

Article

Non-Uniform Spatial Partitions and Optimized Trajectory Segments for Storage and Indexing of Massive GPS Trajectory Data

Yuqi Yang ¹, Xiaoqing Zuo ^{1,*}, Kang Zhao ² and Yongfa Li ¹

¹ Institute of Land and Resources Engineering, Kunming University of Science and Technology, Kunming 650093, China; yyq0730@stu.kust.edu.cn (Y.Y.); yfli@stu.kust.edu.cn (Y.L.)

² Department of Natural Resources of Yunnan Province, Kunming 650224, China; kzhao@whu.edu.cn

* Correspondence: zxq@kust.edu.cn

Abstract: The presence of abundant spatio-temporal information based on the location of mobile objects in publicly accessible GPS mobile devices makes it crucial to collect, analyze, and mine such information. Therefore, it is necessary to index a large volume of trajectory data to facilitate efficient trajectory retrieval and access. It is difficult for existing indexing methods that primarily rely on data-driven indexing structures (such as R-Tree) or space-driven indexing structures (such as Quadtree) to support efficient analysis and computation of data based on spatio-temporal range queries as a service basis, especially when applied to massive trajectory data. In this study, we propose a massive GPS data storage and indexing method based on uneven spatial segmentation and trajectory optimization segmentation. Primarily, the method divides GPS trajectories in a large spatio-temporal data space into multiple MBR sequences by greedy algorithm. Then, a hybrid indexing model for segmented trajectories is constructed to form a global spatio-temporal segmentation scheme, called HHBITS index, to achieve hierarchical organization of trajectory data. Eventually, a spatio-temporal range query processing method is proposed based on this index. This paper implements and evaluates the index in MongoDB and compares it with two other spatio-temporal composite indexes for performing spatio-temporal range queries efficiently. The experimental results show that the method in this paper has high performance in responding to spatio-temporal queries on large-scale trajectory data.

Keywords: GPS trajectory data; trajectory segments; greedy algorithm; spatio-temporal index; Hilbert curves; trajectory spatio-temporal query



Citation: Yang, Y.; Zuo, X.; Zhao, K.; Li, Y. Non-Uniform Spatial Partitions and Optimized Trajectory Segments for Storage and Indexing of Massive GPS Trajectory Data. *ISPRS Int. J. Geo-Inf.* **2024**, *13*, 197. <https://doi.org/10.3390/ijgi13060197>

Academic Editors: Wolfgang Kainz and Hartwig H. Hochmair

Received: 1 March 2024

Revised: 27 May 2024

Accepted: 8 June 2024

Published: 12 June 2024



Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

With the widespread use of GPS intelligent terminals, the advancement of navigation and positioning technology, and the rapid development of mobile Internet, the geographic location, movement status, and related attributes of data collected by positioning devices at different moments during the movement of mobile objects in geospace, namely, trajectory data, are being generated at an unprecedented scale and speed [1,2]. As one of the world's leading transportation platforms, Didi Taxi (DDT) has more than 550 million registered users, generates more than 106 TB of vehicle trajectory data every day, and processes more than 40 billion routing requests [3]. In such datasets, a trajectory is an ordered sequence of locations that can be arranged in chronological order as $T = \{p_1, p_2, p_3, \dots, p_n\}$, where $p_i = \{x, y, t, o\}$, (x, y) is usually the spatial dimensional information (latitude and longitude coordinates) of the trajectory point, t is the temporal dimensional information (sampling time) of the trajectory point, and o is the descriptive information about other relevant attributes or features that may be present in addition to the spatiotemporal characteristics of the trajectory point [2,4]. Figure 1 illustrates the motion paths of each moving object in three-dimensional time-space as well as their actual motion paths in the two-dimensional space plane for different time intervals.

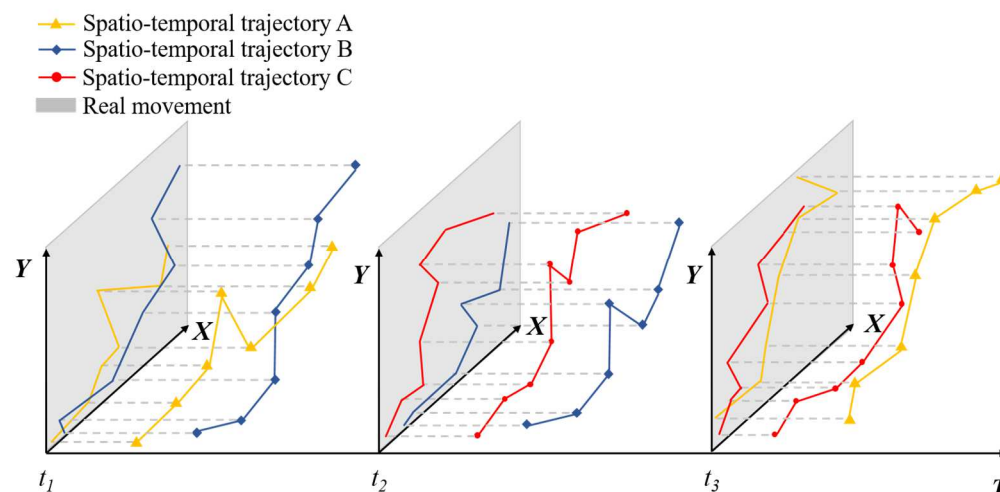


Figure 1. Spatio-temporal trajectories.

Massive spatio-temporal trajectory data provide unprecedented information for understanding the changing patterns of human activities in geospace and cyberspace, facilitating the development of location-based social networks, intelligent transportation systems, and urban computing [5,6]. At the same time, the surge in data volume, the dynamic time-varying nature, and complex road networks bring challenges to the efficient management and query of massive trajectory data [7]. In addition, due to the different behavioral characteristics and overall laws of the mobile objects themselves, the trajectory data generated by them are characterized by very uneven spatial and temporal distribution, which further affects the efficiency of trajectory data indexing and querying.

The spatially efficient indexing representation and processing framework is crucial for trajectory data management. Currently, most solutions for storing and indexing trajectory data rely on R-Tree [8] and variants of R-Tree [5,9]. For indexing spatio-temporal trajectory data, their temporal information is generally involved as the third dimension in the construction of the R-Tree, and the whole trajectory or trajectory segment is stored in the minimum bounding rectangle (MBR) for indexing [10]. However, maintaining this indexing structure is quite complicated. On the one hand, when trajectories radiate over a wide spatial range and a long time span, whether indexing trajectory points or trajectory segments, there are bound to be too large boundaries of individual MBRs as well as a large number of overlapping or redundant MBRs, which makes the MBR-based approach very inefficient and less scalable [8,11]. On the other hand, queries for large trajectory datasets focus on specific geographic regions and time intervals rather than individual trajectories. Thus, there may be thousands of trajectories in a given spatio-temporal region that overlap with the query range. However, since the intermediate nodes of R-Tree allow directory rectangles to cover and overlap, multiple paths and multiple nodes need to be traversed during data retrieval, and the time cost of the query blows up [12].

As the comprehensive requirements of applications become more and more prominent, the full-time mobile object data model and its indexing methods are being intensively investigated, and hybrid indexing methods are becoming a development trend [13]. Aiming at the above problems, we hope to divide trajectories by predefined spatio-temporal granularity; group sub-trajectories that are close to each other spatially and temporally; and provide efficient, flexible, and scalable trajectory indexing and storage strategies. Therefore, this paper proposes the HHBITS hybrid index model (Hash and Hilbert combined B⁺-Tree-based Index for Trajectory Segmentation) for non-relational databases (NoSQL), a massive trajectory data organization and management strategy that combines Hash tables, Hilbert curves, and B⁺-Tree, to obtain the required spatio-temporal trajectory data with the most efficient query processing.

2. Related Work

2.1. Spatio-Temporal Index of Trajectory

To enhance the efficiency of spatio-temporal retrieval of trajectories, the most valid method is to construct a high-performance spatio-temporal index and develop a storage management system for trajectory data. At present, the construction of efficient spatio-temporal indexes for massive trajectory data mainly relies on data-driven and space-driven methods.

The core of data-driven spatio-temporal indexing methods is based on the realization of the deformation and extension of the traditional two-dimensional spatial indexing structure [14]. Among them, the use of R-Tree or its variants to index trajectories is more representative. Wang et al. [15] proposed a distributed trajectory R-Tree, which realizes parallel spatio-temporal querying of massive trajectories through flexible and scalable data partitioning and indexing strategies. Kang et al. [16] implemented a distributed spatio-temporal trajectory segmentation and query framework in a cloud environment, and constructed a distributed spatial R-Tree based on segmented trajectories for scalable trajectory query processing in terms of both memory processing and graph storage access. However, when building an R-Tree to manage large amounts of trajectory data in distributed platforms such as Spark and Hadoop, there can be problems with indexes consuming large amounts of memory as well as large numbers of serialization and deserialization operations, which can severely impact index performance and query efficiency. Moreover, with the continuous movement of mobile objects in spatio-temporal dimensions, the trajectory data increase exponentially, and the R-Tree must be updated as the trajectory data are updated. When confronted with new trajectory data, these spatial data management systems based on Spark and Hadoop usually need to rebuild the index from scratch to achieve excellent query efficiency, which is highly time-consuming.

Spatial-driven spatio-temporal indexes are widely used because of their easy implementation and maintenance, and show far better performance than R-Tree in the face of highly concurrent access and operations. A common approach is to use the GeoHash algorithm to meet the needs of high-frequency updates and organization of trajectory data, as well as common trajectory query operations [17–19]. Qian et al. [14,20] extended the GeoSOT spatial partitioning model to realize multi-level spatio-temporal grid indexing of trajectories and provide efficient trajectory query support. In GCOTraj [21], a large spatio-temporal data space is partitioned into multi-dimensional grid cells, and the data blocks are sorted by space-filling curve (SFC)-based and Graph-Based Ordering (GBO) methods, which significantly improves the querying efficiency. TripCube [9] maintains a complex indexing structure when trajectory data increase dramatically through a flexible trajectory partitioning strategy and a cube-shape-based indexing structure, and demonstrates superior query efficiency when faced with large vehicle trajectory data for a given road network. This space-driven spatio-temporal index structure is easy to build and maintain, with high query efficiency and low memory overhead, but it is difficult to dynamically adjust the index structure according to the denseness of the data without predicting the spatio-temporal distribution of the data, which makes it less flexible and scalable.

2.2. Storage and Querying of Trajectory

For the purpose of storing massive trajectory data in a single-computer environment, researchers have explored this issue and proposed corresponding solutions. They extended the traditional RDBMS to provide efficient data storage management and complex query processing support by optimizing the system's native indexing and querying mechanisms [22,23]. Additionally, TrajStore [24] and SharkDB [25], which are designed for trajectory data storage, have also appeared accordingly.

Nowadays, high-performance storage and computation is an important means of researching and solving challenging problems in various fields [26]. Distributed storage and parallel computing frameworks represented by Hadoop, HBase, Spark, and Flink are gradually being used [27]. Bakli et al. [28] implemented HadoopTrajectory based on

Hadoop, which provides efficient trajectory indexing support and spatio-temporal parallel computing by extending HDFS. Qin et al. [29] proposed a storage and partitioning model for massive trajectory data management for HBase and implemented a co-processor-based multi-level index structure to accelerate spatio-temporal queries. Subsequently, to handle the similarity query problem of large-scale trajectory data, DFTHR [30], a distributed trajectory similarity query framework based on HBase and Redis, was proposed that ensures efficient trajectory query processing. Li et al. [31] extended Geomesa to implement TrajMesa, which supports the storage management of massive trajectories with multiple trajectory query functions. Furthermore, some distributed trajectory analysis systems based on Spark have received attention. For example, TrajSpark [32], DITA [33], and UITraMan [34] are dedicated to efficient parallel processing and iterative computation of trajectories in memory.

3. Methodology

3.1. Trajectory Segmentation Method

3.1.1. Data Model Definition

In this study, a complete GPS trajectory is divided into multiple smaller trajectory segments and distributed to different partitions for organization. Each partition is the minimum boundary for spatio-temporal queries, and the trajectory segments are used for index construction and query processing. In accordance with the indexing and querying requirements of trajectory data, a data model based on segmented trajectories is designed (Figure 2).

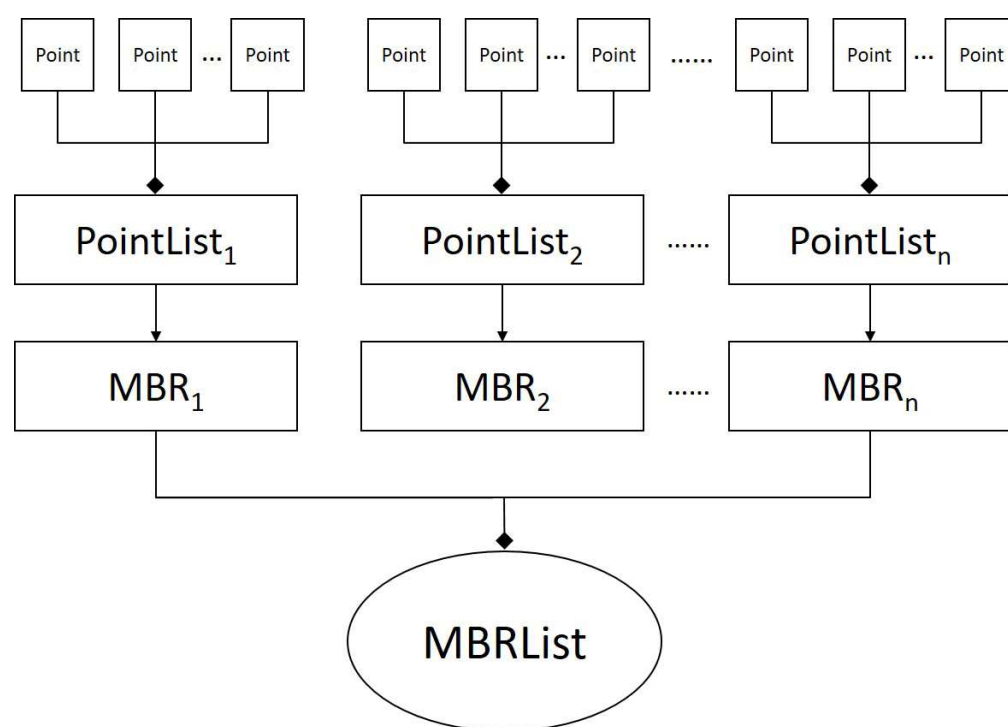


Figure 2. Data model of trajectory segmentation.

Figure 2 gives the definitions of four data entities and their relationships in the data model, which can be described as follows:

1. **Point:** Since GPS trajectories are described as spatio-temporal points, Point is the basic component of this data model. The spatio-temporal attribute with latitude as x , longitude as y , and timestamp as t is denoted as (x, y, t) ;
2. **PointList:** a sequence of trajectory points. This study expresses the trajectory segments by means of the PointList class. PointList consists of multiple Point objects, which are

internally organized and sorted according to a timestamp-ascending data structure, denoted as follows: $\{Point_i\}_N$ ($i = 1, 2, \dots, N$);

3. MBR: the minimum bounding box of a trajectory segment. The MBR class focuses on operations related to trajectory segments, which are processed as basic spatio-temporal units, such as MBR merging in the trajectory-optimized segmentation algorithm and the construction and maintenance of the HHBITS index, which will be described in detail later on. The MBR consists of the PointList corresponding to the sub-trajectory segment and the diagonal vertices representing that MBR, denoted as $(PointList, Point_a, Point_b)$;
4. MBRLIST: MBR sequence. MBRLIST is defined as a container data structure for organizing MBRs in a chained table to facilitate batch manipulation and processing of MBRs, denoted as $\{MBR_i\}_N$ ($i = 1, 2, \dots, N$).

3.1.2. Segmentation for Trajectory Optimization Based on Greedy Algorithm

Most of the raw trajectory data contain a large number of noise points due to various factors such as GPS device failures, sensor errors, transmission errors, and storage errors. Their existence will lead to unsatisfactory trajectory segmentation and ultimately affect the efficiency of storage and indexing. Therefore, the DBSCAN (Density-Based Spatial Clustering of Applications with Noise) algorithm is utilized for noise point removal before trajectory segmentation and index construction.

The DBSCAN algorithm is a density-based clustering algorithm that can be used to identify anomalous points in high-dimensional data. Its clustering principle can be simply interpreted as partitioning high-density point regions into clusters and effectively filtering out low-density point regions, where the density of noise points is less than the density of any cluster class. As shown in Figure 3, the density of cluster A and cluster B of the trajectory is greater than that of the surrounding, while the density of the region where the noise point P_5 is located is less than that of cluster A and cluster B, so the DBSCAN algorithm can also be used for identifying anomalous points in the trajectory data. The DBSCAN algorithm requires the specification of two parameters, ϵ and $minPts$, where ϵ defines the radius of the neighborhood of the point and $minPts$ defines the minimum number of clusters in the clustering.

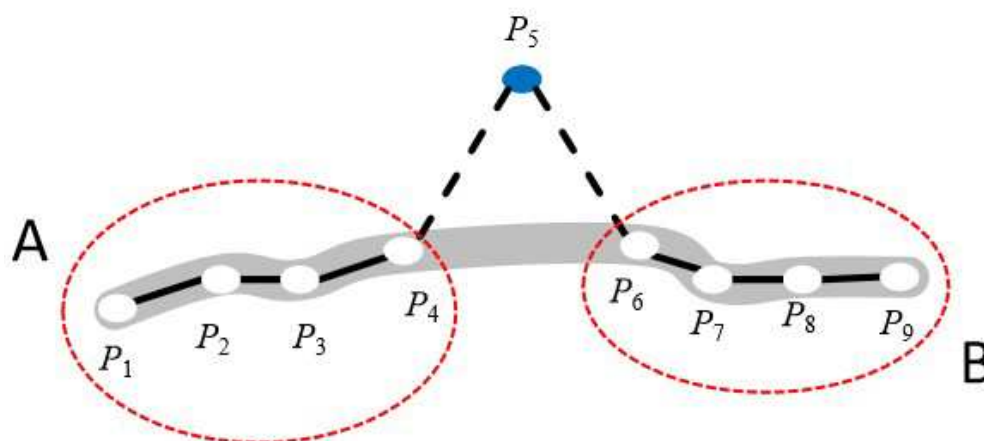


Figure 3. A noisy point in trajectory and clustering.

The following are the general steps for anomaly cleaning of single trajectory data using the DBSCAN algorithm:

1. Select an unvisited trajectory point p as the starting point;
2. Calculate the number of data points in the ϵ -neighborhood of point p . If the number is greater than or equal to $minPts$, mark p as a core point and create a new cluster;
3. Add point p to the current cluster and add all unvisited data points within the ϵ -neighborhood of p to the current cluster;

4. Perform the following operations on each data point q in the current cluster:
 - (1) If q is a core point, add all unvisited data points within the ε -neighborhood of q to the current cluster;
 - (2) If q is not a core point but lies within the ε -neighborhood of another cluster, mark q as a boundary point and add it to the current cluster;
5. When there are no more data points that can be added to the current cluster, the current cluster is considered a complete cluster;
6. Select the next unvisited data point as the starting point and repeat steps 2 to 5 until all data points have been visited;
7. Mark the remaining unallocated data points as noise points and clear them.

The general steps for trajectory noise point removal using DBSCAN are given above, but since the focus of this paper is on the segmentation, indexing, and storage of trajectories, it will therefore not be discussed in more detail here. For the code implementation of DBSCAN and related parameter settings, refer to Github [35]. In addition, the DBSCAN method does not yield optimal trajectory cleaning, which is chosen in this paper to ensure the quality of the trajectory segmentation as much as possible.

In this paper, segmented spatio-temporal indexing of trajectory data is realized by MBR approximation of spatio-temporal objects. However, the choice of MBR construction method greatly affects the indexing efficiency. As shown in Figure 4, spatio-temporal trajectories with longer time intervals have better data compression, but there may be a large MBR, which results in a large amount of blank space, leading to a lower efficiency of the spatio-temporal query. Spatio-temporal trajectories with shorter time intervals have smaller MBRs and a higher efficiency of the spatio-temporal query, but with poorer data compression capability [36]. Therefore, the goal of optimizing trajectory segmentation is to make a trade-off between MBR size and query efficiency, ensuring that trajectories are segmented according to the specified number of segments, while keeping the MBR size within a reasonable range and minimizing redundant and overlapping MBRs as much as possible.

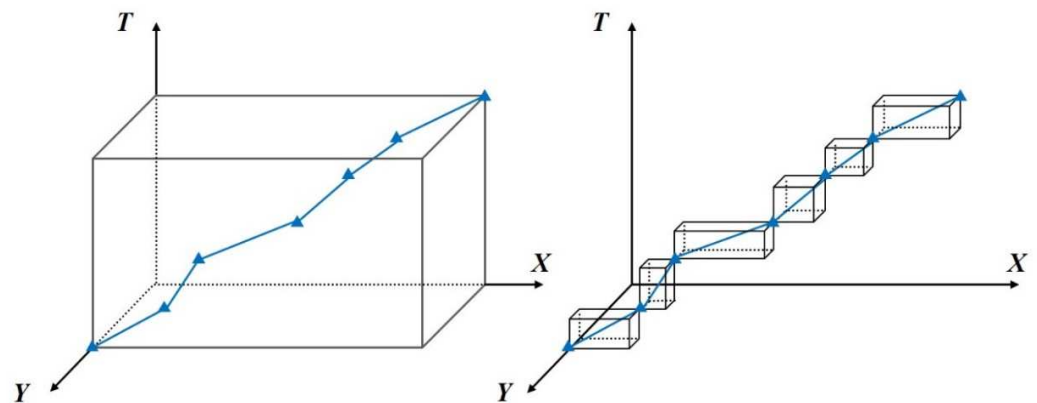


Figure 4. Example of spatio-temporal object splitting.

In this study, the greedy segmentation algorithm [37] is utilized in conjunction with the data model defined in Section 3.1 to achieve maximum N -segment segmentation of a single long trajectory, with the process described in Figure 5. Although the greedy segmentation algorithm cannot guarantee obtaining the global optimal solution, it can ensure that the segments have sufficient length while minimizing the size of the segmented trajectory segment MBR to achieve a local optimal solution. Additionally, the time complexity of the algorithm is relatively low and the actual performance is close to the optimal solution; thus, it is a favorable choice for trajectory segmentation in this study [36].

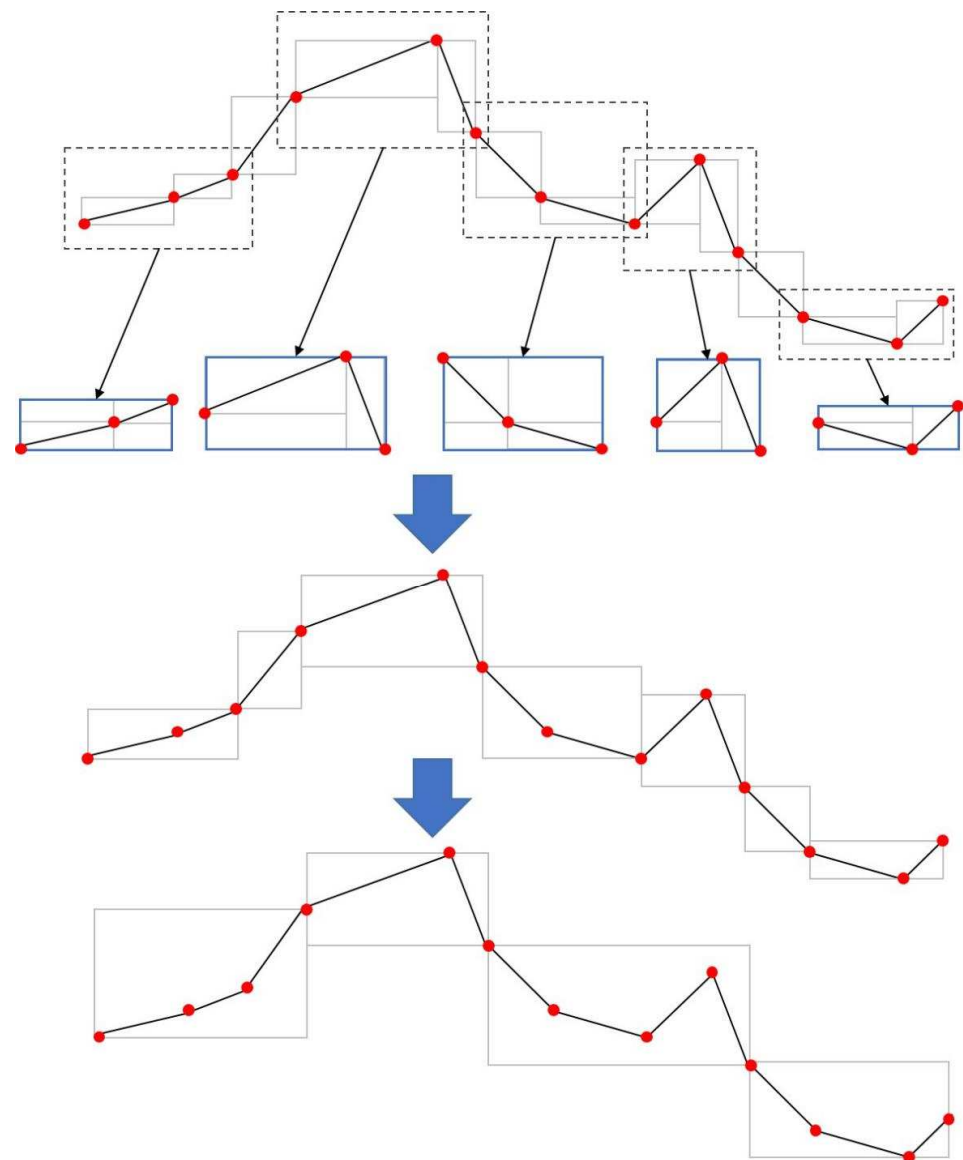


Figure 5. The trajectory greedy splitting process.

1. The greedy segmentation method will be applied independently to each trajectory, executed in three main steps:
2. Select two consecutive points in the trajectory sequence in turn as diagonal vertices to create the MBR sequence;

The merging of neighboring MBRs is considered and the respective sub-trajectory points are distributed into the newly created MBR after merging. In this case, the merging of neighboring MBRs is achieved by means of a predefined merging trade-off criterion [38]:

$$Norm(MBR_i, MBR_j) = V(MBR_{merge}) - V(MBR_i) - V(MBR_j) \quad (1)$$

where V is the computational function of MBR size, and MBR_i and MBR_j are two consecutive MBR objects in the current MBR sequence. The MBR merge operation based on this criterion is listed in Algorithm 1, which selects the MBR corresponding to the current minimum merge tradeoff criterion value for merging;

3. The merging operation in step 2 is performed cyclically, and the merging process is terminated when the number of trajectory segments reaches the division limit.

This workflow is shown in Algorithm 1.

Algorithm 1: Trajectory Segmentation Algorithm.

Input: Trajectory = {point1, point2, ...}, Trajectory ID T_{ID} , segment length N
Output: Segments = Map < T_{ID} , MBRList = {MBR₁, MBR₂, ...}>
1: MBRList $\leftarrow \emptyset$.
2: **for** i in range(1, n) **do**
3: create new Points $pt_1 = \text{Point}(\text{point}_{i-1}.x, \text{point}_{i-1}.y)$, $pt_2 = \text{Point}(pt_i.x, pt_i.y)$;
4: create new MBR $MBR = \text{MBR}(pt_1, pt_2)$;
5: MBRList.add(MBR).
6: **end for**
7: MBRMergeNorm $\leftarrow \emptyset$.
8: **for each** $MBR_i \in \text{MBRList}$ and $MBR_j = \text{MBR}_i.\text{next}()$ **do**
9: $norm < \text{Norm}(MBR_i, MBR_j), i, j > \leftarrow \text{getMergeNorm}(MBR_i, MBR_j)$.
10: MBRMergeNorm.add(norm);
11: **end for**
12: Segments $\leftarrow \emptyset$.
13: **while** MBRList.size() > N **do**
14: MBRindex[a, b] $\leftarrow \text{getMin}(\text{MBRMergeNorm})$.
15: create new MBR $MBR_{new} = \text{runMerge}(MBR_a, MBR_b)$;
16: MBRList.replace(a, MBR_{new}), MBRList.remove(MBR_b)
17: MBRMergeNorm.update();
18: **end while**
19: **return** Segments.Put(T_{ID} , MBRList);

The greedy segmentation algorithm is applied to each trajectory independently. Therefore, this study utilizes Spark for parallel processing of trajectory segmentation.

3.2. Trajectory Spatio-Temporal Index Construction

We propose the HHBITS trajectory spatio-temporal index structure in this study to provide efficient trajectory spatio-temporal query support. For temporal indexing, the joint Hash table provides indexing support for real-time incremental trajectory databases by partitioning uniform intervals in time and sorting and encoding them according to an ascending data structure. For spatial indexing, a combination of space-filling curve and spatial adaptive division is used to realize the reduced dimensional spatial representation of segmented trajectories and to avoid data skewing. This section describes the HHBITS index structure in detail.

3.2.1. Time Index Based on Hash Table

The spatial division of the radial region of the trajectory is definable, whereas time is infinitely extended, and the trajectory grows infinitely with the spatio-temporal changes of the moving object [39]. Therefore, segmentation is performed in both spatial and temporal dimensions as well as distributing the trajectory segments into corresponding spatio-temporal units to improve query efficiency and alleviate the data skewing problem.

A common partitioning method is to treat the temporal dimension as the third dimension of space, enabling synchronous three-dimensional spatial partitioning, and utilizing three-dimensional space-filling curves to encode the partitioning results. However, this method has two drawbacks. Firstly, constructing high-dimensional and high-precision space-filling curves requires more complex encoding computations, resulting in increased time overhead. Secondly, considering that trajectory segments are usually uniformly distributed in time, performing uniform temporal partitioning can significantly reduce the workload of individual nodes [40]. Therefore, in this study, the partitioning and encoding were performed separately in the temporal and spatial dimensions, and the codes were organized using the same index tree, in order to provide efficient trajectory management support while preserving the spatiotemporal distribution characteristics of the trajectories.

We form trajectory partitions in terms of time intervals and divide spatio-temporal trajectories that are in the same interval into the same partition. By encoding the partitions, a simple and efficient hash table is combined to establish the correspondence between the partitions and the codes, and the construction of the time index is realized, as shown in Figure 6, where each *Value* of the hash table is the sequential value obtained by arranging each trajectory time partition in chronological order. Assuming that T_0 is the time corresponding to the earliest trajectory partition in time phase, and dt is the time interval of temporally adjacent trajectory partitions, a simple ascending data structure is adopted to organize the uniquely corresponding time span values in the partition. dt should be set according to the temporal characteristics of the original trajectories, query requirements, and practical application experience for comprehensive consideration.

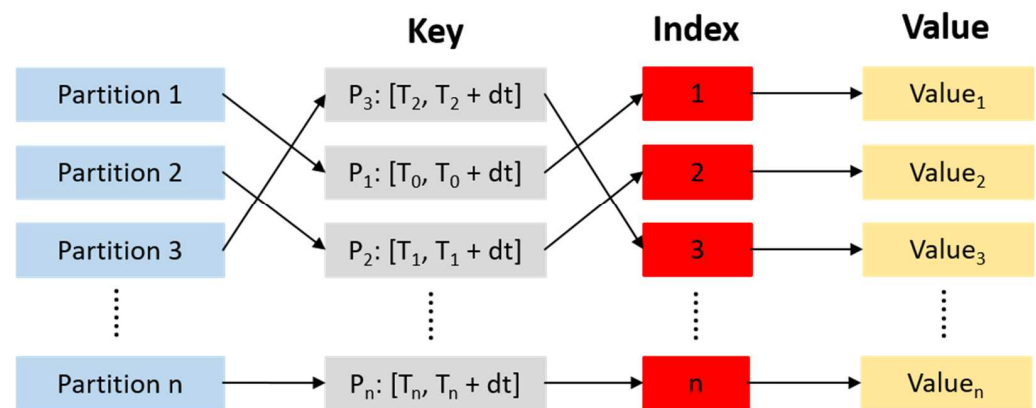


Figure 6. The architecture of the time index.

Subsequently, the trajectories in each independent partition will be converted from trajectory points to MBR sequences by the greedy segmentation algorithm, and a joint Hash table is used to maintain and update the MBR sequences of all the partitions for temporal indexing, which facilitates pruning the querying task by finding the spatio-temporal trajectory segments in a specific interval through the unique partition identifiers, as shown in Figure 6. When a trajectory partition has a time span of dt and its corresponding time interval is $[T, T + dt]$, MBR sequences that are in the same interval are selected to be partitioned into that partition to form a collection of MBR sequences, and a new partition node is created to receive the new MBR sequences.

3.2.2. Spatial Index of Trajectories under Adaptive Partition of Space

A space-filling curve is a one-dimensional continuous curve that traverses a multidimensional space. It has shown high performance in spatial indexing and spatial partitioning for various data types [41]. Commonly used space-filling curves include the Z curve and Hilbert curve [42]. The Z curve has local proximity, although it also has serious spatial mutability, i.e., the subspaces coded by neighboring numbers may not be adjacent to each other, and its coding cannot effectively reflect spatial distance [41]. The Hilbert curve has optimal spatial aggregation and discrete approximation abilities, and adjacent subspaces in space are adjacent and continuous on the curve; thus, it can realize the mapping from multidimensional space to one-dimensional space well [43], as shown in Figure 7.

Compared with the Z curve, the Hilbert curve shows a better spatial clustering effect and does not present a “spatial mutation” phenomenon. Furthermore, the Hilbert curve, as a variant of the Peano curve, has a simpler and more effective implementation (instead of cutting the space into 9 squares of the same size, it is cut into 4 squares of the same size) as well as a wider range of applications [39]. Therefore, the Hilbert curve was chosen as the basis of the coding algorithm in this study.

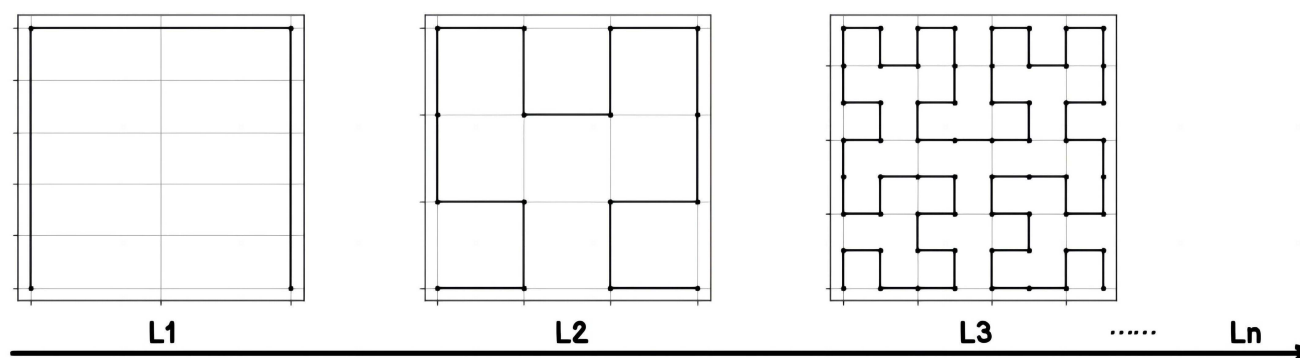


Figure 7. Hilbert curve.

The traditional Hilbert coding uses the idea of gridding each grid as the spatial extent of the partition, but when the data in the space are unevenly distributed, it may lead to the existence of a sub-region with overly dense data, thus generating the problem of partitioned data skew. When new data are loaded into the region, the index update and maintenance will become more complex, and the I/O cost of the query and the size of the candidate set increase, which leads to a reduction in the efficiency of the query [44].

In summary, non-uniform spatial division is considered according to the distribution density of trajectory segments, as shown in Figure 8. The adaptive division of regional space is realized by presetting the threshold of the regional data volume to achieve the purpose of balancing the data volume of each region, which effectively reduces the search space and improves the query efficiency. In order to avoid the high I/O overhead of transmitting the complete dataset in the computation process, we sample the data at a sampling rate of 1% to determine the initial order of the Hilbert [42], and dynamically construct the Hilbert mesh based on this initial order and the local quadrature. We construct the Hilbert lattice dynamically based on this initial order and local quadrature.

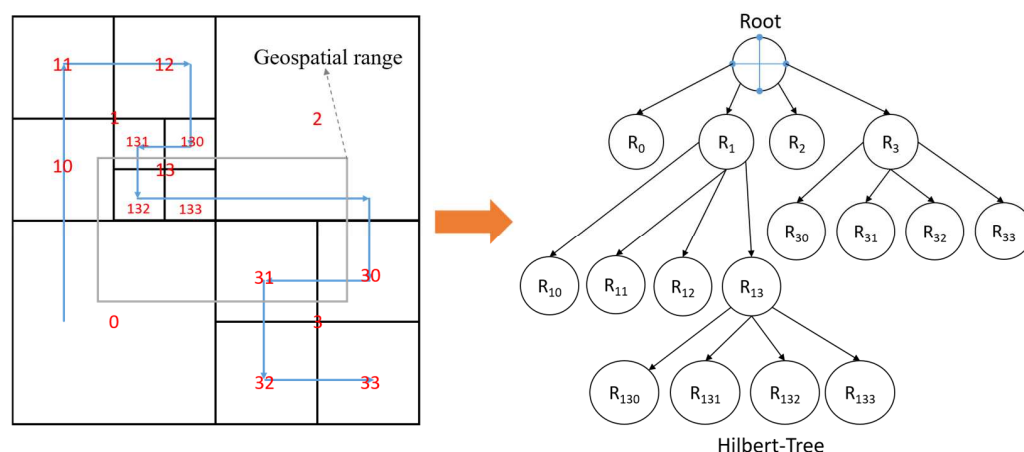


Figure 8. Correspondence between non-uniform spatial divisions and Hilbert curve.

However, MBR objects that are unevenly distributed in space may span multiple divided grids, and deciding which grid they belong to is the boundary data problem. Usually, for such data spanning boundaries, there are specified grids and replication methods [45]. The specified grid method involves selecting one of the multiple grids that the object spans. The copy method copies the objects that span the grids to each grid, but this causes data duplication, causing the results of query-type operations to be erroneous due to large bias. Thus, for the spatio-temporal range query, for avoiding the overhead caused by additional storage and de-duplication operations brought by the subsequent filtering step of the replication method, and considering that we have achieved the optimal segmentation of the trajectory through the greedy algorithm to minimize the blank space

of the MBR of the trajectory segments, which improves the spatial utilization, this study adopts the specified grid method, and guarantees the correctness of the results by paying attention to the boundary selection subsequently.

The complete spatial adaptive partitioning and Hilbert coding process is shown in Algorithm 2.

Algorithm 2: Adaptive Data Segmentation Algorithm Based on Space-Filling Curve Coding.

Input: MBRLIST = $\{MBR_i\}_N$ ($i = 1, 2, \dots, N$), initially selected level N_{ini} , highest level N_{max} , threshold P_{max}

Output: Physical Partition = GridList

1: GridList $\leftarrow \emptyset$.

2: SubdivideGridList $\leftarrow \emptyset$.

3: GridList $\leftarrow$ regularGridSplit(MBRLIST, N_{ini});

4: **for each** $grid_i \in$ GridList **do**

5: **if** ($grid_i$. capacity() $> P_{max}$) **then**

6: SubdivideGridList.add($grid_i$);

7: GridList.remove($grid_i$);

8: **end for**

9: TempGridList $\leftarrow \emptyset$.

10: **while** SubdivideGridList.size() > 0 and $N_{ini} < N_{max}$ **do**

11: TempGridList $\leftarrow$ subdivide(SubdivideGridList).

12: $N_{ini}++$;

13: SubdivideGridList.clear();

14: SubdivideGridList $\leftarrow$ selectSubdivide(TempGridList, GridList);

15: **end while**

16: **return** GridList;

The initial Hilbert curve order (N_{ini}) needs to be determined before spatial adaptive partitioning can be performed. In this paper, the value of N_{ini} is calculated from the perspective of range query. In particular, retrieving spatial objects that intersect with a specific query window ($r^\circ \times r^\circ$) is a typical requirement for neighborhood queries in practical business applications. We assume that the maximum spatial span of the study region is R° . If the spatial region represented by the Hilbert coding is to cover the study region, it should satisfy $2^{N_{ini}} \times r \geq R$, $N_{ini} \geq \left\lceil \log_2 \frac{R}{r} \right\rceil$, and N_{ini} should be normalized and used as the starting order of the Hilbert curve.

At the first division, based on the set initial Hilbert curve order (N_{ini}), the two-dimensional space is uniformly divided into a $2^{N_{ini}} \times 2^{N_{ini}}$ grid. Subsequently, statistics on the number of MBR objects in each Hilbert grid are obtained. If the data contained in a grid cell are less than or equal to the set threshold value (P_{max}), then encoding is performed directly according to the direction of the Hilbert curve; however, if the data contained in a grid cell are greater than or equal to the set threshold (P_{max}), then iterative subdivision is performed in that grid cell alone to construct a locally finer grid. When the number of subdivision orders reaches the pre-set maximum order (N_{max}) or the data contained in a single grid cell are less than the threshold P_{max} , then the subdivision is stopped. P_{max} can be selected based on hardware performance and network speed or the size of the most frequently queried study area. For most applications, a value size of a few thousand to tens of thousands is an appropriate choice [46].

3.3. Trajectory Storage and Query

In this paper, the dense spatio-temporal trajectory data are divided into relatively small groups involved in trajectory spatio-temporal index construction by refining the spatial units, and a spatio-temporal query framework for large-scale trajectory data oriented to NoSQL is designed based on the relational organization model of temporal Hash and Hilbert coding (Figure 9). The main idea is to organize segments of trajectories by spatio-

temporal coding, and to process spatio-temporal range queries of trajectories by using the compound index structure (improved B⁺-Tree) built into the database system.

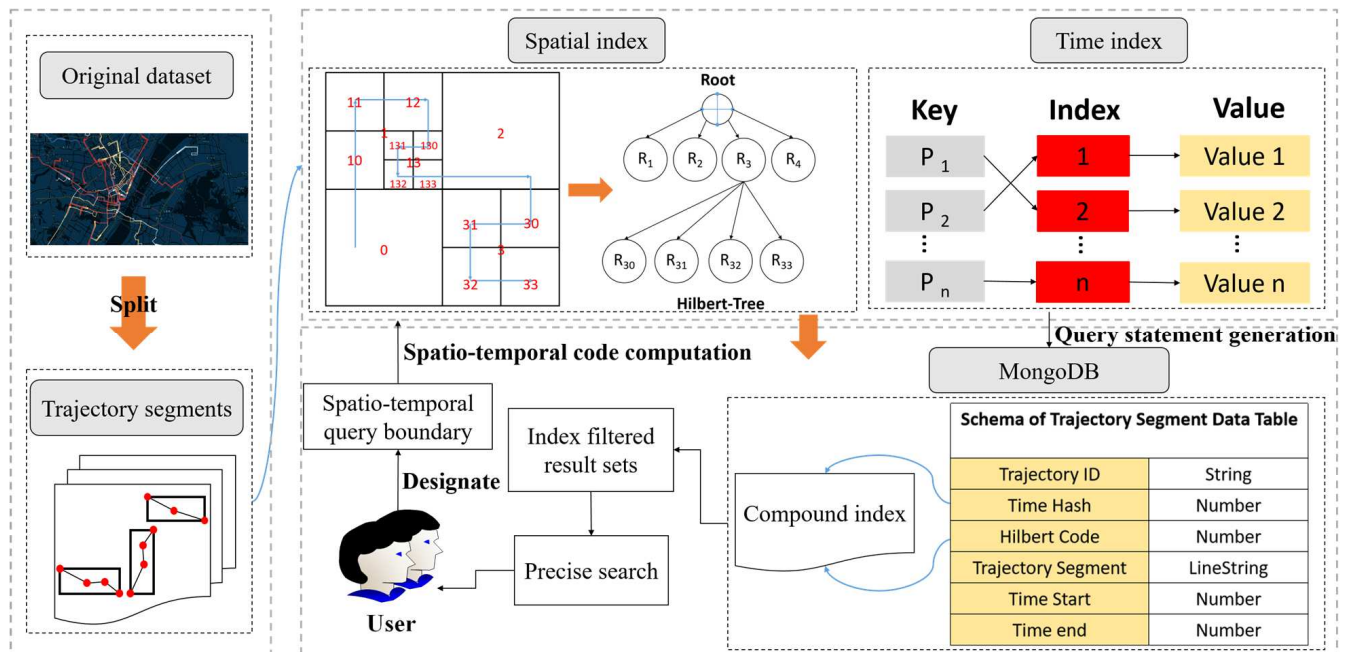


Figure 9. Framework of trajectory segment index for NoSQL.

3.3.1. Storage of Segmented Trajectory Data with MongoDB

In mature spatial databases, R-Tree indexes are usually used to manage trajectory data, and efficient temporal indexing mechanisms are lacking. In contrast, this paper utilizes temporal Hash with Hilbert coding to organize spatio-temporal trajectory data, whose one-dimensionality is easier to manage compared to the complex R-Tree. Therefore, in this paper, the index is ported to the MongoDB database. Primarily, the conversion from trajectory point sequence to trajectory segment sequence is achieved by the greedy segmentation algorithm in Section 3.1, which minimizes the blank space of the MBR of the trajectory segments and improves the space utilization and query accuracy. Subsequently, the temporal Hash code and Hilbert encoding of segmented trajectories are calculated according to the temporal indexing model in Section 3.2. Finally, the segmented trajectory dataset is imported into the database and the compound index is constructed on its temporal hash code and Hilbert coding fields. Compared to building separate indexes on these two columns, the composite index is realized by only one B⁺-Tree, which is able to sort the second index column based on the sorting of the first index column, and has a more efficient query performance, which can accelerate the spatio-temporal query processing on the same segmented trajectory data table. Since Hilbert coding under uneven spatial segmentation is a spatial segmentation coding technique similar to a quadtree, this paper utilizes Hilbert-Tree for description. In summary, range requests normally supported by R-Tree can then be transformed into query processing for temporal Hash tables and Hilbert-Tree and B⁺-Tree indexes. This design facilitates the parallelization of range query processing, and this parallelism can be used for both multiple range queries and single-query processing.

3.3.2. Trajectory Spatio-Temporal Query

The HHBITS index supports all the spatio-temporal query processing operations mentioned in this paper, the more common and important of which are query operations based on specific spatio-temporal boundaries. The spatial range and the time interval are provided as inputs, and the target is to find all the trajectory objects within the range. Combined with the spatio-temporal index model in Section 3.2 of this paper, the spatio-

temporal range query of a trajectory is divided into two phases, i.e., the filtering phase of the HHBITS index and the refinement screening phase. Figure 10 shows the general framework of the “filtering and refinement” model for spatio-temporal range query processing of trajectories. The filtering phase uses a relatively low computational cost to find a candidate set of possible trajectory segments; the refinement phase identifies the final query results from the indexed coarse filter candidate set.

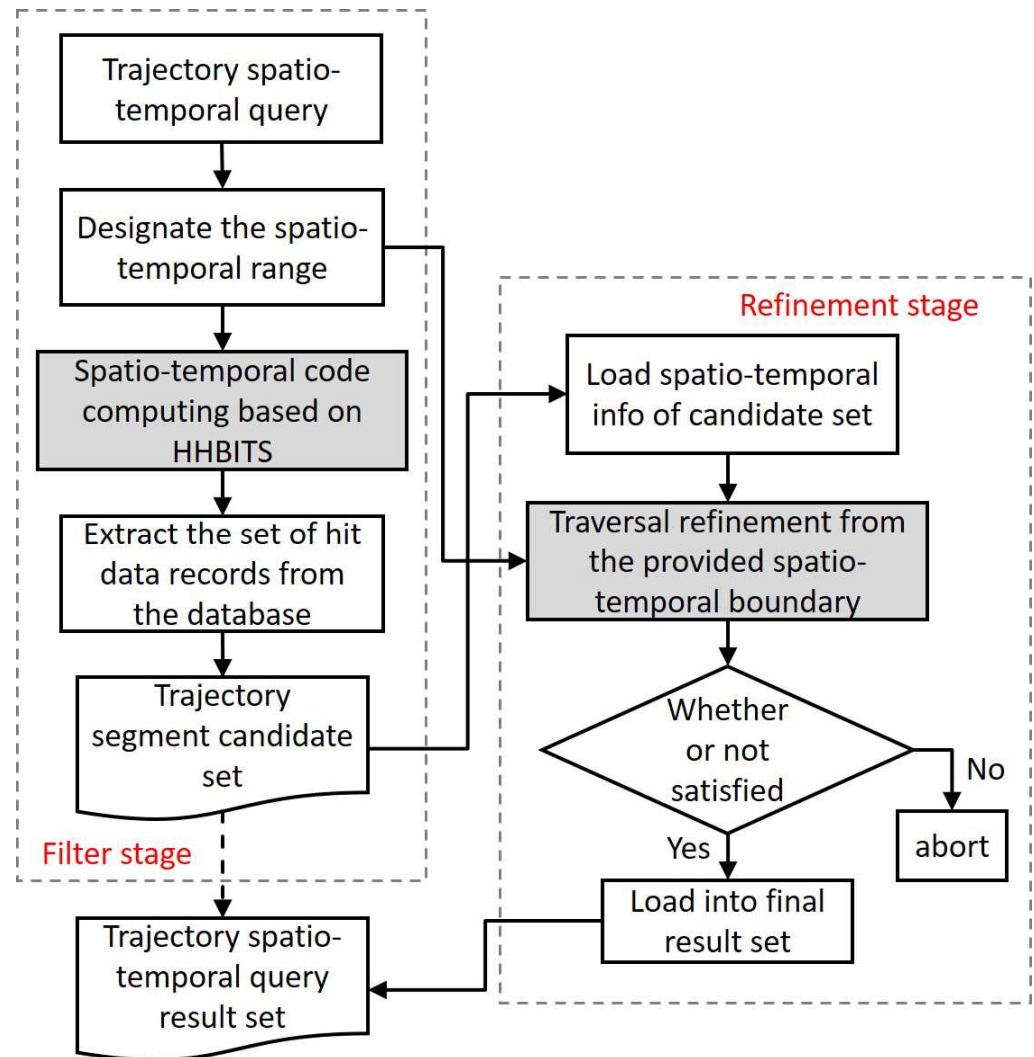


Figure 10. The general framework of spatio-temporal range query.

Among them, the spatio-temporal filtering phase based on HHBITS can be divided into 4 sub-stages:

1. Temporal layer filtering, regardless of the query conditions oriented to a specific point in time or a specific time period, can be indexed and filtered by the temporal Hash table, which maps the temporal range of the query to a specific temporal Hash value or a list of temporal Hash values;
2. Spatial layer filtering, which computes a Hilbert grid encoding set from the Hilbert-Tree based on the given spatial query boundaries;
3. Generating an indexed filter statement based on the filtering results obtained in steps 1 and 2, with filter conditions constructed by combining the corresponding spatio-temporal encoding sets;
4. Delivering the indexed filter statements to the trajectory segment storage table and returning the indexed execution results to await subsequent processing.

In the precise screening phase, the spatio-temporal information of the trajectory segments is extracted from them by traversing the indexed filtering results, and the trajectory segments that are accurately included in the spatio-temporal query boundaries will be screened and returned.

4. Experiments and Results

4.1. Data Description and Experimental Platform

For verifying the feasibility of the hybrid spatio-temporal index structure of trajectories proposed in this paper, this study implements a large-scale GPS trajectory data organization structure and its range query method based on uneven spatial segmentation and trajectory-optimized segmentation based on the MongoDB database, and uses the GPS trajectory dataset from Microsoft GeoLife [47]. The GPS trajectories in this dataset consist of a series of time-stamped points, each of which is represented by various attributes such as its latitude, longitude, and altitude, and contain a total of 18,670 complete GPS trajectories and about 24.9 million trajectory points. In addition, the dataset spans the period from April 2007 to August 2012. The experiments were conducted using Java (JDK1.8) to implement the trajectory-optimized segmentation, partitioned HHBITS construction, and trajectory spatio-temporal query algorithms, with MySQL version 8.0.30, MongoDB version 4.4.0, and Spark version 3.1.2, and were run on a computer configured with an Intel(R) Core(TM) i7-9750H CPU @ 2.60 GHz and 16 GB RAM on a Windows 10 64-bit operating system.

4.2. Performance Analysis

In this subsection, the evaluation experiments will be conducted based on the experimental environment in 4.1 with real datasets. The experiments are mainly divided into two parts: validation of the effect of trajectory segmentation based on the greedy algorithm and experiments on spatio-temporal querying of trajectories based on the HHBITS index.

4.2.1. The Effects of Segmentation Optimization

The segmentation optimization uses the greedy merging method (as described in Section 3.2), which aims to minimize the MBR size of the trajectory segments while maintaining a fixed number of trajectory segments. Through the characterization of the dataset, we noticed that the time span of trajectories with more trajectory point records is usually contained in the interval of 12 to 24 h, while the time span of trajectories with fewer trajectory point records is usually contained in the interval of 0 to 6 h. Therefore, the experiments choose to apply the optimized segmentation method to two different types of trajectories, i.e., long trajectories (spanning more than 12 h) and short trajectories (spanning up to 6 h). We randomly selected 100 trajectories from each of these two types and evaluate the segmentation results by the ratio of the sum of the MBR sizes of the segmented trajectory segments (optimized method/conventional method). Among them, the optimization method is the greedy segmentation algorithm used in this paper. The conventional trajectory segmentation method uses the common trajectory equal interval segmentation method, which means that the trajectory is uniformly divided into a number of trajectory segments with the same number of sub-trajectory points. The number of sub-trajectory points contained in each trajectory segment is $\left\lceil \frac{P_{all}}{N} \right\rceil + 1$, where P_{all} is the number of sub-trajectory points contained in the trajectory and N is the segment length. In some cases, the segment located at the end of the time series contains a different number of sub-trajectory points than the other segments. As shown in Figure 11, both long and short trajectories can effectively reduce the MBR size after segmentation optimization. In addition, the results obtained by applying the optimized segmentation method on long trajectories are better than those on short trajectories, especially when the length of the trajectory segments is high.

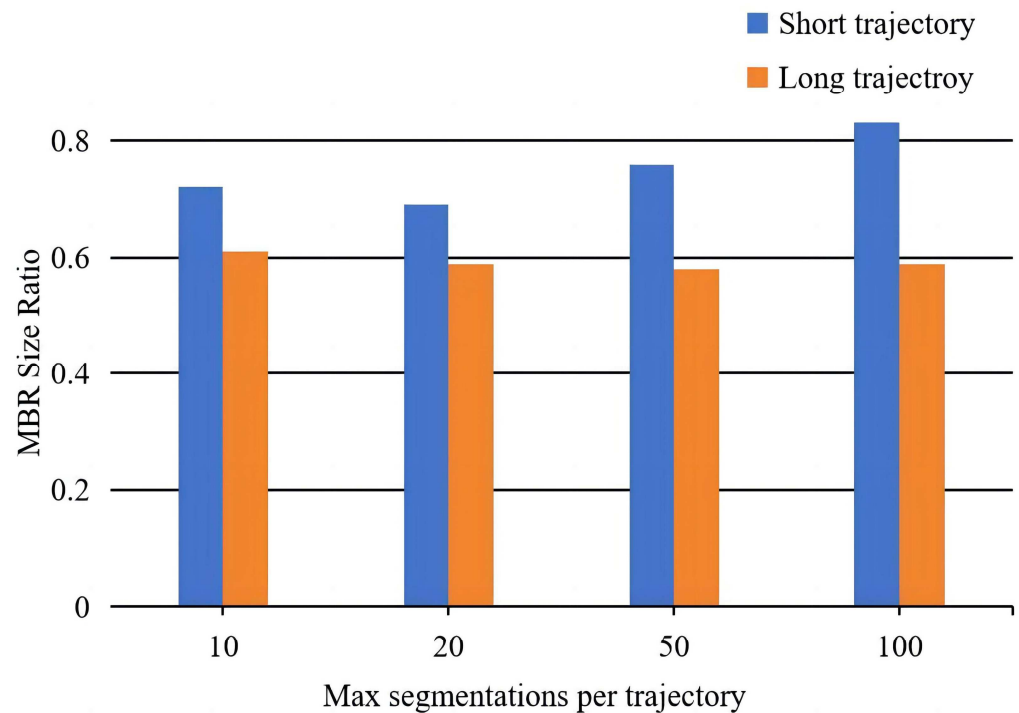


Figure 11. The MBR size ratio of segmentation optimization in different segment lengths.

In this study, the results of trajectory optimization segmentation are used for the construction of indexes to realize the efficient organization of massive GPS trajectory data. In order to verify the query performance based on trajectory optimization segmentation, 3544 trajectories and about 5 million point objects are selected from the original trajectory dataset for the construction of R-Tree, and the spatial query comparison experiments with the conventional trajectory segmentation method are conducted; the results are shown in Figure 12. Due to the optimization of the trajectory segmentation rules, the sum of the bounding box volume of the generated trajectory segments and the area covered and overlapped by the non-leaf nodes of the R-Tree as well as the perimeter are as small as possible, which improves its spatial utilization, reduces the search paths in spatial querying and candidate trajectory segments, and shortens the querying time effectively.

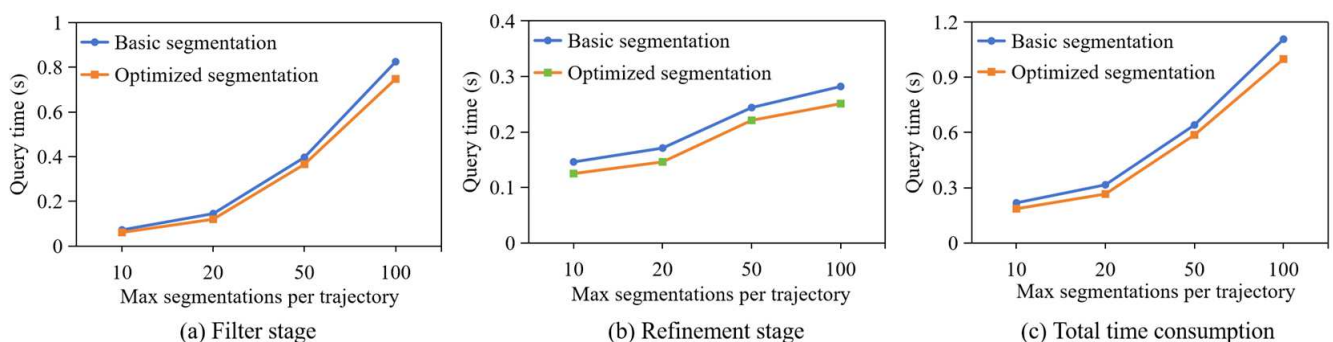


Figure 12. The comparison between basic segmentation and optimized segmentation in spatial query with different segment lengths: (a) index-filtering stage; (b) traversing the refinement stage; (c) total time consumption for spatio-temporal range query.

In addition, this study also examines the query accuracy of the trajectory optimization segmentation method under different segment lengths, and the ratio of the trajectory segments that are truly included in the query range to those returned by indexed coarse filtering is referred to as the query accuracy ratio. The same dataset as the previous

experiment is selected and query tests are conducted under R-Tree with a spatial window of $0.05^\circ \times 0.05^\circ$ to measure the query accuracy and data compression effect of the method, as shown in Figure 13. The results show that the spatial query accuracy of the optimized segmentation method is always higher than that of the conventional segmentation method, and with the increase in the number of trajectory segments, the conventional segmentation method introduces too much blank space, which causes a large number of erroneous values to be loaded into the result set, and the data compression effect is poor. The optimized segmentation method reduces the blank space by minimizing the MBR size of the trajectory segments, which effectively reduces the results that are not within the query range, and the data compression effect is ideal. Therefore, the optimized segmentation method has a better data compression effect and ensures higher query accuracy while supporting efficient trajectory query retrieval.

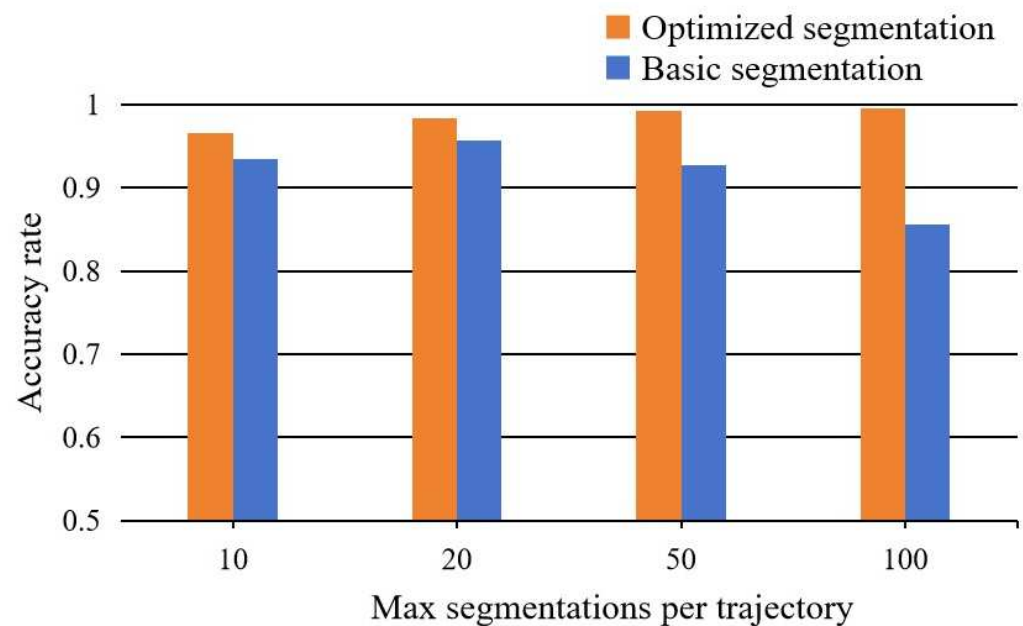


Figure 13. The comparison of query accuracy between basic segmentation and optimized segmentation with different segment lengths.

4.2.2. Trajectory Spatio-Temporal Query

To evaluate the impact of the number of trajectory segments on the spatio-temporal query, the same dataset as the previous experiment is selected for the experiment, with the maximum number of segments per trajectory increased from 10 to 100 for the data import and query test, and the interval is 10. The minimum code length is 5 and the maximum is 12. The threshold of spatial split is 2000. The query range is the space area that corresponds to $0.5^\circ \times 0.5^\circ$ and the time interval is 1 month. The experimental results are shown in Figure 14.

As observed in Figure 14, as the maximum number of segments of a single trajectory increases sequentially, the import time of the segmented trajectory data increases, and the relationship with the maximum number of segments of a single trajectory is positively proportional. The main reason for this is that the larger the maximum number of segments of a single trajectory is, the more segments are written to MongoDB in the identical batch.

In terms of query efficiency, the total number of MBRs increases when the maximum number of segments per trajectory is sequentially increased, and the efficiency of maintaining and updating the indexes in memory and utilizing the indexes for temporal and spatial queries gradually decreases. The query performance degradation is caused by two factors. The first factor is the increase in the number of MBR spatio-temporal objects that need to be parsed when splitting the trajectory into more segments. The second factor is that the Hilbert-Tree becomes more complex as the size of the MBR scales up, which increases

the construction and retrieval time of the indexes and makes the query statements very large and complex, and their processing cost increases dramatically, resulting in a longer execution cycle for a complete spatio-temporal query process. Additionally, as shown in the experimental results in Section 4.2.1, larger trajectory segment lengths are less effective in segmentation when applied to short trajectories, with a certain degree of decrease in the blank space reduction effect, while the effect on long trajectories is not significant. Meanwhile, larger segment lengths result in more MBR objects that need to be organized and analyzed, exacerbating the complexity and time overhead of index filtering and exact querying. Moreover, as shown in Section 3.1.2, greedy segmentation does not guarantee to obtain the global optimal solution, so the maximum number of segments for a single trajectory can be flexibly selected according to the degree of knowledge of the dataset and the management requirements. Above all, to ensure efficient query processing and index update maintenance, the maximum number of trajectory segments is chosen to be 20 in the subsequent spatio-temporal query experiments and index scalability validation.

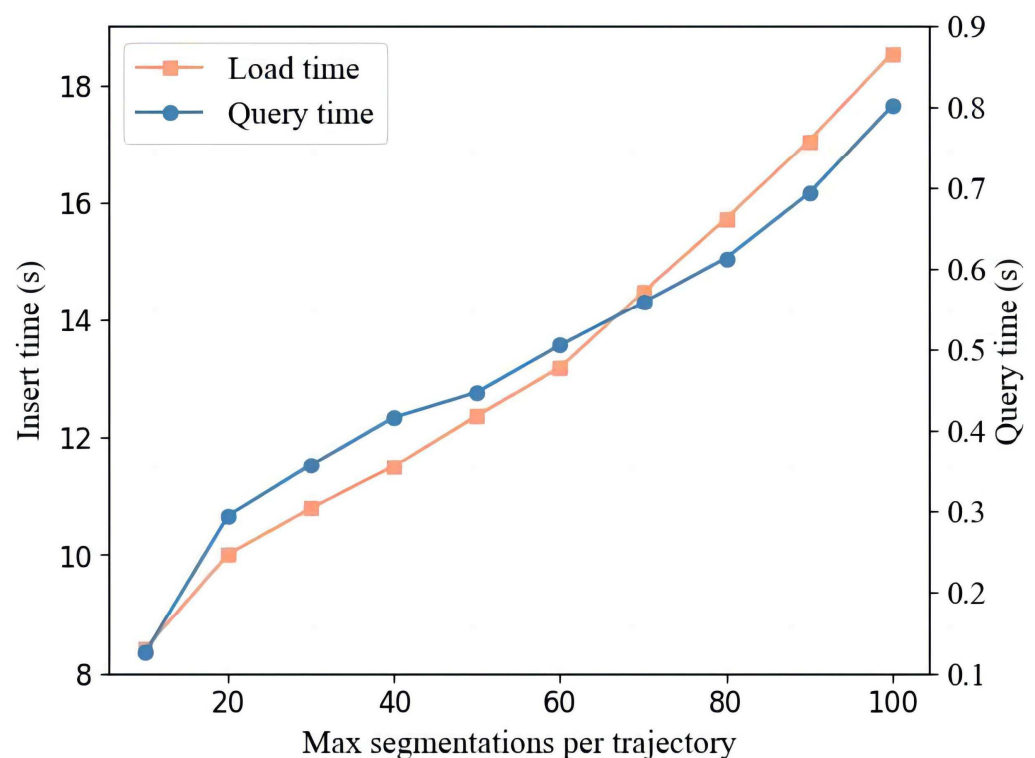


Figure 14. The query time for different segment lengths.

This paper examines trajectory-based spatio-temporal range queries. It attempts to find trajectory segments including certain activities within predefined spatio-temporal boundaries. For performing query efficiency metrics and testing the performance of indexed queries under different spatio-temporal dimension scales, activities using different spatio-temporal ranges are chosen to analyze the execution runtime; spatio-temporal query windows with ranges $0.1^\circ \times 0.1^\circ$, $0.2^\circ \times 0.2^\circ$, $0.5^\circ \times 0.5^\circ$, $1^\circ \times 1^\circ$, $1.5^\circ \times 1.5^\circ$, and $2^\circ \times 2^\circ$ are selected and denoted as query ranges 1, 2, 3, 4, 5, and 6, respectively; and querying experiments are conducted at four different time intervals. The number of experiments is 50. The query time is recorded as the average time of 50 queries, including two parts: (1) index filtering time and (2) precision screening refinement time. In addition, we compare this with three other common methods of composite indexing, which are implemented through two independent indexes. One method, the R-Tree index (a built-in spatial index in MySQL), is based on the spatial location information of trajectory segment and the B⁺-Tree index on its start and end time, which we denote as method 2. The other approach is achieved by 2D location index (a built-in spatial index in MongoDB) on the trajectory

segment with a B⁺-Tree index on the start and end time, which we denote as method 3. For the last method, we choose to encode the trajectory points with the utilized Hilbert curves [48], and subsequently construct a B⁺-Tree in the encoded field, which is similar to the Z3 indexing in GeoMesa, and we identify it as method 4. Moreover, the method proposed in this paper is denoted as method 1.

Figure 15 illustrates the query performance based on spatio-temporal thresholds. The performance of the HHBITS index is much better in terms of effectiveness test results when performing trajectory query processing. The proposed index saves about 40% time on average compared to three other spatio-temporal composite indexes for trajectory data.

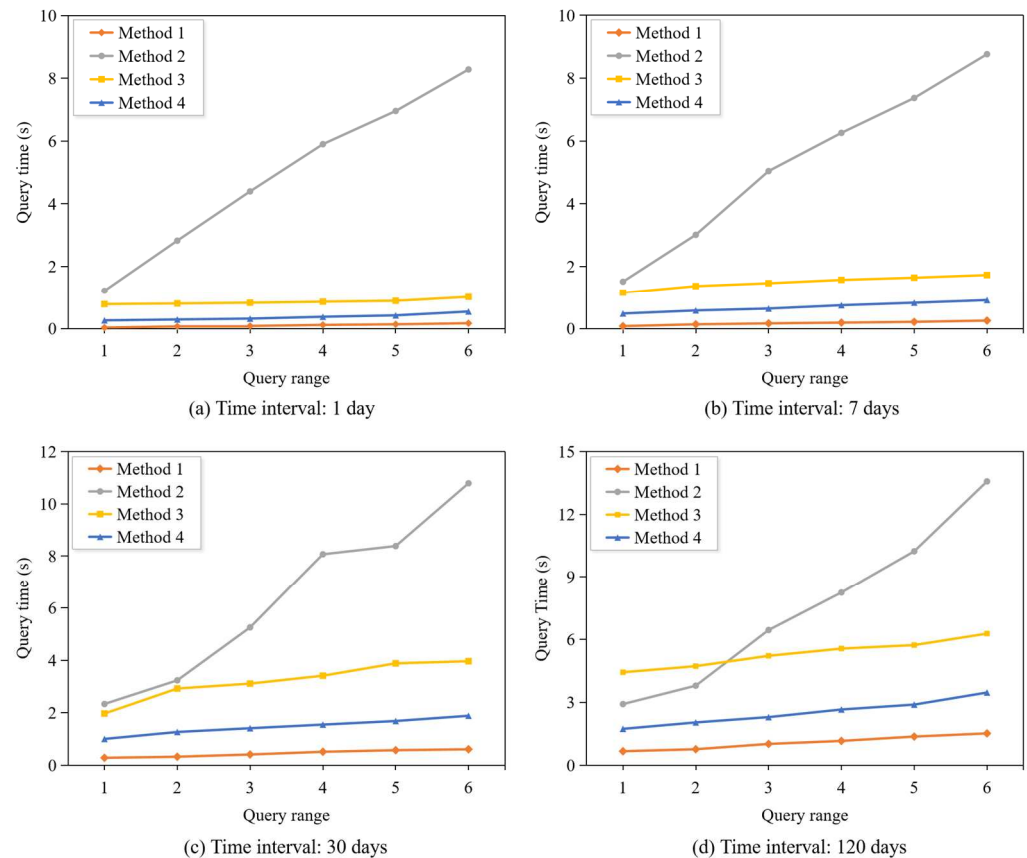


Figure 15. The comparison of the spatio-temporal query performance of methods under different spatio-temporal ranges: (a) time interval: 1 day; (b) time interval: 7 days; (c) time interval: 30 days; (d) time interval: 120 days.

According to Figure 15, the features and advantages of this paper's method in trajectory spatio-temporal querying can be observed. On the one hand, the running time of the query becomes longer with the increase in the spatio-temporal range and data size, because it needs to access a large number of partitions storing MBRs to obtain all the trajectory segments that satisfy the query conditions, but it always maintains a better performance in spatio-temporal querying with good scalability. Second, when a user accesses a small range of trajectories, the search is pruned according to the spatio-temporal boundaries and fewer trajectory partitions are accessed. If the desired trajectory segments are located in the same partition, the query is more effective. On the other hand, the increase in query range has less effect on the query efficiency of the HHBITS index and more effect on the composite index. This phenomenon suggests that for method 2 and method 3, which are regular types of composite index structures, they can only filter data in one dimension (spatial or temporal) at a time, while data in other dimensions can only be filtered later. With the expansion of the query area, a large number of records must still be filtered after the first filtering of the main index; for method 4, it can filter latitude, longitude, and

time at the same time, which reduces the number of three-dimensional queries in one dimension. However, due to the limited precision of space-filling curves, the method with trajectory points as index units still needs a large amount of time for traversal refinement to obtain accurate query results after the index coarse filtering. Although the HHBITS index is encoded separately in the temporal and spatial domains, it is compatible with the database's built-in compound index structure, providing better support for trajectory queries than existing methods. The HHBITS index is particularly adept at dealing with a large range of spatial-temporal queries for trajectories.

4.2.3. Index Scalability Validation

To verify the flexibility and scalability of the index, we chose to complete the construction of the index in three datasets of different sizes, and examined the time consumption of the method in this paper for index construction, updating, and maintenance in the same study area and different data densities. The dataset overview and experimental results are shown in Table 1. Among them, the time spans of the D1, D2, and D3 datasets are from April 2007 to December 2009, April 2007 to October 2010, and April 2007 to August 2012, respectively. The sample data were preloaded into the memory and waited for processing, and the time overhead covers the spatio-temporal encoding computation of segments of trajectories and the overhead of the persistence of B⁺-Tree indexes in MongoDB. In addition, we chose to complete the index construction supported by the D1, D2, and D3 datasets sequentially by incremental building, and the time difference between the completion of index construction of the two neighboring datasets is the time overhead of performing index updates.

Table 1. Description of experimental datasets.

Name	Number of Trajectories	Number of Trajectory Points	Index Construction Time (s)
D1	3542	5,096,851	0.773
D2	9092	13,321,696	2.203
D3	18,670	24,876,978	4.234

The experimental results show that the HHBITS index has excellent performance when facing datasets of different sizes. When the data density of the research area changes dramatically, the update and maintenance of the index can also be completed in a short time, showing good flexibility and scalability.

5. Conclusions and Discussion

With the aim of efficiently processing massive GPS trajectory datasets that contain spatio-temporal information and are unevenly distributed, this paper explores how to provide efficient trajectory data indexing and query support. To this end, this paper proposes a hybrid spatio-temporal indexing structure based on trajectory-optimized segmentation and uneven space division. Initially, the greedy algorithm is used to realize the optimized segmentation of trajectories to reduce the blank space and improve the space utilization of the index and the accuracy of the query. Subsequently, by combining temporal Hash, Hilbert coding under uneven spatial segmentation, and the database's built-in compound index, this paper applies this indexing model to the non-relational database MongoDB. The advantages of this index are that it can adapt to the spatial and temporal distribution of data, is easily integrated with existing database systems, and provides efficient range querying capabilities through standard SQL. In addition, the cost of index updates is low. For data update, it is only necessary to add or delete data table records. Through experimental comparisons, the HHBITS index proposed in this paper significantly outperforms the composite spatio-temporal index based on R-Tree and B⁺-Tree for range queries, especially when the query has a large spatio-temporal boundary.

In future research, we plan to conduct a more in-depth study of the greedy segmentation algorithm to explore the optimal solution computation method of segment length that can be applied to various types of trajectory data. We also plan to optimize the greedy segmentation algorithm so that it can automatically identify trajectories of different lengths for classification and select different segment lengths for different types of trajectories. In addition, we plan to perform parallel construction and real-time maintenance of indexes in a distributed environment and integrate with distributed NoSQL to provide scalable, high-performance, and highly fault-tolerant trajectory indexing and storage strategies. More comparisons with existing solutions such as HadoopTrajectory and GeoMesa will be made to better evaluate the performance of the proposed solution.

Author Contributions: Yuqi Yang conceived, designed, and performed the experiments and wrote the manuscript; Xiaoqing Zuo and Kang Zhao supervised the study; and Yongfa Li offered helpful suggestions and reviewed the manuscript. All authors have read and agreed to the published version of the manuscript.

Funding: The work was supported by the National Natural Science Foundation of China (No. 42161067), Yunnan Province Technical Innovation Talent Development Projects (No. 202405AD350058) and Major Science and Technology Projects of Yunnan Province (No. 202202AD080010).

Data Availability Statement: The data that support the findings of this study are openly available at <https://www.microsoft.com/en-us/research/publication/geolife-gps-trajectory-dataset-user-guide/>.

Acknowledgments: We would like to thank the reviewers for their in-depth suggestions and corrections that helped improve the quality of the paper.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Gao, Q.; Zhang, F.; Wang, R.; Zhou, F. Trajectory Big Data: A Review of Key Technologies in Data Processing. *J. Softw.* **2017**, *28*, 959–992.
2. Li, J.; Liu, J.; Zhao, X.; Huang, Q.; Sun, W.; Xu, Z.; Wang, H. Trajectory Data Management and Analysis Framework Based on Geographical Grid Model: Method and Application. *Geomat. Inf. Sci. Wuhan Univ.* **2021**, *46*, 640–649.
3. Zhao, L.; Mao, J.; Pu, M.; Liu, G.; Jin, C.; Qian, W.; Zhou, A.; Wen, X.; Hu, R.; Chai, H. Automatic Calibration of Road Intersection Topology Using Trajectories. In Proceedings of the 2020 IEEE 36th International Conference on Data Engineering (ICDE), Dallas, TX, USA, 20–24 April 2020; pp. 1633–1644.
4. Zheng, Y. Trajectory Data Mining: An Overview. *ACM Trans. Intell. Syst. Technol.* **2015**, *6*, 29:1–29:41. [\[CrossRef\]](#)
5. Wang, S.; Bao, Z.; Culpepper, J.S.; Cong, G. A Survey on Trajectory Data Management, Analytics, and Learning. *ACM Comput. Surv.* **2021**, *54*, 39:1–39:36. [\[CrossRef\]](#)
6. Yu, L.; Xiang, L.; Sun, S.; Guan, X.; Wu, H. kNN Query Processing for Trajectory Big Data Based on Distributed Column-Oriented Storage. *Geomat. Inf. Sci. Wuhan Univ.* **2021**, *46*, 736–745.
7. Luo, Y.; Chen, B. Adaptive data model and index structure for network- constrained trajectories. *J. Geo-Inf. Sci.* **2023**, *25*, 63–76.
8. Guttman, A. R-Trees: A Dynamic Index Structure for Spatial Searching. In Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data, Boston, MA, USA, 18–21 June 1984; Association for Computing Machinery: New York, NY, USA, 1984; pp. 47–57.
9. Xu, T.; Zhang, X.; Claramunt, C.; Li, X. TripCube: A Trip-Oriented Vehicle Trajectory Data Indexing Structure. *Comput. Environ. Urban Syst.* **2018**, *67*, 21–28. [\[CrossRef\]](#)
10. Aydin, B.; Akkineni, V.; Angryk, R.A. Modeling and Indexing Spatiotemporal Trajectory Data in Non-Relational Databases. In *Managing Big Data in Cloud Computing Environments*; IGI Global: Hershey, PA, USA, 2016; pp. 133–162. ISBN 978-1-4666-9834-5.
11. Li, G.; Tang, J. A New R-Tree Spatial Index Based on Space Grid Coordinate Division. In Proceedings of the International Conference on Informatics, Cybernetics, and Computer Engineering (ICCE2011), Melbourne, Australia, 19–20 November 2011; Jiang, L., Ed.; Springer: Berlin/Heidelberg, Germany, 2012; pp. 133–140.
12. Peng, Z.; Feng, J.; Wang, Q.; Xiong, W. A moving object indexing method that supports frequent location updating. *J. Geo-Inf. Sci.* **2017**, *19*, 152–160.
13. Gong, J.; Ke, S.; Zhu, Q.; Zhang, Y. An Efficient Trajectory Data Index Integrating R-tree, Hash and B*-tree. *Acta Geod. Cartogr. Sin.* **2015**, *44*, 570–577.
14. Qian, C.; Yi, C.; Cheng, C.; Pu, G.; Wei, X.; Zhang, H. GeoSOT-Based Spatiotemporal Index of Massive Trajectory Data. *ISPRS Int. J. Geo-Inf.* **2019**, *8*, 284. [\[CrossRef\]](#)
15. Wang, H.; Belhassena, A. Parallel Trajectory Search Based on Distributed Index. *Inf. Sci.* **2017**, *388–389*, 62–83. [\[CrossRef\]](#)

16. Kang, H.; Liu, Y.; Zhang, W. Cloud-Based Framework for Spatio-Temporal Trajectory Data Segmentation and Query. *IEEE Trans. Cloud Comput.* **2022**, *10*, 258–275. [CrossRef]
17. Xiang, L.; Wang, D.; Gong, Y. Organization and Efficient Range Query of Large Trajectory Data Based on Geohash. *Geomat. Inf. Sci. Wuhan Univ.* **2017**, *42*, 21–27.
18. Xiang, L.; Gao, M.; Wang, D.; Gong, Y. Geohash-Trees: An Adaptive Index Which can Organize Large-Scale Trajectories. *Geomat. Inf. Sci. Wuhan Univ.* **2019**, *44*, 436–442.
19. Guan, X.; Bo, C.; Li, Z.; Yu, Y. ST-Hash: An Efficient Spatiotemporal Index for Massive Trajectory Data in a NoSQL Database. In Proceedings of the 2017 25th International Conference on Geoinformatics, Redondo Beach, CA, USA, 2–4 August 2017; pp. 1–7.
20. Liu, H.; Yan, J.; Wang, J.; Chen, B.; Chen, M.; Huang, X. HGST: A Hilbert-GeoSOT Spatio-Temporal Meshing and Coding Method for Efficient Spatio-Temporal Range Query on Massive Trajectory Data. *ISPRS Int. J. Geo-Inf.* **2023**, *12*, 113. [CrossRef]
21. Yang, S.; He, Z.; Chen, Y.-P.P. GCOTraj: A Storage Approach for Historical Trajectory Data Sets Using Grid Cells Ordering. *Inf. Sci.* **2018**, *459*, 1–19. [CrossRef]
22. Pelekis, N.; Frentzos, E.; Giatrakos, N.; Theodoridis, Y. HERMES: A Trajectory DB Engine for Mobility-Centric Applications. *IJKBO* **2015**, *5*, 19–41. [CrossRef]
23. Zimányi, E.; Sakr, M.; Lesuisse, A.; Bakli, M. MobilityDB: A Mainstream Moving Object Database System. In Proceedings of the 16th International Symposium on Spatial and Temporal Databases, Vienna, Austria, 19–21 August 2019; Association for Computing Machinery: New York, NY, USA, 2019; pp. 206–209.
24. Cudre-Mauroux, P.; Wu, E.; Madden, S. TrajStore: An Adaptive Storage System for Very Large Trajectory Data Sets. In Proceedings of the 2010 IEEE 26th International Conference on Data Engineering (ICDE 2010), Long Beach, CA, USA, 1–6 March 2010; pp. 109–120.
25. Zheng, B.; Wang, H.; Zheng, K.; Su, H.; Liu, K.; Shang, S. SharkDB: An in-Memory Column-Oriented Storage for Trajectory Analysis. *World Wide Web* **2018**, *21*, 455–485. [CrossRef]
26. Mei, S.; Guan, H.; Wang, Q. An Overview on the Convergence of High Performance Computing and Big Data Processing. In Proceedings of the 2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS), Singapore, 11–13 December 2018; pp. 1046–1051.
27. Xiong, S.; Ouyang, X.; Xiong, W. Distributed or Centralized: An Experimental Study on Spatial Database Systems for Processing Big Trajectory Data. In Proceedings of the 2023 IEEE 8th International Conference on Big Data Analytics (ICBDA), Harbin, China, 3–5 March 2023; pp. 8–13.
28. Bakli, M.; Sakr, M.; Soliman, T.H.A. HadoopTrajectory: A Hadoop Spatiotemporal Data Processing Extension. *J. Geogr. Syst.* **2019**, *21*, 211–235. [CrossRef]
29. Qin, J.; Ma, L.; Niu, J. THBase: A Coprocessor-Based Scheme for Big Trajectory Data Management. *Future Internet* **2019**, *11*, 10. [CrossRef]
30. Qin, J.; Ma, L.; Liu, Q. DFTHR: A Distributed Framework for Trajectory Similarity Query Based on HBase and Redis. *Information* **2019**, *10*, 77. [CrossRef]
31. Li, R.; He, H.; Wang, R.; Ruan, S.; Sui, Y.; Bao, J.; Zheng, Y. TrajMesa: A Distributed NoSQL Storage Engine for Big Trajectory Data. In Proceedings of the 2020 IEEE 36th International Conference on Data Engineering (ICDE), Dallas, TX, USA, 20–24 April 2020; pp. 2002–2005.
32. Zhang, Z.; Jin, C.; Mao, J.; Yang, X.; Zhou, A. TrajSpark: A Scalable and Efficient In-Memory Management System for Big Trajectory Data. In Proceedings of the Web and Big Data, Beijing, China, 7–9 July 2017; Chen, L., Jensen, C.S., Shahabi, C., Yang, X., Lian, X., Eds.; Springer International Publishing: Cham, Switzerland, 2017; pp. 11–26.
33. Shang, Z.; Li, G.; Bao, Z. DITA: Distributed In-Memory Trajectory Analytics. In Proceedings of the 2018 International Conference on Management of Data, Houston, TX, USA, 10–15 June 2018; Association for Computing Machinery: New York, NY, USA, 2018; pp. 725–740.
34. Ding, X.; Chen, L.; Gao, Y.; Jensen, C.S.; Bao, H. UtraMan: A Unified Platform for Big Trajectory Data Management and Analytics. *Proc. VLDB Endow.* **2018**, *11*, 787–799. [CrossRef]
35. Jasinski, M. Datamining. Available online: <https://github.com/marciogi/datamining> (accessed on 8 August 2016).
36. Bao, Y.; Huang, Z.; Gong, X.; Zhang, Y.; Yin, G.; Wang, H. Optimizing Segmented Trajectory Data Storage with HBase for Improved Spatio-Temporal Query Efficiency. *Int. J. Digit. Earth* **2023**, *16*, 1124–1143. [CrossRef]
37. Hadjieleftheriou, M.; Kollios, G.; Tsotras, V.J.; Gunopulos, D. Efficient Indexing of Spatiotemporal Objects. In Proceedings of the Advances in Database Technology—EDBT, Prague, Czech Republic, 25–27 March 2002; Jensen, C.S., Šaltenis, S., Jeffery, K.G., Pokorný, J., Bertino, E., Böhm, K., Jarke, M., Eds.; Springer: Berlin/Heidelberg, Germany, 2002; pp. 251–268.
38. Rasetic, S.; Sander, J.; Elding, J.; Nascimento, M.A. A Trajectory Splitting Model for Efficient Spatio-Temporal Indexing. In Proceedings of the 31st VLDB Conference, Trondheim, Norway, 30 August–2 September 2005.
39. Cao, B.; Feng, H.; Liang, J.; Li, X. Hilbert Curve and Cassandra Based Indexing and Storing Approach for Large-Scale Spatiotemporal Data. *Geomat. Inf. Sci. Wuhan Univ.* **2021**, *46*, 620–629.
40. Gong, X.; Huang, Z.; Wang, Y.; Wu, L.; Liu, Y. High-Performance Spatiotemporal Trajectory Matching across Heterogeneous Data Sources. *Future Gener. Comput. Syst.* **2020**, *105*, 148–161. [CrossRef]
41. Kang, Y.; Gui, Z.; Ding, J.; Wu, J.; Wu, H. Parallel Ripleys’ K function based on Hilbert spatial partitioning and Geohash indexing. *J. Geo-Inf. Sci.* **2022**, *24*, 74–86.

42. Eldawy, A.; Alarabi, L.; Mokbel, M.F. Spatial Partitioning Techniques in SpatialHadoop. *Proc. VLDB Endow.* **2015**, *8*, 1602–1605. [[CrossRef](#)]
43. Yao, X.; Yang, J.; Li, L.; Ye, S.; Yun, W.; Zhu, D. Parallel Algorithm for Partitioning Massive Spatial Vector Data in Cloud Environment. *Geomat. Inf. Sci. Wuhan Univ.* **2018**, *43*, 1092–1097.
44. Zhao, X.; Huang, X.; Qiao, J.; Kang, R.; Li, N.; Wang, J. A Spatio-Temporal Index Based on Skew Spatial Coding and R-Tree. *J. Comput. Res. Dev.* **2019**, *56*, 666–676.
45. Aji, A.; Wang, F.; Vo, H.; Lee, R.; Liu, Q.; Zhang, X.; Saltz, J. Hadoop GIS: A High Performance Spatial Data Warehousing System over Mapreduce. *Proc. VLDB Endow.* **2013**, *6*, 1009–1020. [[CrossRef](#)]
46. Wang, J.; Shan, J. Space-Filling Curve Based Point Clouds Index. In Proceedings of the 8th International Conference on GeoComputation, Ann Arbor, MI, USA, 31 July–3 August 2005; pp. 551–562.
47. Zheng, Y.; Xie, X.; Ma, W.-Y. GeoLife: A Collaborative Social Networking Service among User, Location and Trajectory. *IEEE Data Eng. Bull.* **2010**, *33*, 32–39.
48. Wu, Y.; Cao, X.; An, Z. A Spatiotemporal Trajectory Data Index Based on the Hilbert Curve Code. *IOP Conf. Ser. Earth Environ. Sci.* **2020**, *502*, 012005. [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.