taxi GPS trajectories J. Comput. Environ. and Urban Syst. 36 140–153
[2] Zhan X, Hasan S, Ukkusuri S V, et al. 2013 Urban link travel time estimation using large-scale taxi data with partial information J. Transp. Res. Part C: Emerging Technologies 33 37–49
[3] Gong H, Chen C, Bialostozky E, et al. 2012 A GPS/GIS method for travel mode detection in New York City J. Computers Environ. and Urban Syst. 36 131–139
[4] Castro P S, Zhang D and Li S 2012 Urban traffic modelling and prediction using large scale taxi GPS traces J.
[5] Qian C, Yi C, Cheng C, et al. 2019 GeoSOT-based spatiotemporal index of massive trajectory data ISPRS Int. J. of Geo-Inf. 8 284
[6] Theodoridis Y, Vazirgiannis M and Sellis T 1996 Patio-temporal indexing for large multimedia applications Proc. IEEE Int. Conf. on Multimed, Comput. & Syst.
[7] Pfoser D, Jensen C S and Theodoridis Y 2000 Novel approaches to the indexing of moving object trajectories Proc. VLDB pp 395–406
[8] Li G and Tang J 2011 A new R-tree spatial index based on space grid coordinate division Proc. Int. Conf. on Inf., Cybern., and Comput. Eng. (ICCE2011) (Melbourne, Australia) pp 19–20
[9] Wang L and Xu 2010 Q GPS-free localization algorithm for wireless sensor networks J. Sens. 10 5899–926
[10] Van der Spek S, Van Schaick J, De Bois P, et al. 2009 Sensing human activity: GPS tracking. J. Sens. 9 3033–55

[11] Deng K, Xie K, Zheng K, et al. 2011 Trajectory indexing and retrieval (Computing with spatial trajectories) (New York: Springer) pp 35–60

[12] Frentzos E 2008 Trajectory data management in moving object databases PhD Book (University of Piraeus: Department of Informatics) 10

[13] Chakka V P, Everspaugh A and Patel J M 2003 Indexing large trajectory data sets with SETI Proc. CIDR., 75: 76.

[14] Zhou P, Zhang D, Salzberg B, et al. 2005 Close pair queries in moving object databases Proc. of the 13th annual ACM Int. Workshop on Geogr. Inf. Syst. pp 2–11

[15] Wang L, Zheng Y, Xie X, et al. 2008 A flexible spatio-temporal indexing scheme for large-scale GPS track retrieval Proc. of the Ninth Int. Conf. on Mob. Data Management (mdm 2008) IEEE pp 1–8

[16] Fox A, Eichelberger C, Hughes J, et al. 2013 Spatio-temporal indexing in non-relational distributed databases Proc. of IEEE Int. Conf. on Big Data pp 291–99

[17] Chen S, Ooi B C, Tan K L, et al. 2008 ST 2 B-tree: a self-tunable spatio-temporal b+-tree index for moving objects Proc. of the ACM SIGMOD Int. Conf. on Management of Data pp 29–42

[18] Guan X, Cheng B, Li Z, et al. 2017 ST-hash: An efficient spatiotemporal index for massive trajectory data in a NoSQL database Proc. of Int. Conf. on Geoinformatics

[19] Geohash http://en.wikipedia.org/wiki/Geohash [Online; accessed 20-June-2013].

[20] Balkić Z, Šoštarić D and Horvat G 2012 GeoHash and UUID Identifier for Multi-Agent Systems Kes Int. Symposium on Agent & Multi-agent Systems: Technologies & Applications

[21] Xiang Longgang, Wang Dehao and Gong Jianya 2017 Geohash coding organization and efficient range query for large-scale trajectory data J. of Wuhan Univ.: Inf. Sci. Edition 1 21–27.

[22] Tong Xiaochong, Wang Wei, Wang Lin, et al. 2016 An effective multi-scale time-segmentation method and integer coding calculation J. of Surv. and Mapp. z 1

[23] Moon B, H. Jagadish, C. Faloutsos, et al. 2001 Analysis of the clustering properties of the Hilbert space-filling curve J. Knowl. & Data Eng. IEEE Transactions on 13 124–141

[24] Zhao Xuesheng, Qi Jin, Sun Wenbin, et al. 2016 Review of the research progress of earth-cutting grids J. Acta Geodae Sinica 45 1–14

[25] Butz A R 1969 Convergence with Hilbert’s space filling curve J. of Comput. and Syst. Sci. 3 128–146.

[26] Faloutsos C and Roseman S 1989 Fractals for secondary key retrieval Proc. of the eighth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems pp 247–252

[27] Peano G 1890 Sur une courbe, qui remplit toute une aire plane J. Mathematische Annalen 36 157–160

[28] Li C and Feng Y 2005 Algorithm for Analyzing N-Dimensional Hilbert Curve Proc. Int. Conf. on Adv. in Web-age Inf. Management

[29] Yuan J, Zheng Y, Xie X, et al. 2013 T-Drive: Enhancing Driving Directions with Taxi Drivers’ Intelligence IEEE Transactions on Knowledge and Data Engineering 25 220–232
Acknowledgement: the National Natural Science Foundation of China, No. 41401465, 41371384; National Defense Foundation of China, No. 3601015.