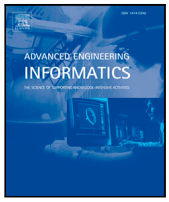
Advanced Engineering Informatics 57 (2023) 102064

Contents lists available at ScienceDirect

Advanced Engineering Informatics

journal homepage: www.elsevier.com/locate/aei



Full length article

Tinba: Incremental partitioning for efficient trajectory analytics

Ruijie Tian [a,1], Weishi Zhang [a,b,*], Fei Wang [a,*], Kemal Polat [c,1], Fayadh Alenezi [d]

[a] Information Science and Technology College, Dalian Maritime University, Dalian, Liaoning, China
[b] Key Laboratory of Intelligent Software, Dalian, Liaoning, China
[c] Department of Electrical and Electronics Engineering, Faculty of Engineering, Bolu Abant Izzet Baysal University, Bolu, Turkey
[d] Department of Electrical Engineering, College of Engineering, Jouf University, 72238, Saudi Arabia

ARTICLE INFO

Keywords:
Incremental partitioning
Big data
Trajectory similarity
DFS
Distributed computing
NP-hardness

ABSTRACT

Applications with mobile and sensing devices have already become ubiquitous. In most of these applications, trajectory data is continuously growing to huge volumes. Existing systems for big trajectory data organize trajectories at distributed block storage systems. Systems like DITA that use block storage (e.g., 128 MB each) are more efficient for analytical queries, but they cannot incrementally maintain the partitioned data and do not support delete operations, resulting in difficulties in trajectory analytics. In this paper, we propose an incremental trajectory partitioning framework Tinba that enables distributed block storage systems to efficiently maintain optimized partitions under incremental updates of trajectories. We employ a data flushing technique to bulk ingest trajectory data for random writing in distributed file system (DFS). We recast the incremental partitioning problem as an optimal partitioning problem and prove its NP-hardness. A cost–benefit model is proposed to address the optimal partitioning problem. Moreover, Tinba supports most of the existing similarity measures to quantify the similarity between trajectories. A heuristic is developed to instantiate the Tinba framework. Comprehensive experiments on real-world and synthetic datasets demonstrate the advancements in ingestion performance and partition quality, as opposed to other trajectory partition methods.

1. Introduction

The development of sensing devices and positioning technologies has enabled many applications from different domains to collect large amounts of trajectory data. For instance, Didi captures 106TB of trajectory data every day.[2] As of December 2021, Uber recorded 28.98 billion trips, of which 6.3 billion trips have been recorded in the year of 2021.[3] As a result, the trajectory data collected by these applications are not only characterized by large volumes, but also by constant growth and updates. Most importantly, these characteristics make trajectory analysis applications difficult.

Existing studies focus on optimizing this problem in two types of distributed systems, distributed databases and distributed block storage systems. In distributed databases, each record uniquely corresponds to a key in order to find its exact location [1], employing a log-structured merge tree (LSM-tree) to provide a low-cost index for a high rate of trajectory inserts (and deletes). In contrast, block storage system roughly divides the data into partitions [2–4] and ignores the exact location of the records. Given that the analytical query reads all the records in the partition to which it accesses. Therefore, the block storage system is more suitable for trajectory analytics tasks than NoSQL database.

Existing block storage systems focus on optimizing this problem for static data. However, batch ingestion of trajectory data exceeds the partition maintenance capability of block storage systems, because (1) *random write limitation problem of DFS*: it is challenging to break the sequential write limitation of DFS to support random writes of trajectory batch ingestion; (2) *incremental partition problem*: since batch ingestion produces a low quality partition set, it is challenging to design partition reorganization techniques to reduce the cost of all partition reorganization and trajectory analytics cost; (3) *versatility problem of similarity functions*: It is challenging that the designed data reorganization techniques supports various trajectory similarity functions to be applicable for different trajectory analytics tasks.

To compensate the gap between the limited incremental partition maintenance capability of distributed block storage systems and the urgency of practical applications for efficient trajectory analytics, we

* Corresponding authors.
E-mail addresses: teesiv@dlmu.edu.cn (W. Zhang), feiwang@dlmu.edu.cn (F. Wang), kpolat@ibu.edu.tr (K. Polat), fshenezi@ju.edu.sa (F. Alenezi).

[1] Ruijie Tian and Kemal Polat contributed equally to this article.

[2] https://sts.didiglobal.com

[3] https://expandedramblings.com/index.php/uber-statistics/

https://doi.org/10.1016/j.aei.2023.102064
Received 2 January 2023; Received in revised form 5 May 2023; Accepted 16 June 2023
Available online 27 June 2023

propose a Trajectory incremental big data analytics framework Tinba. We devise a data flushing technique for random writes on DFS. We recast the incremental partitioning problem as an optimal partitioning problem and prove its NP-hardness, which means that constructing optimal partitions is a difficult task. So we decompose the optimal partitioning problem into *partition selection* and *data reorganization* problem. Partition selection provides a set of candidate partitions to be reorganized, while data reorganization physically reorganizes the selected partitions. Since any static index of the trajectory data can be used to solve the data reorganization problem, this paper mainly focuses on the partition selection problem. The challenge is how to accurately estimate the similarity search cost and use that cost as a benchmark for partition quality. In light of this, we propose a cost model to estimate the processing cost of analytical queries, and a benefit model to estimate the benefit of data reorganization. We design a heuristic for instantiating Tinba and propose a greedy algorithm to select a set of most beneficial partitions for reorganization.

In summary, this paper makes the following contributions.

- We investigated partition problem of incremental trajectory and propose a framework Tinba, which can efficiently support incremental trajectory data with batch updates for trajectory analytics in block storage systems.
- We recasts the incremental partitioning problem as an optimal partitioning problem and proves its NP-hardness.
- We propose a cost–benefit model to select partitions for reorganization. The cost model is used to estimate the processing cost of the analytical query, and the benefit model to estimate the benefit after data reorganization.
- We devise three heuristics for instantiating Tinba and propose a greedy algorithm to select a subset of optimal partitions for reorganization.
- We conduct a comprehensive evaluation on real world and synthetic datasets. The results indicate that Tinba outperforms state-of-the-art big trajectory partition methods.

The remainder of the paper is organized as follows. Related work is discussed in Section 2. Section 3 recasts the incremental partitioning problem as an optimization problem and proves its NP-hardness. Section 4 provides an overview of the Tinba framework. Section 5 propose a cost–benefit model for optimal partitioning problem. Section 6 shows three implementations of the Tinba. Section 7 presents the results of our experimental study and Section 8 concludes the paper.

2. Related work

2.1. Trajectory analytics systems

Existing trajectory analytics systems can be divided into two categories, centralized systems and distributed systems. The distributed systems can be roughly categorized into distributed databases and block storage systems, as shown in Fig. 1(b) and 1(c).

2.1.1. Centralized systems

As shown in Fig. 1(a), MobilityDB [5], BerlinMOD [6] and TrajStore [7] design specific storage structures and indexes for traditional database engines. Torch [8] adopts a columnar data schema to provide better query and analytics performance. However, these systems are designed for centralized architectures, so they are inefficient or impossible to handle big trajectory data. Moreover, the flexibility of the system is limited because the storage structure is limited by the underlying database engine.

2.1.2. Distributed databases

As shown in Fig. 1(b), such systems are mainly focused on NoSQL databases (e.g., TrajMesa [1], ST-hash [9] and Trass [10]) and big data management systems (e.g., AsterixDB [11]). The advantage of this type of system is that the random write limitation of DFS is solved by converting random writes to sequential writes using LSM-Tree [12]. The disadvantage is that it is costly to process records, as they must be retrieved one by one by index. As a result, those systems are appropriate for selective queries that return a small number of records, but not for analytical queries that process the majority or all records.

2.1.3. Block storage systems

Fig. 1(c) shows systems in this category that create partitions directly on blocks of data in DFS. For example, Hadoop-based systems (e.g., Summit [13], CloST [4] and PARADASE [14]) and Spark-based systems (e.g., TrajSpark [2], DITA [3], DFT [15] and other [16–18]). Some of these systems, such as DITA [3], while considering data locality and workload, similarity analysis is optimized only for static trajectory datasets. In other words, whenever a new record is added or an old record is deleted, the entire data must be completely repartitioned for optimal performance. Several adaptive trajectory data partitioning techniques, e.g., TrajStore [18], Geohash-Tree [19] and others [20–22], have been developed to address this limitation. However, they are either limited to cloud-based platforms [20], database systems [22] or in-memory partition [21], neither of which addresses incremental trajectory partitioning over distributed block storage systems.

In summary, in order to enable big trajectory data to efficiently process analytical queries in the presence of a large number of insertions and deletions, this paper studies the incremental partitioning problem in distributed block storage system. Fig. 1(d) shows the proposed work to introduce incremental partitioners into block storage systems for incremental maintenance of DFS partitions. This work is similar to LSM-Tree in achieving high frequency updates on a constrained DFS compared to a distributed database.

2.2. Trajectory similarity measures

According to the evaluation of different similarity measures by existing studies [23,24], the most representative similarity functions are selected for analytics in this paper, including Fréchet distance [25], Dynamic Time Warping (DTW) [26], Edit Distance on Real Sequences (EDR) [27], and Longest Common Subsequence Distance (LCSS) [28]. Each distance function has distinct capabilities. This paper demonstrates the design and instantiation of the Tinba using the most popular similarity measure (Fréchet distance) for geometric curves and time series data. We also conduct a comprehensive experimental evaluation using DTW, EDR, and LCSS functions to demonstrate the versatility of the Tinba. We describe these functions in Appendix B.

2.3. Trajectory analytics

There have been several studies on trajectory analytics, including clustering [29], outlier identification [30], classification [31–33], pattern mining [34,35] and simplification [36,37]. Wang et al. [38] provides a comprehensive review of related work on trajectory data management and analytics. In this paper, we focus on the most currently popular trajectory analytics task, i.e., trajectory similarity searches [3,10,15,18].

3. Problem statement

In this section, firstly, some basic definitions are presented. Secondly, this paper recasts the incremental partitioning problem as an optimal partitioning problem and proves its NP-hardness. Table 1 lists the frequently notations used in this paper.

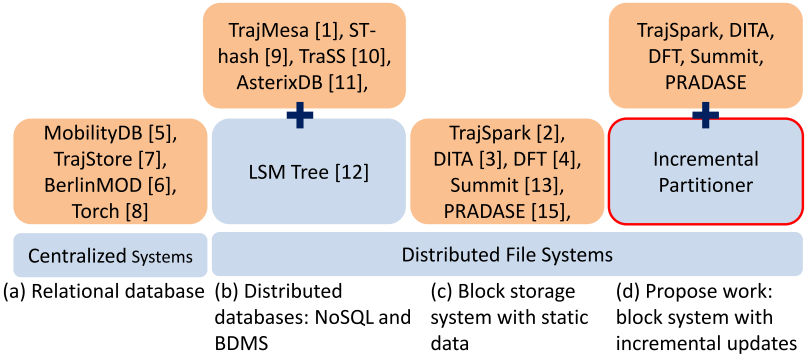


Fig. 1. Architecture of trajectory analytics processing methods.

Table 1
Frequently used notations.

Notation	Description
b	Default block size, e.g., 128 MB
$MBR(D)$	Minimum MBR: $\cup_{i=1}^{n} mbr(r_i)$
$asize(D)$	The actual size of D is the sum of all trajectory sizes: $\sum_{i=1}^{n} size(r_i)$. This represents the disk space required to store these trajectories.
$ablocks(D)$	The number of actual blocks of D: $\lceil \frac{asize(D)}{b} \rceil$
$csize(D)$	The compact size of D is the size that this set will occupy if deleted trajectories are removed: $\sum_{i=1}^{n} (1 - 2 \cdot is\_d_i) \cdot size(r_i)$.
$cblocks(D)$	The number of compact blocks occupied by D if compacted: $\lceil \frac{csize(D)}{b} \rceil$
$\textsc{MinDist}(Q, p)$	The minimal distance from trajectory Q to partition p

3.1. Preliminary

Definition 1 (*Trajectory*)**.** A trajectory is a sequence of points, denoted as $\mathcal{T} = \{t_1, \ldots, t_m, MBR, size, is\_d\}$, where MBR and size represents minimum bounding rectangle and size in bytes, respectively. For simplicity we assume each point t_i is represented as a 2-dimensional tuple ($latitude$, $longitude$). In addition, is_d is a identifier that indicates whether the trajectory is deleted.

Definition 2 (*Partitioning*)**.** Given a dataset D, a global partitioning $\mathcal{P}$ is a collection of subset of D:

$$\mathcal{P} = \{p_1, \ldots, p_m\}$$
$$s.t.\ \forall\ p_i \subseteq D,\ p_i \cap p_j = \varnothing (i \neq j)\ and\ \cup_i p_i = D \tag{1}$$

In this paper, we assume that the partitions are non-overlapping with each other. Although it is difficult for trajectory data, this assumption is orthogonal to the optimal problem studied in this paper.

According to the statistics of the literature on trajectory similarity analytics in the last five years [23,24], Fréchet distance [25] is the most widely used one. Therefore, Fréchet distance is used as the default similarity metric. In addition, this paper also shows how to support other similarity measures in Appendix B.

Definition 3 (*Fréchet Distance*)**.** Given two trajectories $\mathcal{T} = \{t_1, \ldots, t_m\}$ and $\mathcal{Q} = \{q_1, \ldots, q_n\}$, Fréchet distance is computed as below.

$$F(\mathcal{T}, \mathcal{Q}) = \begin{cases} \max\limits_{1 \le i \le m} \textsc{Dist}(t_i, q_1) & if\ n = 1 \\ \max\limits_{1 \le j \le n} \textsc{Dist}(t_1, q_j) & if\ m = 1 \\ \max\left\{ \textsc{Dist}(t_m, q_n), \min\left\{ F(\mathcal{T}^{m-1}, \mathcal{Q}), F(\mathcal{T}^{m-1}, \mathcal{Q}^{n-1}), F(\mathcal{T}, \mathcal{Q}^{n-1}) \right\} \right\} & others \end{cases} \tag{2}$$

where $\mathcal{T}^{m-1} = \{t_1, \ldots, t_{m-1}\}$ and $\mathcal{Q}^{n-1} = \{q_1, \ldots, q_{n-1}\}$ are the sub-trajectories of trajectories $\mathcal{T}$ and $\mathcal{Q}$ respectively, and $\textsc{Dist}(t_i, q_j)$ is the Euclidean distance of two GPS points t_i and q_j.

Definition 4 (*Trajectory Similarity*)**.** Given two trajectories $\mathcal{T}$ and $\mathcal{Q}$, a trajectory distance function f(e.g., Fréchet distance) and a threshold ε, if $f(\mathcal{T}, \mathcal{Q}) \le \varepsilon$, $\mathcal{T}$ and $\mathcal{Q}$ is similar.

Definition 5 (*Edit Cost*)**.** Given two partitions $\mathcal{P}$ and $\mathcal{P}'$, the edit cost is defined as the sum of the number of actual blocks read and the number of compact blocks written when adding, deleting, or replacing $\mathcal{P}$ to make it into $\mathcal{P}'$.

$$EC(\mathcal{P}, \mathcal{P}') = \sum_{p_i \in \mathcal{P} \setminus \mathcal{P}'} \left(ablocks(p_i) + cblocks(p_i) \right) \tag{3}$$

where $ablocks(p_i) = \lceil \frac{asize(D)}{b} \rceil$ is the number of actual blocks of D, $cblocks(p_i) = \lceil \frac{csize(D)}{b} \rceil$ is the number of compact blocks occupied by D if compacted.

Definition 6 (*Search Cost*)**.** Give a partition $\mathcal{P}$ and a query $\mathcal{Q}$, similarity search cost is formally defined as follows.

$$QC(\mathcal{P}, \mathcal{Q}) = \sum_{p_i \in \mathcal{P}} QC(p_i, \mathcal{Q}) \tag{4}$$

where $QC(p_i, \mathcal{Q})$ is a cost function for one partition p_i, formalized in Section 5.1. $QC(\mathcal{P}, \mathcal{Q})$ measures the average cost of running a similarity search with $\mathcal{Q}$ on $\mathcal{P}$, reflecting the effect of partitioning on search processing.

3.2. Problem formulation

The key to incremental partitioning is to use the existing partition $\mathcal{P}$ of the dataset D to find the optimal partition $\mathcal{P}'$ for the changed dataset D' without starting from scratch [39]. Thus, we recast the incremental partitioning problem as an optimal partitioning problem to be solved.

Definition 7 (*Optimal Partitioning Problem*)**.** Given a partitioning state $\mathcal{P}$, a read/write blocks threshold $\mathcal{B}$ and a query trajectory $\mathcal{Q}$. The optimal partitioning problem is to find a new partition $\mathcal{P}'$ by editing $\mathcal{P}$ that satisfy the following conditions:

(1) the edit cost of $\mathcal{P}$ and $\mathcal{P}'$ is not greater than $\mathcal{B}$, and
(2) $QC(\mathcal{P}', \mathcal{Q})$ is minimized.

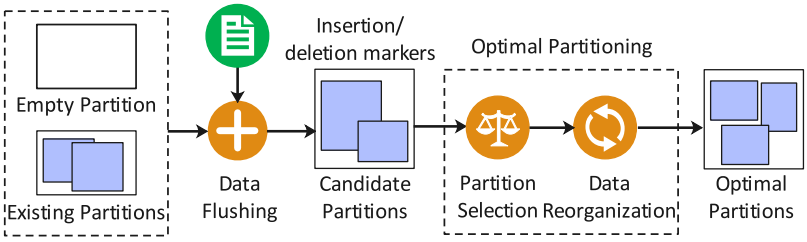


Fig. 2. An overview of Tinba framework.

Before analyzing and proposing solutions, the following conclusions are given.

Theorem 1 (*Complexity of Optimal Partitioning*). *Optimal partitioning problem is NP-hard. (see the proof in Appendix A.1)*

4. Overview of the Tinba framework

In this section, we firstly introduce the partition dataframe on a DFS, and then discuss each step of the Tinba, including data flushing, partition selection and data reorganization, as shown in Fig. 2.

Partition DataFrame. In terms of distributed block storage systems, each partition is persisted on disk as a separate file [3,13–15]. The metadata of the partition is stored in DFS as a master file. In other words, the master file is also a global index. In the trajectory analytics process, the master file is initially examined to determine the partitions to be accessed. The selected partitions are then processed in parallel using a distributed computing engine.

Data Flushing. The data flushing consists of two operations: buffer flushing and persistence flushing. Usually, in distributed systems, buffer flushing ingests a new batch into the memory buffer and when the components in the buffer reach a specified threshold, a persistent flushing operation is triggered to store the data sequentially to disk.

Partition Selection. The partition selection step is used to select the partitions that need to be reorganized from the set of candidate partitions. System constraints that limit the amount of disk I/O in this process were taken into account while designing this step. The purpose is to select some low quality partitions (e.g., overlapping partitions) for reorganization to improve partition quality. Since this step uses the master file to select partitions, it imposes a negligible overhead.

Data Reorganization. The data reorganization step achieves optimal partition processing by physically rewriting the selected partitions into optimal partitions and removing deleted records. This step is implemented by the existing static trajectory partitioning algorithm. Since a physical rewrite completely scans the partition, the cost of data reorganization is linearly related to the total size of the selected partitions.

5. Cost-benefit model for partition selection

As shown in Fig. 2, we provide an overview of framework, including data flushing and optimal partitioning. Given that optimal partitioning problem is NP-hard (Section 3.2), this paper focuses on its sub-problem, partition selection question, presents a theoretical demonstration, and designs a cost–benefit model.

5.1. Cost model for similarity search

Previous studies have shown that in analysis tasks, the cost of similarity search is a benchmark for the trajectory partitioning quality [1,5,7,17].

Existing query cost models primarily consider the number of disk pages. These models make sense for a traditional DBMS that accesses data from disk because the disk pages are small (e.g., 4 KB). Assuming that the smallest unit of access to disk is the disk page, then the cost of all disk pages can be treated as uniform. When dealing with DFS, this assumption no longer holds. In DFS, data is stored in blocks ranging in size from a few megabytes to 128 MB. As a result, the cost of access is no longer uniform for each block.

Our analysis process begins with a similarity search and attempts to estimate the cost. We suppose $\mathcal{P} = \{p_1, \ldots, p_m\}$ is the partition state of the dataset D, and $w(p_i)$, $h(p_i)$ are the width and height of the minimum bound rectangle (MBR) of partition p_i, respectively. There are two possible relations between search Q and partition p_i, overlapping and disjointing. As shown in Fig. 3, the buffer region of Q (the blue part) is disjoint to partition p_2, which means that searching for Q does not require processing p_2. Instead, Q overlaps with p_1, so the partition p_1 needs to be processed to return the result. To determine the relationship between the query Q and the partition p_i, we computed the probability that Q intersects p_i. For this purpose, this paper defines a buffer B_i for the partition p_i, which extends in all directions with a buffer size of ϵ. It can be observed that Q overlaps a partition (e.g., p_1) if the minimum distance between Q and buffer region (e.g., B_1) is not greater than ϵ; otherwise, it is disjoint. The probability that the query Q intersects the partition p_i is the ratio of the area of B_i to the area of $MBR(D)$ (cf. Appendix A.2).

Theorem 2. *The probability of intersection of partition p_i and query Q is equal to the ratio of the area of buffer region B_i to the area of data domain of $MBR(D)$. (see the proof in Appendix A.2)*

Thus, the average number of blocks provided by partition p_i to Q is:

$$LC_b(p_i, Q) = \frac{\left(w(p_i) + 2\epsilon\right)\left(h(p_i) + 2\epsilon\right)}{w(P)h(P)} \cdot ablocks(p_i) \tag{5}$$

the average size of data contributed by partition p_i to Q is:

$$SC_s(p_i, Q) = \frac{\left(w(p_i) + 2\epsilon\right)\left(h(p_i) + 2\epsilon\right)}{w(P)h(P)} \cdot asize(p_i) \tag{6}$$

Next, we analyze the cost of processing a search in HDFS. First, the cost of locating overlapping blocks is fixed, which is related to the number of partitioned blocks. Second, the cost of scanning the whole partition depends on the size of the partitioning. Consequently, the total cost (running time) to finish a similarity search can be expressed as a linear combination of $LC_b(p_i, Q)$ and $SC_s(p_i, Q)$ as follows:

$$QC(p_i, Q) = k_b \cdot LC_b(p_i, Q) + k_s \cdot SC_s(p_i, Q) \tag{7}$$

where k_b and k_s are hardware-specific coefficients, respectively.

According to Eq. (4), the cost of answering the query Q_i on the partitions of $\mathcal{P}$ is as follows:

$$QC(\mathcal{P}, Q) = k_b \cdot \sum_{p_i \in \mathcal{P}} LC_b(p_i, Q) + k_s \cdot \sum_{p_i \in \mathcal{P}} SC_s(p_i, Q) \tag{8}$$

5.2. Benefit model for data reorganization

This section presents a *benefit model* which utilizes the cost model to compute the benefits of data reorganization, i.e., the cost of similarity search after data reorganization subtracted the cost of similarity search

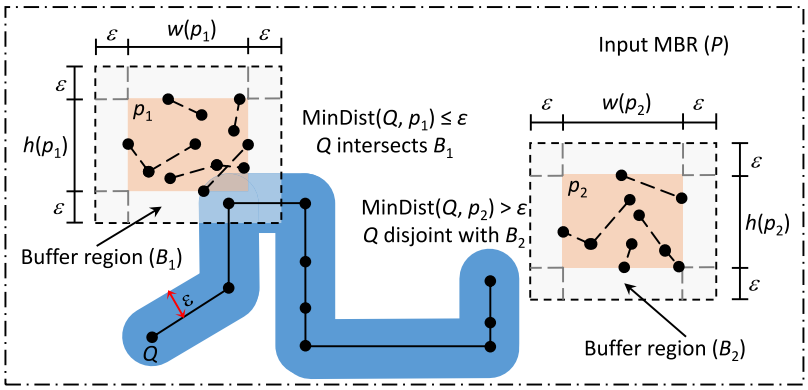


Fig. 3. Relationship of a similarity search with partitions.

before reorganization. To formalize the benefits of reorganization, the partitioning state at t moments is denoted $\mathcal{P}_t$ and the partitioning state after reorganization is denoted as $\mathcal{P}_{t+1}$. We suppose that a set of low quality partitions $\mathcal{G}_t = \{p_1, \ldots, p_n\} \subseteq \mathcal{P}_t$ is selected for reorganization and a set of new partitions $\mathcal{G}_{t+1} = \{p'_1, \ldots, p'_m\} \subseteq \mathcal{P}_{t+1}$ is generated. The reorganization benefit of $\mathcal{G}_t$ is defined as the search cost saved when the partition is reorganized from $\mathcal{P}_t$ to $\mathcal{P}_{t+1}$.

$$benefit(\mathcal{G}_t, \mathcal{Q}) = QC(\mathcal{P}_t, \mathcal{Q}) - QC(\mathcal{P}_{t+1}, \mathcal{Q}) \tag{9}$$

The goal of optimal partitioning problem is to find the minimum $QC(\mathcal{P}_{t+1}, \mathcal{Q})$, so the problem turns into finding the maximum of $benefit(\mathcal{G}_t, \mathcal{Q})$.

We can rewrite $\mathcal{P}_t$ as $(\mathcal{P}_t - \mathcal{G}_t) \cup \mathcal{G}_t$, $\mathcal{P}_{t+1}$ is similar.

$$\begin{aligned} benefit(\mathcal{G}_t, \mathcal{Q}) =& QC((\mathcal{P}_t - \mathcal{G}_t) \cup \mathcal{G}_t, \mathcal{Q}) - QC((\\ & \mathcal{P}_{t+1} - \mathcal{G}_{t+1}) \cup \mathcal{G}_{t+1}, \mathcal{Q}) \end{aligned} \tag{10}$$

Since the cost model is linear, we convert Eq. (10) using the superposition method as follows.

$$\begin{aligned} benefit(\mathcal{G}_t, \mathcal{Q}) =& QC(\mathcal{G}_t, \mathcal{Q}) - QC(\mathcal{G}_{t+1}, \mathcal{Q}) + QC(\\ & \mathcal{P}_t - \mathcal{G}_t, \mathcal{Q}) - QC(\mathcal{P}_{t+1} - \mathcal{G}_{t+1}, \mathcal{Q}) \end{aligned} \tag{11}$$

But $\mathcal{P}_t - \mathcal{G}_t \equiv \mathcal{P}_{t+1} - \mathcal{G}_{t+1}$, which are sets of unselected partitions. They cost the same, which means the benefit of reorganization $\mathcal{G}_t$ is:

$$benefit(\mathcal{G}_t, \mathcal{Q}) = QC(\mathcal{G}_t, \mathcal{Q}) - QC(\mathcal{G}_{t+1}, \mathcal{Q}) \tag{12}$$

To better understand the benefit model, Fig. 4 gives three scenarios for reorganizing partitions. For simplicity, we assume that $k_b = 1$, $k_s = 0$, the area of $MBR(\mathcal{D})$ is $w(\mathcal{P}) \cdot h(\mathcal{P}) = 10$ and distance threshold $\varepsilon = 0.1$.

The first scenario as shown in Fig. 4(a), partition p_1 with 4 blocks is reorganized into 4 partitions with one block. According to Eq. (5), $LC(p_1, \mathcal{Q}) = \frac{(2+2\times0.1)\cdot(2+2\times0.1)}{10} \cdot 4 = 1.936$ and $\sum_{i=2}^{5} LC(p_i, \mathcal{Q}) = 4 \cdot \frac{(1+2\times0.1)\cdot(1+2\times0.1)}{10} \cdot 1 = 0.576$. Therefore, the cost is reduced by $1.936 - 0.144 = 1.36$. This shows that by dividing p_1 into four smaller single partitions, an average of 1.36 fewer blocks will need to be accessed during processing the query $\mathcal{Q}$. This scenario suggests that splitting a large partition containing multiple blocks into multiple smaller partitions containing one block can significantly reduce the search cost (i.e., the number of block accesses).

The second scenario as shown in Fig. 4(b), partition p_1 with *empty area* is split into partition p_2, p_3 with one block that does not cover *empty area*. The costs before and after the reorganization are $LC(p_1, \mathcal{Q}) = \frac{(2+2\times0.1)\cdot(2+2\times0.1)}{10} \cdot 2 = 0.968$ and $LC(p_2, \mathcal{Q}) + LC(p_3, \mathcal{Q}) = 2 \cdot \frac{(1+2\times0.1)\cdot(1+2\times0.1)}{10} \cdot 1 = 0.288$, respectively. The cost is reduced by 0.68. This scenario suggests that reorganizing partitions containing *empty area* into partitions that do not cover *empty areas* can reduce search costs.

Fig. 4(c) shows the scenario of splitting two overlapping partitions. We split the two overlapping partitions into six disjoint partitions. The cost before and after reorganization are $LC(p_1, \mathcal{Q}) + LC(p_2, \mathcal{Q}) = 2 \cdot \frac{(2+2\times0.1)\cdot(2+2\times0.1)}{10} \cdot 3 = 2.904$ and $\sum_{i=3}^{8} LC(p_i, q) = 6 \cdot \frac{(1+2\times0.1)\cdot(1.5+2\times0.1)}{10} \cdot 1 = 1.244$, respectively. The cost reduction is 1.68. This scenario suggests that splitting overlapping partitions can reduce search cost.

The above scenario analysis shows that the benefit model proposed in this paper tends to generate a small number of single block partitions. This corresponds to DFS's storage design, which always splits large files into several single blocks of fixed size (e.g., 128 MB). Moreover, the additional overhead added by calculating the benefits through the master file is negligible.

In order to estimated the cost savings from data reorganization, we consider the three scenarios mentioned above. However, in practice, estimating the cost savings of scenario 3 is a trivial because it requires knowing the distribution of data within the partition in advance, which is not available in the master file. Therefore, only the first two scenarios are considered in this paper. It is a challenge to calculate the benefit without knowing the actual $\mathcal{G}_t$. Thus, the key idea in computing the benefits is to estimate $benefit(\mathcal{G}_t, \mathcal{Q})$ without physically reorganizing $\mathcal{G}_t$. To address this challenge, we estimate $\mathcal{G}_{t+1}$ to obtain $\widehat{\mathcal{G}_{t+1}}$, which is an estimated set of partitions. To estimate $\widehat{\mathcal{G}_{t+1}}$, we assume that the reorganization produces $m = cblocks(\mathcal{G}_t)$ single block, equal-sized, disjoint partitions. According to this assumption, the total area of m partitions is equal to the total area of $\mathcal{G}_t$. Therefore, the estimated area of each new partition $p'_i (i = 1, \ldots, m)$ satisfies the following condition.

$$\widehat{w(p'_i)} \cdot \widehat{h(p'_i)} = \frac{w(\mathcal{G}_t) \cdot h(\mathcal{G}_t)}{cblocks(\mathcal{G}_t)} \tag{13}$$

Lemma 3. *$QC(\widehat{\mathcal{G}_{t+1}}, \mathcal{Q})$ has a lower bound. (see the proof in Appendix A.3)*

Although our assumption represents an ideal scenario, it still serves as a good indicator of the potential benefits that can obtained. For example, in practice, the reorganization process may obtain lower benefits (e.g., in Fig. 4(c), where the reorganized partitions are overlapping) or higher benefits (e.g., in Fig. 4(b), where the empty region is not covered after reorganization). Finally, we reorganize $\mathcal{G}_t$ as the estimated benefit of $\widehat{\mathcal{G}_{t+1}}$ rewritten as:

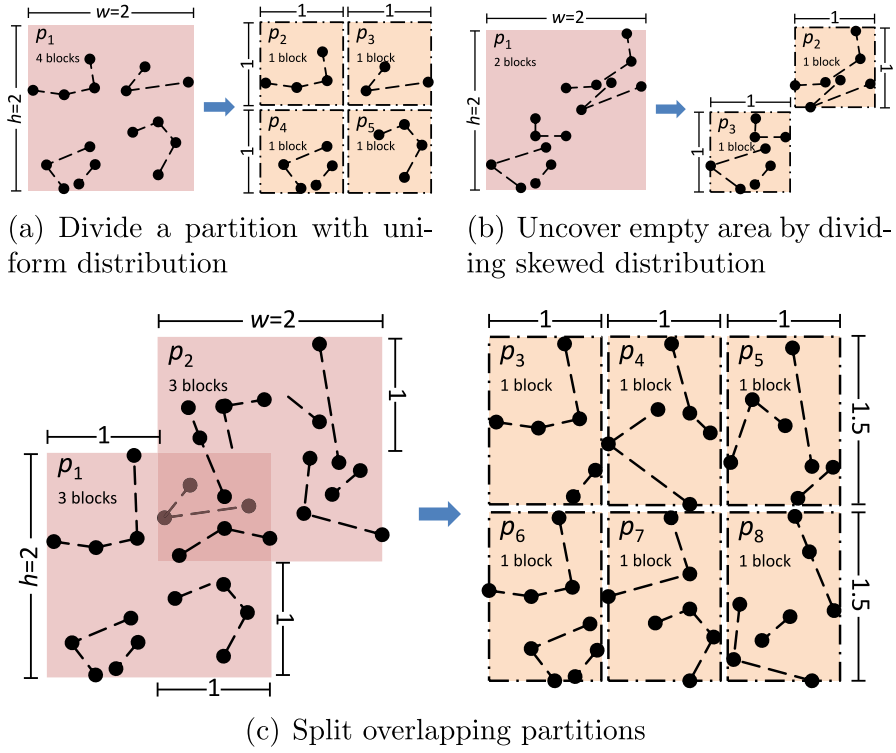
$$\widehat{benefit}(\mathcal{G}_t, \mathcal{Q}) = QC(\mathcal{G}_t, \mathcal{Q}) - QC(\widehat{\mathcal{G}_{t+1}}, \mathcal{Q}) \tag{14}$$

In summary, we can use Eq. (14) to select the partitions with the maximum *benefit upper bound* for reorganization when the disk I/O budget is limited, so as to obtain the optimized set of partitions. Since incremental partitioning process is an NP-hard (Section 3.2), we design a heuristic to address this problem, which will be discussed in Section 6.

6. Instantiation of the Tinba framework

6.1. Cost-based instantiation

This section presents a cost–benefit model-based heuristic (termed CBM) to instantiate the Tinba framework. The heuristic integrates



(a) Divide a partition with uniform distribution

(b) Uncover empty area by dividing skewed distribution

(c) Split overlapping partitions

Fig. 4. Three scenarios for reorganizing partitions to improve partition benefit.

a cost–benefit model to minimal partitioning costs. When reorganizing partitions, any partitioning strategy with strong spatial locality and good load balancing can be used. We chose to use STR (Sort-Tile-Recursive) as the trajectory partitioning strategy because of its simplicity and proven effectiveness by [40], but other trajectory partitioning techniques can also be used, e.g., R-tree and R*-Grove [41]. All these techniques are based on sample partitioning. The instantiation steps are detailed below. C_1

Data Flushing: As shown in Fig. 5(a), this step employs the R-tree insertion algorithm to select and append a partition for each trajectory. The existing partitions and unpartitioned new data (or deletion markers) are input into the flushing step.

Partition Selection: This problem has been proven to be NP-hard in Section 3. Therefore, based on the benefit model, we propose a greedy algorithm to select a set of most beneficial partitions for reorganization.

Algorithm 1 presents the pseudo code of greedy algorithm. The input are current partition $\mathcal{P}$, budget $\mathcal{B}$, and query trajectory $\mathcal{Q}$. The output is the selected set of partitions for reorganization. We initializes a set of selected partitions $\mathcal{G}$ (Line 1). We then traverse all partitions in $\mathcal{P}$ and calculate the benefit for each partition (Lines 5–10). Specifically, we calculate the benefit b for each partition p_i when it is added to $\mathcal{G}$ (Line 5). If b is greater than the maximum benefit, then p_i is added to $\mathcal{G}$ and removed from $\mathcal{P}$ (Lines 6–10). This step is repeated until the allocated budget $\mathcal{B}$ is reached. Finally, the set of selected partitions $\mathcal{G}$ is returned.

Data Reorganization: When the partition selection step provides a collection of low quality partitions for reorganization, we need to determine whether to reorganize them all at once or in smaller groups. If they are reorganized in a group, this can be quite inaccurate if the gaps between partitions are large. As a result, we first group the selected partitions, add all overlapping or partition distances less than threshold ϵ to a group, and then individually reorganize each group. The reason is that grouping the selected partitions decreases the overlap between the reorganized partitions and the existing partitions, making the estimated benefits more realistic.

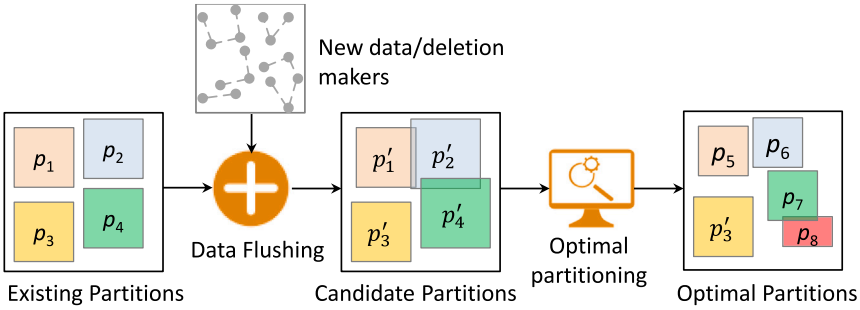
Algorithm 1: Greedy Selection Algorithm

Input: Existing partition $\mathcal{P} = \{p_1, \ldots, p_m\}$, threshold $\mathcal{B}$, search trajectory $\mathcal{Q}$;
Output: Selected partitions $\mathcal{G}$;
1 Initialize a set of selected partitions $\mathcal{G} = \{\}$;
2 **while** $ablocks(\mathcal{G}) \leq \mathcal{B}$ *and* $asize(\mathcal{P}) > 0$ **do**
3 $\quad max\_benefit = 0$;
4 $\quad$ **foreach** $p_i \in \mathcal{P}$ **do**
5 $\quad\quad b = \max\Big\{benefit(\widehat{\mathcal{G} \cup \{p_i\}}, \mathcal{Q}),$
6 $\quad\quad\quad benefit(\widehat{\mathcal{G}}, \mathcal{Q})\Big\} + benefit(\widehat{p_i}, \mathcal{Q})$;
7 $\quad\quad$ **if** $b > max\_benefit$ **then**
8 $\quad\quad\quad max\_benefit = b$;
9 $\quad\quad\quad p* = p_i$;
10 $\quad\quad \mathcal{G} = \mathcal{G} \cup \{p*\}$;
11 $\quad\quad \mathcal{P} = \mathcal{P} - \{p*\}$;
Result: $\mathcal{G}$

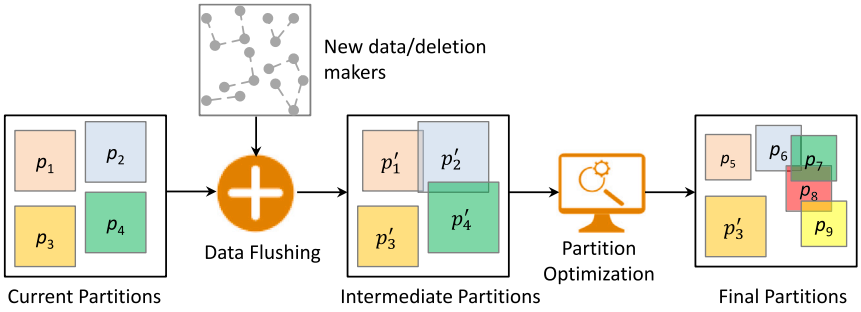
6.2. R-tree-based instantiation

This section presents a R-tree-based heuristic (termed R-P) to instantiate the Tinba framework. The basic idea involves treating each partition as a leaf node in an R-tree. The data flushing, partition selection and data reorganization steps are illustrated below.

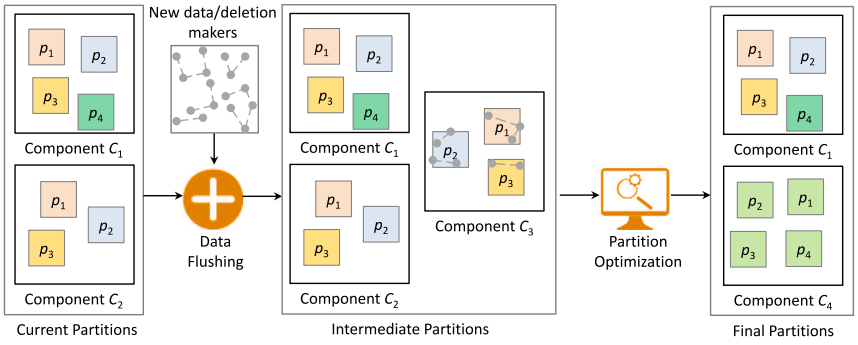
Data Flushing: This step employs the insertion algorithm of the R-tree, selecting a partition for each track, as shown in Fig. 5(b). The input to this step is the existing partition and a batch of unpartitioned new data or deletion markers. Using the MBR of the existing partition, this step scans the unpartitioned data and follows the ChooseSubtree method of the R-tree to attach each track to a partition. The output of this step is a set of intermediate partitions.



(a) An instantiation example based on the CBM method



(b) An instantiation example based on the R-P method



(c) An instantiation example based on the LSM-P method

Fig. 5. Three different instantiation of the Tinba framework.

Partition Selection: After the data flushing is finished, this step determines the partitions that need to be reorganized. Based on the R-tree design, partition selection step selects all partitions that exceed the maximum capacity of M (M = 128 MB in this paper). For example, in Fig. 5(b), partitions p'_1, p'_2 and p'_4 are chosen for reorganization as they exceed the maximum capacity.

Data Reorganization: The data reorganization step reorganizes the partitions chosen in the former step using any static partitioning algorithm with strong locality (e.g., STR). Particularly, R-P only divides the partition into smaller partitions, and has no merging mechanism for smaller partitions. Therefore, when the dataset is updated over time, many small partitions may exist, as shown in Fig. 5(b).

6.3. LSM-tree-based instantiation

This section presents a LSM-tree-based heuristic (termed LSM-P) to instantiate the Tinba framework. For the LSM-tree, each batch is flushed and indexed as an individual component. the LSM-tree compression policy merges these components based on their size and creation time.

Data Flushing: This step ingests a batch that is indexed by an R-tree. As the batch size may be larger than the DFS block size, the batch may contain more than one partition. As shown in Fig. 5(c), the new batch is indexed as new component C_3 in addition to the current components C_1 and C_2. Unlike R-P flushing, LSM-P creates new partitions, while R-P does not create new partitions.

Partition Selection: The partition selection step obtains the metadata of the components, i.e., component size and creation time, from the master and auxiliary files. We use HBase's LSM compression policy to identify the components to be merged.

Data Reorganization: The data reorganization step reorganizes all partitions from the components chosen in the former step into a new component. Fig. 5(c) illustrates how components C_2 and C_3 are reorganized into a new component C_4.

7. Experiments

In this section, we conduct a comprehensive experiments on real-world and synthetic trajectory datasets to highlight the advantages of the proposed work over other alternative methods.

7.1. Experimental setup

Datasets. We use the following real-world and synthetic datasets in our experiments.

OSM-TRAJ(search): It contains all publicly available GPS tracks (of various moving objects) collected by OpenStreetMap[4]. We filtered out clearly anomalous trajectories, including those are extremely long (spanning across the world) or too short (staying at one point). OSM-TRAJ(search) contains 6.5 million trajectories and is 82.4 GB in size.

OSM-TRAJ(join): It is a subset of OSM-TRAJ(search). Since distributed similarity join requires more memory than search, we generated OSM-TRAJ(join) of size 50 GB by sampling OSM-TRAJ(search).

SYN-TRAJ: To test the scalability of various systems, we also synthesize a trajectory dataset with 400.2 GB, the process is as follows: we take the entire road network of the United States from TIGER/Line [42], generate a large number of shortest path queries, and use the returned shortest paths as synthetic trajectories. To better simulate real trajectories on the road network, we randomly generate shortest path queries such that the distances of the shortest paths returned to follow a Gaussian distribution with both mean and standard deviation set to 30 km.

Methods tested. We have compared the following four methods, including AsterixDB [11] (VLDB 2014), TrajMesa [1] (ICDE 2020), DFT [15] (VLDB 2017) and DITA [3] (SIGMOD 2018).

AsterixDB[5]: A scalable, ingestion-oriented big data management system that supports high frequency insertions. It natively supports the LineString type instead of the trajectory type. However, LineString defines a portable data interchange format, GeoJSON[6], which is a geographic representation of an array of point coordinates. In our experiments, we use this format to store trajectory as LineString, thus extending it to support trajectory similarity analytics.

TrajMesa[7]: An *key–value* database which allows per partition file remain several KBs in size. It uses $XZ2^{+}$ index scheme for trajectory similarity search.

DFT[8]: A distributed in-memory query framework for processing trajectory similarity search over a large set of trajectories, which supports Fréchet distance and Hausdorff distance.

DITA[9]: A most recent distributed in-memory trajectory analytics system that support trajectory similarity search and join.

Tinba, our method: For the Tinba framework, we have presented a heuristic CBM in Section 6. Moreover, we selected the R-P and LSM-P methods in Section 6 for comparison with the CBM method.

For the fairness of the experiments, we make the following settings against the baseline techniques.

1. For AsterixDB, we employ HDFS as the external data layer for access. Other technologies employ HDFS as the storage layer.
2. We only consider the global pruning in TrajMesa, and the batch filtering technique in DITA and DFT.
3. To eliminate system cache,[10] we randomly select 100 different queries, perform each query only once, and take the 5%–95% interval and median response time of all queries as the final results.

Evaluation metrics. We focus on evaluating the following metrics in our experiments. Ingestion Time: the runtime to ingest a batch of data.

[4] https://www.openstreetmap.org/.
[5] https://asterixdb.apache.org.
[6] https://en.wikipedia.org/wiki/GeoJSON.
[7] http://trajmesa.urban-computing.com/code/TrajMesa-src-2019-06-24.zip
[8] https://github.com/InitialDLab/traj-sim-spark
[9] https://github.com/TsinghuaDatabaseGroup/DITA
[10] HBase will cache results in memory to expedite the same queries.

Table 2
Parameter Settings (Default value is bolded).

Parameter	Value
Threshold ϵ	0.001, 0.002, 0.003, 0.004, **0.005**
b	32 MB, 64 MB, **128 MB**, 256 MB, 512 MB
Batch Size	4 GB, **8 GB**, 16 GB, 24 GB, 32 GB
$\mathcal{B}$	16, 32, **64**, 128, 256
δ	**0**, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8
Dataset Size	OSM-TRAJ(search): 50, 54, 58, 62, 66, 70, 74, 78, **82** OSM-TRAJ(join): 8, 16, 24, 32, 40, **48** SYN-TRAJ: 368, 376, 384, 392, **400**

Total Area: let $\mathcal{A}(p_i)$ be the area of partition p_i, which is the product of its side lengths. The total area of the partitions is defined as $\mathcal{A}(\mathcal{P}) = \sum_{p_i \in \mathcal{P}} ablocks(p_i) \cdot \mathcal{A}(p_i)$. We multiply by the number of physical blocks $ablocks(p_i)$ of partition p_i because big trajectory analytics systems process each block separately. The total area is preferably reduced to minimize overlap with the query buffer region (see Fig. 3).

PSD: Partition's Standard Deviation. Let $\overline{asize}(p_i)$ be the average partition size. The PSD is defined as $PSD(\mathcal{P}) = \sqrt{\frac{\sum_{p_i \in \mathcal{P}} \left(asize(p_i) - \overline{asize}(p_i)\right)^2}{|\mathcal{P}|}}$. Lowering this value is preferred to balance the load across partitions.

Block Utilization: block utilization measures how full HDFS blocks are and is defined as $\mathcal{U}(\mathcal{P}) = \frac{\sum_{p_i \in \mathcal{P}} asize(p_i)}{b \cdot \sum_{p_i \in \mathcal{P}} ablocksp_i}$. In big data applications, each block is processed in a separate task that takes seconds to setup. Having full or near-full blocks minimizes the overhead of setup. The maximum block utilization is 1.0.

Latency: end-to-end execution time for a similarity search/join.

Implementation. All the experiments were conducted on a cluster with 1 master and 31 slave nodes with 16 GB RAM and 4-core Intel(R) Xeon(R) CPU E5-2620@2.10 GHz processors. Each node in the cluster was connected to a Gigabit Ethernet switch and ran Ubuntu 16.04.01 with Spark 2.1.0 and Hadoop 2.7.2. All algorithms were implemented using Java and Scala in Apache Spark, a popular distributed computing engine.

Parameter Settings. Table 2 listed the parameter settings used in the experiments. In particular, the dataset size in Section 7.2 is set to $20, 21, \ldots, 80$, and the total size of incremental data for delete workload is half of the dataset in Section 7.3. For example, the dataset is 50 GB, then the total incremental data size is 25 GB, which is split into 4 batches. When we varied a parameter, the others were set to default values. Based on existing studies [3], we select $0.001, 0.002, \ldots, 0.005$ as distance thresholds.

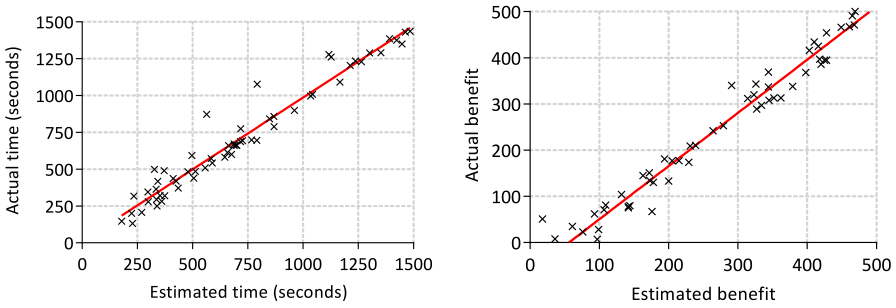
7.2. Effectiveness of cost-benefit model

We first study the effectiveness of the cost–benefit model separately. We partitioned the OSM-TRAJ(search) dataset of different sizes, and then randomly selected varying trajectories to perform similarity searches on these partitions.

Fig. 6(a) shows the relationship between estimated and actual search times. The results show a linear relationship between estimated and actual search times with a correlation equal to 0.976. This demonstrates the effectiveness of the cost model. The relationship between estimated and actual benefits is depicted in Fig. 6(b). We observe that the estimated and actual benefits are highly correlated, with a correlation of 0.985. This suggests that the proposed benefit model can be employed with reliability in the partition selection process.

7.3. Comparison of three instantiation methods

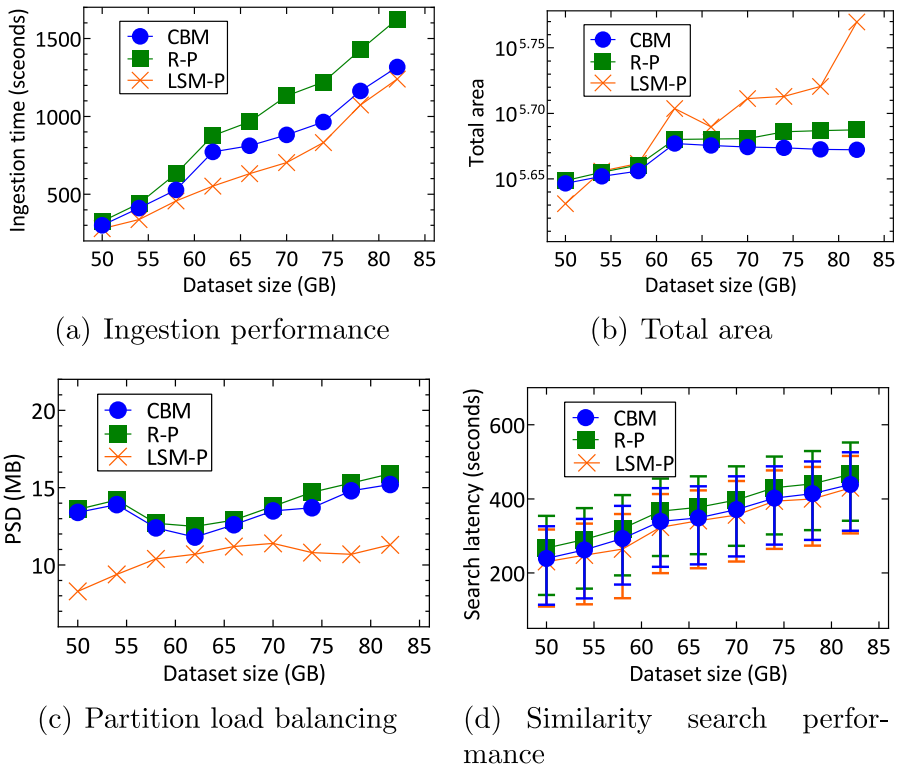
Figs. 7–9 showed how the three heuristics performed on three workloads: insert, delete, and insert+delete. We compared ingestion time, total area, PSD, and search latency using the OSM-TRAJ(search)



(a) Estimated and actual serach time

(b) Estimated and actual reorganize benefit

Fig. 6. Verification of the cost–benefit model on OSM-TRAJ(search).



(a) Ingestion performance

(b) Total area

(c) Partition load balancing

(d) Similarity search performance

Fig. 7. Comparison of three heuristics on OSM-TRAJ(search) with insert workload.

dataset. We split the raw OSM-TRAJ(search) into 8 GB batches to simulate the data flow in Fig. 3. Each batch covers a subset of the dataset and is designed to ensure that the distribution of ingested data changes over time.

Figs. 7(a), 8(a) and 9(a) showed the ingestion time for three heuristics on three workloads, respectively. It is clear that CBM outperforms R-P, but both are slower than LSM-P. There are two reasons: (1) CBM and R-P have the same data flushing operation, which appends new entries or deletion marks to current partitions. The optimal partitioning operation distinguishes R-P from CBM in that R-P separates each overflow partition independently, while CBM selects only the low quality partitions for optimal processing. (2) The LSM-P flushing operation partitions and persists the newly ingested data in an individual component on disk, which is usually more efficiency than appending to an existing partition. LSM-P optimal partitioning operation is performed periodically in accordance with the LSM's compaction policy, which always reassembles selected components together. However, the compaction operation is not always triggered, the ingestion performance on LSM-P is usually faster than CBM and R-P.

Figs. 7(b), 8(b) and 9(b) evaluated the total area of three different heuristics on three workloads when varying the dataset size. We had following observation. (1) CBM has the smallest total area because CBM uses a cost–benefit model to optimize the partition reorganization process to produce high-quality partitions. (2) R-P's total area is slightly larger than that of CBM but extremely smaller than that of LSM-P. There are two reasons for this: (*i*) R-P's overflow node splitting strategy makes the total area of the partition after splitting no larger than the total area before splitting, while LSM-P does not; (*ii*) R-P achieves local optimality by splitting overflow nodes, while CBM achieves global optimality through a cost–benefit model. (3) The total area of LSM-P increases with the addition of new components and decreases periodically when the merge operation is triggered.

We performed a similarity search task on the partitioned dataset in order to select the optimal among the three heuristics. All heuristics perform well in an insert workload, as shown in Fig. 7(d). For the delete and insert+delete workloads in Figs. 8(d) and 9(d), however, CBM is always preferable.

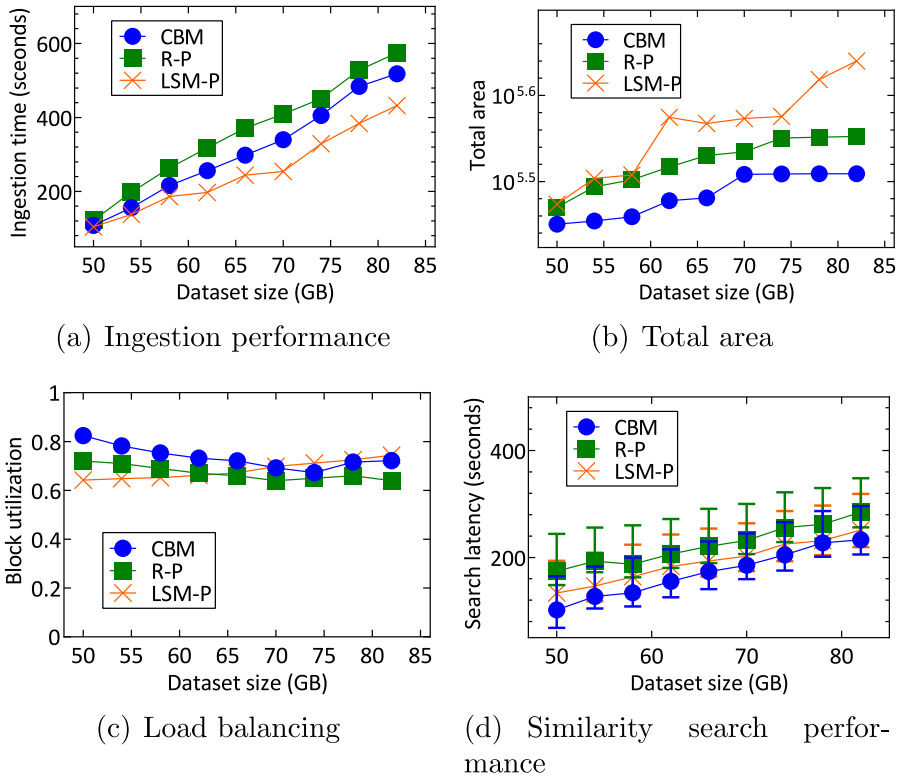


Fig. 8. Comparison of three heuristics on OSM-TRAJ(search) with delete workload.

In all remaining experiments, we will use CBM as the **default** instantiation of Tinba for comparison against baseline techniques.

7.4. Comparison against tested methods

Fig. 10 compares the performance of all tested methods. Since DFT and DITA are designed for static trajectory data, they must reorganization the entire partition after each data flushing process.

7.4.1. Ingestion performance

The experimental dataset size is initially set to 46 GB, and then batches of 4 GB in size are added one at a time in order to test the ingestion performance when the data is large. Experiments were conducted to measure the cumulative time of ingest all batches. The experimental results are as follows. (1) As the dataset size grows, all techniques take more time as a larger dataset produce more batches. (2) Tinba significantly outperforms baseline methods. For example, when the dataset is equal to 78 GB, DFT took 5652 s, DITA took 5125 s, TrajMesa took 2432 s, AxterixDB took 2580 s while Tinba took 1064 s. There are three reasons: (*i*) Tinba adopt the *block-level* partitioning strategy, the overhead is low. (*ii*) Only new records are inserted by AsterixDB and TrajMesa, but they are impacted by the *key–value* index and require the index structure to locate the exact location of each record. (*iii*) Tinba optimized the data reorganization process with cost–benefit model (which is proposed in Section 5), while DFT and DITA repartitioned the whole data.

7.4.2. Scalability

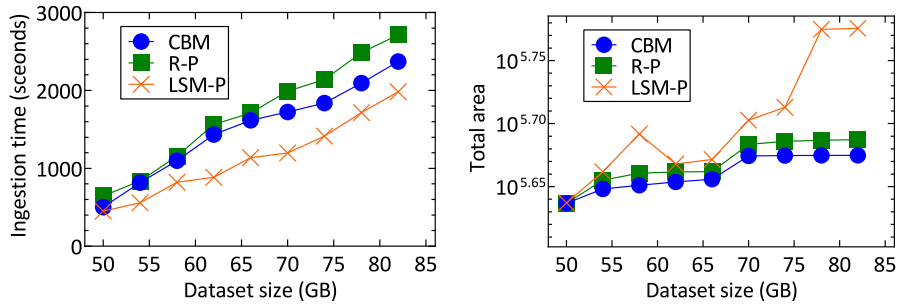
We evaluated the scalability of Tinba and AsterixDB as shown in Fig. 10(b) with different dataset size, since Tinba and AsterixDB are the best way to ingest the workload. We start with a 360 GB data and append several 8 GB batches to verify that Tinba and AsterixDB can process very large dataset. As figure suggested, the gap between Tinba and AsterixDB tends to be larger as the size of the data increases. The reason is that Tinba achieves better scalability with its effective incremental partitioning scheme, well-designed cost–benefit model. For example, when the dataset size was increased from 384 GB to 392 GB, AsterixDB spent 1676 s, while Tinba only spent 782 s.

7.4.3. Similarity search performance

Fig. 10(c) showed the performance of similarity search of all techniques. We omit the results for TrajMesa as it took more than 12.4 min to perform the search process. We observed that AsterixDB has higher query latency than other techniques. This is because AsterixDB uses the *key–value* indexing scheme, which can quickly locate a single record in the results. However, as data size increases, accessing records individually becomes expensive. DFT and DITA showed reasonable performance on small datasets and are also much better than AsterixDB on large dataset. Also, Tinba is significantly better than other techniques. The reasons were two-fold: (*i*) Tinba adopts the *block-level* partitioning scheme to reduce the number of access partitions, which reduces the overall processing time of similarity search. (*ii*) Tinba designed a cost–benefit model to produce high quality partitions.

7.4.4. Similarity join performance

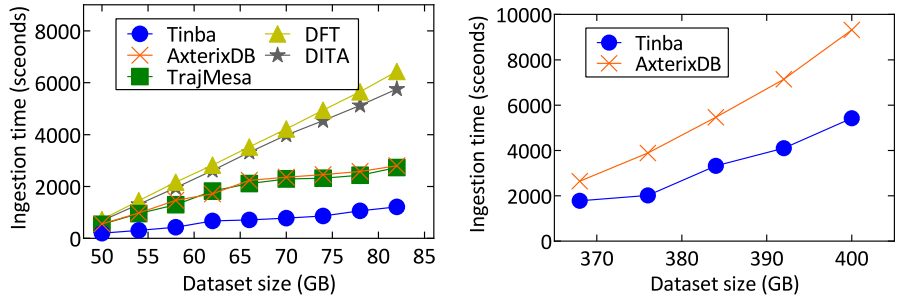
We performed a trajectory self-join search to find all similar pairs on OSM-TRAJ (join) dataset. We did not consider DFT as it only supports trajectory similarity search. Fig. 10(d) showed the join latency in Tinba, AsterixDB and DITA. Experimental observations are as follows. (1) Tinba and DITA are constantly better than AsterixDB by 6.5x-8.8x especially when inputs size was large. The reason is that AsterixDB had smaller partitions and they required more jobs to complete the search than block level partitioning techniques. (2) Tinba achieves the smallest join latency, but is only 1.3x faster than DITA on average. This is because: (*i*) DITA always reorganizes the entire partition, so it becomes highly optimized; (*ii*) Tinba optimally reorganizes the intermediate partitions by a optimal partitioning step after each batch ingestion, so it can achieve superior performance.



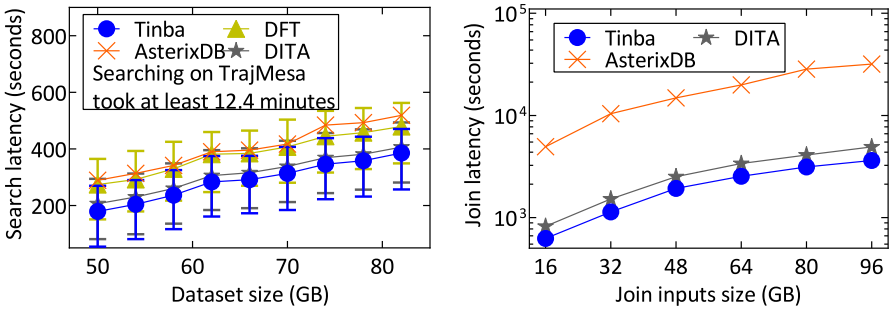
(a) Ingestion performance (b) Total area

(c) Load balancing (d) Similarity search performance

Fig. 9. Comparison of three heuristics on OSM-TRAJ(search) with insert+delete workload.



(a) Ingestion performance on OSM-TRAJ(search) (b) Scalability on SYN-TRAJ



(c) Search latency on OSM-TRAJ(search) (d) Join latency on OSM-TRAJ(join)

Fig. 10. Comparison against tested methods.

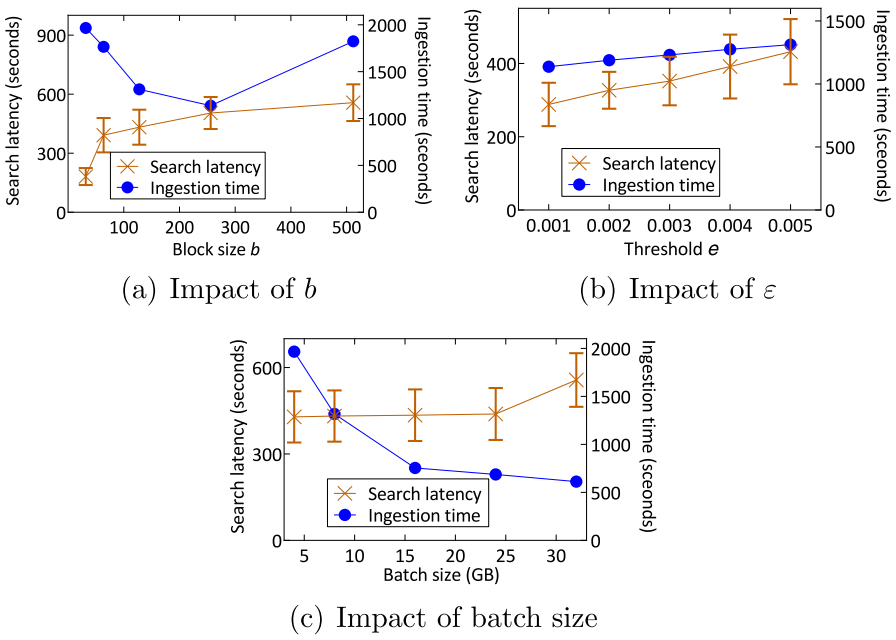


Fig. 11. Influence of different partitioning parameters in Tinba.

7.5. Impacts of b, ε and batch size on tinba

Fig. 11(a) showed how HDFS block size impacted the ingestion time and search latency. We ingested the OSM-TRAJ(search) dataset, while varied the block size and had the following observations. With the increase of block size, the ingestion time decreases and then gets larger, shows a "concave" trend. However, search efficiency slows down for larger block sizes, which is to be expected since the search has to process more data on average. This suggests that the best choice of the block size for HDFS is 128 MB or 256 MB, since the gap between ingestion time and search latency is minimal at these two points.

Fig. 11(b) showed how the threshold ε impacted the cost model. We varied the threshold ε and observed that the maximum gap of ingestion time is 175 s, which is relatively stable. The search efficiency showed a decreasing trend, but the average search latency differed by 144 s at maximum, which was a small gap. Intuitively, partitions optimized for one threshold ε are expected to fit well into other ε.

As shown in Fig. 11(c), we evaluated the performance of ingestion and search with varying batch size and had the following observations. (1) With the increases of the batch size, the ingestion time drastically decreased, due to fewer flushing and reorganization operations performed during the ingestion phase. (2) Tinba achieved relatively stable search performance when the batch size is from 4 GB to 24 GB since Tinba's optimized data reorganization to maintain high quality partitions. However, when the batch size increased from 24 GB to 32 GB, the search latency started to rise because larger batches resulted in less frequent reorganization and fewer optimal partitioning processing, which produced low quality partitions.

8. Conclusions

In this paper, we proposed Tinba, an incremental partitioning framework for efficient trajectory data analytics. We recasted the incremental partitioning problem as an optimal partitioning problem and proves its NP-hardness. We developed a heuristic techniques to demonstrate the feasibility of Tinba. We proposed a cost model to estimate the processing cost of analytical queries and design a benefit model to estimate the cost savings of data reorganization. We propose a greedy algorithm to select a set of most beneficial partitions for reorganization. The incremental partitioning technique of Tinba is adapted to most trajectory similarity measures, e.g., DTW, EDR, LCSS and Fréchet distance. Comprehensive experiments on real-world and synthetic datasets demonstrate that Tinba outperforms other big trajectory partitioning methods in terms of ingestion performance and partition quality. In the future, we plan to optimize cost model by considering partitioned data migration costs, and improve the versatility of the framework by considering adaptive cost models.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This work was supported by the National Key Research and Development Program of China (2020YFF0410947) and the National Natural Science Foundation of China (62103072). Additional funding was provided by the China Postdoctoral Science Foundation (2021M690502).

Appendix A. Proof of Theorem and Lemma

A.1. Proof of Theorem 1

To prove that the optimal partitioning problem is also NP-hard, a *polynomial reduction* needs to be constructed to transform the 0–1 Knapsack problem [43] to this problem. Specifically, we need to show that (1) 0–1 Knapsack problem can be reduced to a optimal partitioning problem, (2) the optimal solution can be mapped from the 0–1 Knapsack problem to this problem, (3) both problem reduction and solution mapping are required Polynomial time.

Reduction Algorithm: To reduce 0–1 Knapsack problem to this problem, we define a cost function independently for each partition. The form of the reduction is as follows.

- The current partitioning $\mathcal{P}$ contains n partitions, each partition with size w_i and query cost v_i.
- The threshold $\mathcal{B}$ is equivalent to the weight capacity W.
- The cost function for one partition $QC(p, \mathcal{Q})$ is defined as:

$$QC(p,\mathcal{Q}) = \begin{cases} 0 & if\ p \in \mathcal{P} \\ \sum_{p_i \in \mathcal{P}} v_i - \sum_{p_i \in \mathcal{P}'} v_i & others \end{cases} \tag{A.1}$$

Finally, we define $QC(\mathcal{P}', \mathcal{Q}) = \sum_{p_i \in \mathcal{P}'} QC(p_i, \mathcal{Q})$. In particular, the cost for a partitioning state $\mathcal{P}'$ is the total of cost for each partition in $\mathcal{P}'$. The objective function $\sum_{i \in I} w_i \le W$ of the Knapsack problem reduces to $\sum_{i \in I} \lceil \frac{w_i}{b} \rceil + \sum_{i \in I} \lceil \frac{w_i}{b} \rceil \le \mathcal{B} \Rightarrow EC(\mathcal{P}, \mathcal{P}') \le \mathcal{B}$.

Solution Mapping: Assume that I is an optimal solution to 0–1 Knapsack problem, $\mathcal{F}$ is a mapping function, and $\mathcal{P}' = \{p_i | i \in I\}$ is a solution of optimal partitioning problem. For any $i \in I$, we have $\mathcal{F}(i) \in \mathcal{P}'$. Similarly, for any $p_i \in \mathcal{P}'$, we have $\mathcal{F}(p_i) \in I$.

Reduction Complexity: The above reduction algorithm and solution mapping can be completed in time complexity $\mathcal{O}(n)$.

To sum up, it can be concluded that the optimal partitioning problem is NP-hard.

A.2. Proof of Theorem 2

We first give the two events: event E_1, the query $\mathcal{Q}$ intersects with the data domain $MBR(D)$; event E_2, the query $\mathcal{Q}$ intersects with the partition p_i.

According to the Bayesian theorem, the probabilities of E_1 and E_2 satisfy $P(E_1|E_2) \cdot P(E_2) = P(E_2|E_1) \cdot P(E_1)$. Also $P(E_1|E_2) = 1$, and $P(E_2) = P(E_2|E_1) \cdot P(E_1)$. The possible values of $P(E_1)$ are 0 or 1, which denote disjoint and intersection, respectively. We ignore the case where $P(E_1) = 0$, so the query $\mathcal{Q}$ and the partition p_i must not intersect at this point. When $P(E_1) = 1$, we have $P(E_2) = P(E_2|E_1) \Rightarrow$

$$P(E_2) = \frac{S(MBR(B_i))}{S(MBR(D))} \cdot \frac{S(\mathcal{Q} \cap MBR(D))}{S(MBR(D))}$$

where $S()$ and $\mathcal{Q} \cap MBR(D)$ denote the area function and intersecting region, respectively. Also, $\frac{S(\mathcal{Q} \cap MBR(D))}{S(MBR(D))}$ is consistent for each partition. For simplicity, let $\frac{S(\mathcal{Q} \cap S(D))}{S(D)} = 1$. So we have $P(E_2) = \frac{MBR(B_i)}{S(MBR(D))}$.

A.3. Proof of Lemma 3

According to Eq. (5), it can be derived that

$$QC_b(p_i', \mathcal{Q}) \ge \frac{\widehat{w(p_i')} \cdot \widehat{h(p_i')} + 2\varepsilon \sqrt{\widehat{w(p_i')} \cdot \widehat{h(p_i')}} + 4\varepsilon^2}{w(\mathcal{P})h(\mathcal{P})} \cdot$$
$$ablocks(p_i') = QC_b^L(p_i', \mathcal{Q})$$

and similarly $SC_s(p_i', \mathcal{Q}) \ge LC_s^L(p_i', \mathcal{Q})$. So, we have that $QC(\widehat{G_{t+1}}, \mathcal{Q}) \ge LC_b^L(p_i', \mathcal{Q}) + SC_s^L(p_i', \mathcal{Q})$.

Appendix B. Extension to other similarity measures

B.1. Dynamic time warping (DTW)

Definition 8 (*DTW*)**.** Given two trajectories $\mathcal{T} = \{t_1, \ldots, t_m\}$ and $\mathcal{Q} = \{q_1, \ldots, q_n\}$, DTW [36] is computed as below.

$$DTW(\mathcal{T},\mathcal{Q}) = \begin{cases} \sum_{i=1}^{m} \textsc{Dist}(t_i, q_1) & if\ n = 1 \\ \sum_{j=1}^{n} \textsc{Dist}(t_1, q_j) & if\ m = 1 \\ \textsc{Dist}(t_m, q_n) + \min\Big(DTW(\mathcal{T}^{m-1}, \mathcal{Q}^{n-1}), \\ DTW(\mathcal{T}^{m-1}, \mathcal{Q}), DTW(\mathcal{T}, \mathcal{Q}^{n-1}) \Big) & others \end{cases}$$

where $\mathcal{T}^{m-1}$ is the prefix trajectory of $\mathcal{T}$ by removing the last point.

According to Eq. (2) and Definition 8, we can conclude that $DTW(\mathcal{T}, \mathcal{Q})$ is not less than $Fréchet(\mathcal{T}, \mathcal{Q})$ constant. As shown in Fig. B.12, given two trajectories $\mathcal{T}_1$ and $\mathcal{T}_3$, $DTW(\mathcal{T}_1, \mathcal{T}_3) = 6.41 > Fréchet(\mathcal{T}_1, \mathcal{T}_3) = 1.41$. Tinba does not need to update ϵ by accumulating distances from it when querying the partition. Similarly, we can still utilize buffer region filtering and cost-based estimation.

B.2. Edit distance on real sequences (EDR)

Definition 9 (*EDR*)**.** Given two trajectories $\mathcal{T}$ and $\mathcal{Q}$, and a matching threshold $\epsilon \ge 0$, EDR [44] is:

$$EDR(\mathcal{T},\mathcal{Q}) = \begin{cases} n & if\ m = 0 \\ m & if\ n = 0 \\ \min\Big(EDR(\mathcal{T}^{2,m}, q^{2,n}) + subcost(t_1, q_1), \\ EDR(\mathcal{T}^{2,m}, q) + 1, EDR(\mathcal{T}, q^{2,n}) + 1 \Big) & others \end{cases}$$

where $\mathcal{T}^{2,m}$ represents trajectory $\mathcal{T}$ with its first point removed, and $subcost(t, q) = 0$ if $\textsc{Dist}(t, q) \le \epsilon$; 1 otherwise.

Given two trajectories $\mathcal{T}_1$ and $\mathcal{T}_3$ in Fig. B.12, let $\epsilon = 1$, we have $EDR(\mathcal{T}_1, \mathcal{T}_3) = 2$. For each partition's MBR, we compute the distance to the query trajectory $\mathcal{Q}$. According Appendix B.2, if it is beyond ϵ, $subcost(t, q)$ is always equal to 1 and $EDR(\mathcal{T}, \mathcal{Q}) = \max(m, n)$, we safely prune this partition.

B.3. Longest common subsequence distance (LCSS)

Definition 10 (*LCSS*)**.** Given two trajectories $\mathcal{T}$ and $\mathcal{Q}$ with lengths m and n, an integer $\delta \ge 0$ and a matching threshold $\epsilon \ge 0$, $LCSS$ is defined as below [44]:

$$LCSS(\mathcal{T},\mathcal{Q}) = \begin{cases} 0 & if\ m = 0\ or\ n = 0 \\ 1 + LCSS(\mathcal{T}^{m-1}, \mathcal{Q}^{n-1}) & if\ |m - n| \le \delta\ \& \\ \max\Big(LCSS(\mathcal{T}^{m-1}, \mathcal{Q}), & \textsc{Dist}(t_m, q_n) \le \epsilon \\ LCSS(\mathcal{T}, \mathcal{Q}^{n-1}) \Big) & others \end{cases}$$

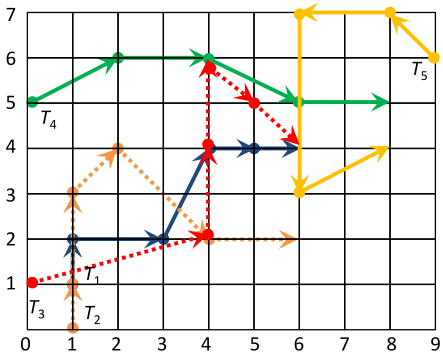
where $\mathcal{T}^{m-1}$ is the prefix trajectory of $\mathcal{T}$ with the last point removed.

Given two trajectories $\mathcal{T}_1$ and $\mathcal{T}_3$ in Fig. B.12, let $\delta = 1, \epsilon = 1$, we have $LCSS(\mathcal{T}_1, \mathcal{T}_3) = 5$. Similar to EDR, for each partition's MBR, we compute the distance to the query trajectory $\mathcal{Q}$. According Definition 10, if it is beyond ϵ, $LCSS(\mathcal{T}, \mathcal{Q})$ is always equal to 0, we also safely prune this partition.

Appendix C. Additional experiments

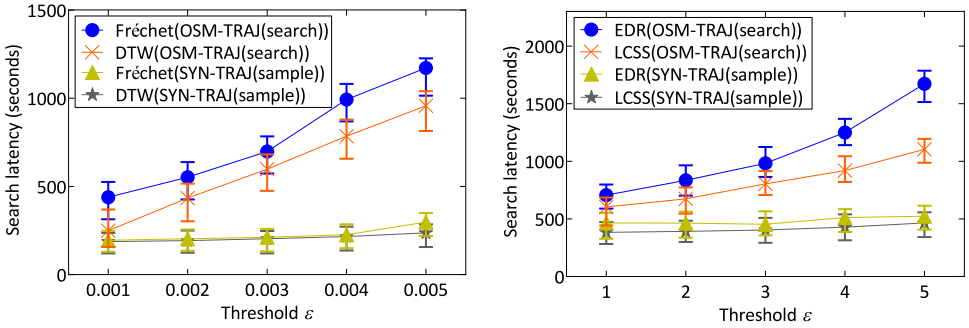
C.1. Evaluation of other distance functions

In this section, we evaluate the performance of Tinba on different similarity measures. The parameters ϵ and δ were set to 0.0001 and 3, respectively. We generate SYN-TRAJ(sample) by sampling 10% of SYN-TRAJ. The experimental results are shown in Fig. C.13. We could observe that: (1) All distance functions took more time as ϵ increases, because the larger the threshold, the more results; (2) LCSS was faster than EDR with the same threshold, because LCSS has a max interval constraint of matching point while EDR does not; (3) DTW is faster than Fréchet for the same threshold because DTW computes the cumulative sum of distances, while Fréchet selects the maximum distance.



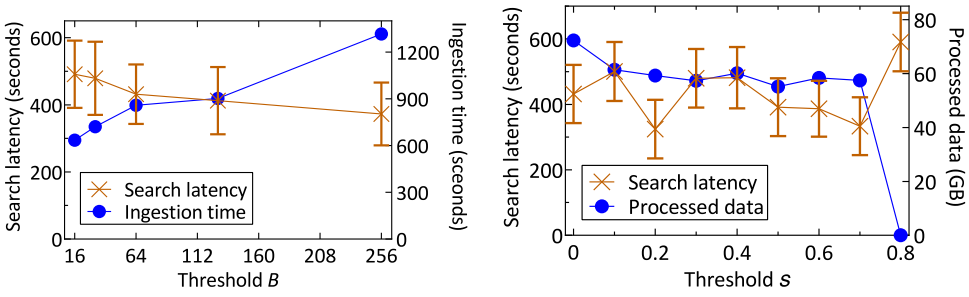
Notation	Trajectory
T_1	(1,1), (1,2), (3,2), (4,4), (5,4), (6,4)
T_2	(1,0), (1,3), (2,4), (4,2), (6,2)
T_3	(0,1), (4,2), (4,4), (4,6), (5,5), (6,4)
T_4	(0,5), (2,6), (4,6), (6,5), (8,5)
T_5	(9,6), (8,7), (6,7), (6,3), (8,4)

Fig. B.12. Example Trajectories.



(a) Varying ε: Fréchet and DTW (b) Varying ε: EDR and LCSS

Fig. C.13. Evaluation on OSM-TRAJ(search) and SYN-TRAJwith the Fréchet, DTW, EDR and LCSS functions.



(a) Impact of $\mathcal{B}$ (b) Impact of σ

Fig. C.14. Influence of parameters $\mathcal{B}$ and σ.

C.2. Impacts of $\mathcal{B}$ and σ

Fig. C.14(a) shows the impact of threshold $\mathcal{B}$ on search latency and ingestion time. With the increase of threshold, the query latency decreases gradually as a larger threshold cause more partitions to be reorganized, which improves partition quality. Note, however, that after $\mathcal{B} \geq 128$, there is no longer any significant reduction in query latency. The $\mathcal{B}$ does have a significant impact on ingestion time. This is because, when $\mathcal{B}$ increases, the number of blocks reads and the number of writing new condensed partitions also increase, which incurs significant I/O overhead. The trends for search latency and ingest time are opposite. This suggests that we can further reduce the search time while maintaining a high ingestion rate, and we plan to investigate how to choose an optimal $\mathcal{B}$ value in the future.

In order to highlight the impact of reorganization frequency in Tinba, we introduce a parameter σ, in which the reorganization process is only triggered if the benefit ratio (the ratio of $benefit(\mathcal{G}_t, \mathcal{Q})$ and $QC(\mathcal{G}_t, \mathcal{Q})$) is greater than threshold σ. When $\sigma = 0$, Tinba always reorganizes. Fig. C.14(b) shows how the value of σ does have an impact on the query latency and the amount of data being reorganized. With the increase of σ, a small amount of data is processed because there is less reorganization. The query latency worsens significantly only when σ is very high. The above shows that we can further optimize the partitioning time while keeping the same search performance, and we plan to further investigate this impact in the future.

References

[1] R. Li, H. He, R. Wang, S. Ruan, Y. Sui, J. Bao, Y. Zheng, Trajmesa: A distributed nosql storage engine for big trajectory data, in: 2020 IEEE 36th International Conference on Data Engineering, ICDE, 2020, pp. 2002–2005, http://dx.doi.org/10.1109/ICDE48307.2020.00224.

[2] Z. Zhang, C. Jin, J. Mao, X. Yang, A. Zhou, Trajspark: A scalable and efficient in-memory management system for big trajectory data, in: L. Chen, C.S. Jensen, C. Shahabi, X. Yang, X. Lian (Eds.), Web and Big Data, Springer International Publishing, Cham, 2017, pp. 11–26.

[3] Z. Shang, G. Li, Z. Bao, Dita: Distributed in-memory trajectory analytics, in: Proceedings of the 2018 International Conference on Management of Data,

SIGMOD '18, Association for Computing Machinery, New York, NY, USA, 2018, pp. 725–740, http://dx.doi.org/10.1145/3183713.3183743.
[4] H. Tan, W. Luo, L.M. Ni, Clost: A hadoop-based storage system for big spatio-temporal data analytics, in: Proceedings of the 21st ACM International Conference on Information and Knowledge Management, CIKM '12, Association for Computing Machinery, New York, NY, USA, 2012, pp. 2139–2143, http://dx.doi.org/10.1145/2396761.2398589.
[5] M. Bakli, M. Sakr, E. Zimányi, Distributed Spatiotemporal Trajectory Query Processing in Sql, SIGSPATIAL '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 87–98, http://dx.doi.org/10.1145/3397536.3422262.
[6] C. Düntgen, T. Behr, R.H. Güting, Berlinmod: A benchmark for moving object databases, VLDB J. 18 (6) (2009) 133–1368, http://dx.doi.org/10.1007/s00778-009-0142-5.
[7] P. Cudre-Mauroux, E. Wu, S. Madden, Trajstore: An adaptive storage system for very large trajectory data sets, in: 2010 IEEE 26th International Conference on Data Engineering, ICDE 2010, 2010, pp. 109–120, http://dx.doi.org/10.1109/ICDE.2010.5447829.
[8] S. Wang, Z. Bao, J.S. Culpepper, Z. Xie, Q. Liu, X. Qin, Torch: A search engine for trajectory data, in: The 41st International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '18, Association for Computing Machinery, New York, NY, USA, 2018, pp. 535–544, http://dx.doi.org/10.1145/3209978.3209989.
[9] X. Guan, C. Bo, Z. Li, Y. Yu, St-hash: An efficient spatiotemporal index for massive trajectory data in a nosql database, in: 2017 25th International Conference on Geoinformatics, 2017, pp. 1–7, http://dx.doi.org/10.1109/GEOINFORMATICS.2017.8090927.
[10] H. He, R. Li, S. Ruan, T. He, J. Bao, T. Li, Y. Zheng, Trass: Efficient trajectory similarity search based on key–value data stores, in: 2022 IEEE 38th International Conference on Data Engineering, ICDE, 2022, pp. 2306–2318, http://dx.doi.org/10.1109/ICDE53745.2022.00218.
[11] S. Alsubaiee, Y. Altowim, H. Altwaijry, A. Behm, V. Borkar, Y. Bu, M. Carey, I. Cetindil, M. Cheelangi, K. Faraaz, E. Gabrielova, R. Grover, Z. Heilbron, Y.-S. Kim, C. Li, G. Li, J.M. Ok, N. Onose, P. Pirzadeh, V. Tsotras, R. Vernica, J. Wen, T. Westmann, Asterixdb: A scalable, open source bdms, Proc. VLDB Endow. 7 (14) (2014) 1905–1916, http://dx.doi.org/10.14778/2733085.2733096.
[12] P. O'Neil, E. Cheng, D. Gawlick, E. O'Neil, The log-structured merge-tree (lsm-tree), Acta Inf. 33 (4) (1996) 351–385, http://dx.doi.org/10.1007/s002360050048.
[13] L. Alarabi, Summit: A scalable system for massive trajectory data management, SIGSPATIAL Special 10 (3) (2019) 2–3, http://dx.doi.org/10.1145/3307599.3307601.
[14] Q. Ma, B. Yang, W. Qian, A. Zhou, Query processing of massive trajectory data based on mapreduce, in: Proceedings of the First International Workshop on Cloud Data Management, CloudDB '09, Association for Computing Machinery, New York, NY, USA, 2009, pp. 9–16, http://dx.doi.org/10.1145/1651263.1651266.
[15] D. Xie, F. Li, J.M. Phillips, Distributed trajectory similarity search, Proc. VLDB Endow. 10 (11) (2017) 1478–1489, http://dx.doi.org/10.14778/3137628.3137655.
[16] X. Ding, L. Chen, Y. Gao, C.S. Jensen, H. Bao, Ultraman: A unified platform for big trajectory data management and analytics, Proc. VLDB Endow. 11 (7) (2018) 787–799, http://dx.doi.org/10.14778/3192965.3192970.
[17] Z. Fang, L. Chen, Y. Gao, L. Pan, C.S. Jensen, Dragoon: A hybrid and efficient big trajectory management system for offline and online analytics, VLDB J. 30 (2) (2021) 287–310, http://dx.doi.org/10.1007/s00778-021-00652-x.
[18] H. Yuan, G. Li, Distributed in-memory trajectory similarity search and join on road network, in: 2019 IEEE 35th International Conference on Data Engineering, ICDE, 2019, pp. 1262–1273, http://dx.doi.org/10.1109/ICDE.2019.00115.
[19] M. Gao, L. Xiang, J. Gong, Organizing large-scale trajectories with adaptive geohash-tree based on secondo database, in: 2017 25th International Conference on Geoinformatics, 2017, pp. 1–6, http://dx.doi.org/10.1109/GEOINFORMATICS.2017.8090926.
[20] O. Alqahtani, T. Altman, An adaptive large-scale trajectory index for cloud-based moving object applications, in: Advanced Information Networking and Applications, Springer, Cham, 2021, pp. 80–94, http://dx.doi.org/10.1007/978-3-030-75075-6_7.
[21] S. Wang, D. Maier, B.C. Ooi, Fast and adaptive indexing of multi-dimensional observational data, Proc. VLDB Endow. 9 (14) (2016) 1683–1694, http://dx.doi.org/10.14778/3007328.3007334.
[22] C. Li, Z. Wu, P. Wu, Z. Zhao, An adaptive construction method of hierarchical spatio-temporal index for vector data under peer-to-peer networks, ISPRS Int. J. Geo-Inf. 8 (11) http://dx.doi.org/10.3390/ijgi8110512.
[23] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, E. Keogh, Querying and mining of time series data: Experimental comparison of representations and distance measures, Proc. VLDB Endow. 1 (2) (2008) 1542–1552, http://dx.doi.org/10.14778/1454159.1454226.
[24] X. Wang, A. Mueen, H. Ding, G. Trajcevski, P. Scheuermann, E. Keogh, Experimental comparison of representation methods and distance measures for time series data, Data Min. Knowl. Discov. 26 (2) (2013) 275–309, http://dx.doi.org/10.1007/s10618-012-0250-5.
[25] K. Buchin, M. Buchin, C. Wenk, Computing the fréchet distance between simple polygons, Comput. Geom. 41 (1) (2008) 2–20, http://dx.doi.org/10.1016/j.comgeo.2007.08.003, special Issue on the 22nd European Workshop on Computational Geometry (EuroCG).
[26] D.J. Berndt, J. Clifford, Using dynamic time warping to find patterns in time series, in: Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining, AAAIWS'94, AAAI Press, 1994, pp. 359–370.
[27] L. Chen, M.T. Özsu, V. Oria, Robust and fast similarity search for moving object trajectories, in: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data, SIGMOD '05, Association for Computing Machinery, New York, NY, USA, 2005, pp. 491–502, http://dx.doi.org/10.1145/1066157.1066213.
[28] M. Vlachos, G. Kollios, D. Gunopulos, Discovering similar multidimensional trajectories, in: Proceedings 18th International Conference on Data Engineering, 2002, pp. 673–684, http://dx.doi.org/10.1109/ICDE.2002.994784.
[29] S. Wang, Z. Bao, J.S. Culpepper, T. Sellis, X. Qin, Fast large-scale trajectory clustering, Proc. VLDB Endow. 13 (1) (2019) 29–42, http://dx.doi.org/10.14778/3357377.3357380.
[30] M. Shen, D.-R. Liu, S.-H. Shann, Outlier detection from vehicle trajectories to discover roaming events, Inform. Sci. 294 (C) (2015) 242–254, http://dx.doi.org/10.1016/j.ins.2014.09.037.
[31] J.-G. Lee, J. Han, X. Li, H. Gonzalez, Traclass: Trajectory classification using hierarchical region-based and trajectory-based clustering, Proc. VLDB Endow. 1 (1) (2008) 1081–1094, http://dx.doi.org/10.14778/1453856.1453972.
[32] C.A. Ferrero, L.O. Alvares, W. Zalewski, V. Bogorny, Movelets: Exploring relevant subtrajectories for robust trajectory classification, in: Proceedings of the 33rd Annual ACM Symposium on Applied Computing, SAC '18, Association for Computing Machinery, New York, NY, USA, 2018, pp. 849–856.
[33] J. Bian, D. Tian, Y. Tang, D. Tao, Trajectory data classification: A review, ACM Trans. Intell. Syst. Technol. 10 (4) http://dx.doi.org/10.1145/3330138.
[34] F. Giannotti, M. Nanni, F. Pinelli, D. Pedreschi, Trajectory pattern mining, in: Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Association for Computing Machinery, New York, NY, USA, 2007, pp. 330–339, http://dx.doi.org/10.1145/1281192.1281230.
[35] Q. Fan, D. Zhang, H. Wu, K.-L. Tan, A general and parallel platform for mining co-movement patterns over large-scale trajectories, Proc. VLDB Endow. 10 (4) (2016) 313–324, http://dx.doi.org/10.14778/3025111.3025114.
[36] D. Zhang, M. Ding, D. Yang, Y. Liu, J. Fan, H.T. Shen, Trajectory simplification: An experimental study and quality analysis, Proc. VLDB Endow. 11 (9) (2018) 934–946, http://dx.doi.org/10.14778/3213880.3213885.
[37] C. Long, R.C.-W. Wong, H.V. Jagadish, Direction-preserving trajectory simplification, Proc. VLDB Endow. 6 (10) (2013) 949–960, http://dx.doi.org/10.14778/2536206.2536221.
[38] S. Wang, Z. Bao, J.S. Culpepper, G. Cong, A survey on trajectory data management, analytics, and learning, ACM Comput. Surv. 54 (2) http://dx.doi.org/10.1145/3440207.
[39] T. Vu, A. Eldawy, V. Hristidis, V. Tsotras, Incremental partitioning for efficient spatial data analytics, Proc. VLDB Endow. 15 (3) (2021) 713–726, http://dx.doi.org/10.14778/3494124.3494150.
[40] S. Leutenegger, M. Lopez, J. Edgington, Str: a simple and efficient algorithm for r-tree packing, in: Proceedings 13th International Conference on Data Engineering, 1997, pp. 497–506, http://dx.doi.org/10.1109/ICDE.1997.582015.
[41] T. Vu, A. Eldawy, R*-grove: Balanced spatial partitioning for large-scale datasets, Front. Big Data 3, http://dx.doi.org/10.3389/fdata.2020.00028.
[42] Tiger/Line, https://www.census.gov/cgi-bin/geo/shapefiles/index.php.
[43] E. Horowitz, S. Sahni, Computing partitions with applications to the knapsack problem, J. ACM 21 (2) (1974) 277–292, http://dx.doi.org/10.1145/321812.321823.
[44] K. Toohey, M. Duckham, Trajectory similarity measures, SIGSPATIAL Special 7 (1) (2015) 43–50, http://dx.doi.org/10.1145/2782759.2782767.