World Wide Web (2018) 21:455–485
DOI 10.1007/s11280-017-0466-9

# SharkDB: an in-memory column-oriented storage for trajectory analysis

**Bolong Zheng[1] · Haozhou Wang[2] · Kai Zheng[3] · Han Su[4] · Kuien Liu[2] · Shuo Shang[5]**

Received: 24 October 2016 / Revised: 13 April 2017 /
Accepted: 27 April 2017 / Published online: 5 May 2017


**Abstract** The last decade has witnessed the prevalence of sensor and GPS technologies that produce a high volume of trajectory data representing the motion history of moving objects. However some characteristics of trajectories such as variable lengths and asynchronous sampling rates make it difficult to fit into traditional database systems that are disk-based and tuple-oriented. Motivated by the success of column store and recent development of in-memory databases, we try to explore the potential opportunities of boosting the performance of trajectory data processing by designing a novel trajectory storage within main memory. In contrast to most existing trajectory indexing methods that keep consecutive

✉ Kai Zheng
zhengkai@suda.edu.cn

Bolong Zheng
b.zheng@uq.edu.au

Haozhou Wang
hawang@pivotal.io

Han Su
suan.sue@gmail.com

Kuien Liu
kliu@pivotal.io

Shuo Shang
jedi.shang@gmail.com

[1] The University of Queensland, Brisbane, QLD, Australia

[2] Pivotal Incorporated, San Francisco, CA, USA

[3] School of Computer Science and Techonology, Soochow University, Suzhou, China

[4] Big Data Research Center, University of Electronic Science and Technology of China, Chengdu, China

[5] King Abdullah University of Science and Technology, Thuwal, Saudi Arabia

samples of the same trajectory in the same disk page, we partition the database into frames in which the positions of all moving objects at the same time instant are stored together and aligned in main memory. We found this column-wise storage to be surprisingly well suited for in-memory computing since most frames can be stored in highly compressed form, which is pivotal for increasing the memory throughput and reducing CPU-cache miss. The independence between frames also makes them natural working units when parallelizing data processing on a multi-core environment. Lastly we run a variety of common trajectory queries on both real and synthetic datasets in order to demonstrate advantages and study the limitations of our proposed storage.

**Keywords** Spatial database · Trajectory · In-memory · Storage

## 1 Introduction

Driven by rapid development in sensor technology, GPS-enabled mobile devices and wireless communications, large amounts of data describing the motion history of moving objects, known as *trajectories*, are currently generated with unprecedented rate from a variety of application domains such as geographical information systems, location-based services, vehicle navigation, video tracking and so on. This calls for effective and efficient technologies to manage large scale trajectory data, which serves as a corner stone for more advanced data analytical tasks.

Even though spatial databases have been extensively studied as a research area for decades with several successful commercializations,[1] they were designed only to support basic spatial types such as points, lines and polygons. Trajectories, on the other hand, are not easy to fit into a relational table with pre-defined schema since each tuple (i.e., trajectory) has different number of attributes (i.e., time-stamped points). To fix this problem, we can simply treat the entire sequence of points as a single attribute and store them in one column. Nevertheless this kind of storage will severely deteriorate query performance since it is almost impossible to utilize the spatio-temporal locality amongst trajectories. Having witnessed the limitations of existing spatial database systems in the past decade, researchers have dedicated lots of efforts in proposing novel techniques for trajectory data management. What lie in the core of these techniques are trajectory indexes, the basic design principle of which is grouping trajectory segments that are close to each other and putting them in the same node (in tree-based index) or cell (in grid index).

Owing to the development of cheap RAM-based storage technology, modern computing hardware can afford much larger main memory. It provides lots of potential opportunities to significantly improve the efficiency of big data management by shifting more or even all data from disk-based storage into main memory. This triggers numerous research interests recently on in-memory data management from both database and data mining communities, ranging from memory-based indexing techniques [33] to in-memory database systems [31]. However, existing main-memory based database systems are designed for relational data, which is not suitable for storing and querying the trajectory data that possess both spatial and temporal information.

[1] Many database management systems offer additional components or extensions to support spatial types and operators such as Oracle, MySQL, PostgreSQL, etc.

In this work, we propose an in-memory column-oriented trajectory storage, which is a read-optimized structure for storing and processing massive trajectory data. This storage combines the merits of high throughput of main memory and benefits of column-wise store for analytical tasks. Meanwhile, our design also supports effective compression and efficient query processing on trajectory data.

In order to take advantage of column-oriented data structure and compression techniques for storing and analysing trajectory data, we foresee two main challenges. The first one is how to convert trajectory data into column-oriented data structure, which not only supports varied length and multi-dimensional trajectory data, but also performs efficient query processing in the main memory. The second one is how to deploy compression techniques on the column-oriented data structure, which supports query processing on compressed data without sacrificing much performance.

We address the above challenges by a carefully designed data structure with two-phase data processing, which combines both column-oriented data structure and compression techniques to store and analyse a huge amount of trajectory data in an in-memory database system. More specifically, trajectory allocation phase will leverage the trajectory calibration techniques [38] to calibrate the trajectory data and convert the calibrated trajectory data into column-oriented data structure. In the second phase, a trajectory encoding component compresses the trajectory data and stores them in column-oriented data structure such that an efficient query processing can be supported.

Our previous work [42] has demonstrated the efficiency of proposed data structure for trajectory query data processing. However, the algorithms proposed in previous work are only limited for point-based and region-based trajectory queries, where the query is a point ($k$NN query) and a region (window query). Yet there is another type of trajectory query called trajectory similarity search still remaining to study, which looks for a set of trajectories that are similar to a given trajectory. It is widely used in many applications such as trajectory pattern mining, trip planning and event detection. Therefore it is critical to develop an algorithm to support trajectory similarity search efficiently.

Our key contributions in this work can be summarized as follows.

- We identify limitation of relational database system in handling large trajectory data.
- We develop a novel column-oriented in-memory trajectory storage, which can support compact trajectory storage in the main memory and efficient trajectory data processing.
- We propose novel techniques to process trajectory similarity search efficiently.
- We deploy the system and conduct extensive experiments with several synthetic and real world large trajectory data sets. The results demonstrate that our system can achieve a high performance compares with traditional database structures, and can offer the promise of real time computing.

The rest of this paper is organized as follows. A review of related work is provided in Section 2. In Section 3, we present an overview of our storage design. Sections 4, 5 and 6 present the details of storage structure and query processing algorithms. Section 7 reports on our experimental observations. Section 8 concludes the paper.

## 2 Related work

In this section, we review previous work on column-oriented data structure, in-memory data management, trajectory storage and indexing.

### 2.1 Column-oriented store architecture

Boncz et al. [8] develop a modern in-memory database system, called MonetDB, with the MIL query language to fully support the column-oriented data structure. On the other hand, the row-oriented architecture is suitable on OLTP (Online Transactional Processing) style applications, and Stonebraker et al. [37] indicate that system oriented towards ad-hoc querying of large amounts of data should be read-optimized. Therefore, Héman et al. [22] propose a new column-oriented data structure, called Positional Delta Tree (PDT). To improve query performance, Lemke et al. [27] propose a new algorithm to process query such as scan and aggregation operations on the compressed column-oriented data structures. Ivanova et al. [23] study a new architecture to provide an recycler intermediate in the column-oriented data structures, which is using a lightweight mechanism. Krueger et al. [26] present a linear merge algorithm to utilize the power of parallel computing for fast updating in the compressed in-memory column-oriented structures. Unfortunately, although these column-oriented data structures based storage systems are very powerful, they did not consider to store the spatial-temporal data such as trajectory data, which is a natural multi-dimensional data type that combines with unstructured data format. Hence, using those storage systems to store the trajectory data directly will significantly reduce the performance of query processing.

### 2.2 In-memory data management

There are several in-memory database systems have been proposed or implemented. IMS/VS Fast Path [18] is one of the earliest commercial database product. It uses the group committed for transactions to support productivity. An other earlier version of in-memory database [1, 7] based on IBM's OBE system uses points to speed up the ad-hoc transaction queries. Baulier et al. [3] implement a commercial in-memory database system, which is called DataBlitz Storage Manager. Plattner [31] indicates that in-memory database outperforms the traditional database system for both OLTP (Online Transactional Processing) and OLAP (Online Analytical Processing). Subsequently, [32] implements a new column-oriented based in-memory database. Bining et al. [6] propose a dictionary-based order-preserving string compression algorithm in the in-memory column system. On the other hand, Rao and Ross [33] propose CSB+-trees as the main-memory index structure, which is a CPU cache optimized method and is improved from B+-trees. At the same time, a cost model for in-memory database system is proposed by Manegold et al. [28], which provides an accurate cost function for database operations. Due to the underlying use of the relational model in these in-memory database systems, analytical trajectory queries cannot be well-supported.

### 2.3 Trajectory storage and indexing

The most popular and classical data structure for spatial data is R-tree [16, 19, 20]. However, directly applying R-tree on spatial dimension and temporal dimension is not good enough. According to [30], we know approximating the line segments of trajectories with MBBs (Minimum Bounding Box) in R-tree is problematic, since large amounts of "dead space" are introduced where the MBB covers a large portion of space but the actual space occupied by the trajectory is small. Therefore, many optimizations are proposed to make the R-tree based structures support the trajectory data [16, 19, 21]. Gowanlock and Casanova [19] proposes a method to split trajectories into smaller segments such that the R-tree can be

used without too many "dead space". This is in fact similar to our work of splitting trajectories into columns. Although trajectory splitting is an option for the R-tree, we do not compare it with our method since it is not the major focus of our work. TB-tree [30] uses a hybrid tree structure to store and index both spatial and temporal information, but is not adequate to process long trajectories, which makes the bounding rectangles very large. TPR-tree [36] and TPR*-tree [39] invoke the predication model to predict the future positions of moving objects. On the other hand, partitioning trajectories into segments becomes a new way to improve the query performance. Rasetic et al. [34] derive an analytical cost model to control the splitting process for a trajectory into segments based on given query. SETI [10] stores trajectory segments in a 3D R-tree for their spatial information. Meanwhile, SETI indexes the temporal information by using one dimensional time lines to increase the search performance. PIST [9] partitions the sample points rather than partitioning the trajectories. Similarly, the TrajStore [13] proposes a new adaptive storage system that indexes the trajectory data based on quad-tree index and clustering methods. These algorithms or systems are designed for disk based systems, which means I/O cost between hard disk and memory is the main concern. However, there is no I/O cost in in-memory based systems. Thus these algorithms and systems are not feasible for in-memory environment.

### 2.4 Trajectory similarity measurement

In a sense, the trajectory data is a kind of time-series data with spatial value. A fundamental ingredient of trajectory analysis tasks are the distance/similarity measurements that allow to effectively determine the similarity of trajectories. But unlike other simple data types such as ordinal variables or geometric points where the distance definition is straightforward, the distance between trajectories needs to be carefully defined in order to reflect the true underlying similarity. This is due to the fact that trajectories are essentially high dimensional data attached with both spatial and temporal attributes, which needs to be considered for similarity measures. As such, there are over distance measures for similarity of trajectory or time series data in the literature, e.g., Euclidean distance (ED) [14], Dynamic Time Warping (DTW) [5, 24], distance based on Longest Common Subsequence (LCSS) [41], Edit Distance with Real Penalty (ERP) [11], Edit Distance on Real sequence (EDR) [12], DISSIM [17], and similarity search based on Threshold Queries (TQuEST) [2]. Many of these works and some of their extensions have been widely cited in the literature and applied to facilitate query processing and data mining of trajectory data. However, these similarity measurements are not suitable for the large scale trajectory data. The main issue is that a trajectory may contain several months or even several years records for only one moving object. In practical, most similarity trajectory analytic tasks are focusing on particular time period. Therefore, these similarity measurements cannot return useful results for our trajectory analytic tasks especially when using large scale dataset.

## 3 Storage architecture

The storage architecture is illustrated in Figure 1, which includes two main parts: the real-time query processing component and the in-memory data storage. The in-memory data storage system encodes the raw historical trajectories [15] and stores them in a re-designed column-oriented data structure. The real-time query processing component is able to support

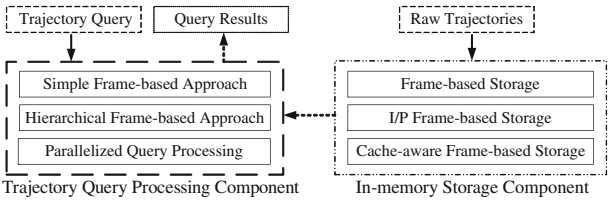



**Figure 1** Storage architecture

queries including window query, nearest neighbour query and trajectory similarity query in real time. We give a brief overview of these two parts in the following:

**In-memory storage** In the in-memory data storage component, we create a novel frame based column-oriented data structure to store the trajectory data. Hence, we propose three storage models, frame-based storage, I/P frame-based storage and cache-aware frame based storage, to convert and calibrate the raw trajectory data into this new frame based data structure. The last two models also embed trajectory compression algorithm to reduce the data footprint in the memory. We will discuss this part in Section 4.

**Real-time query processing system** We implement three query processing approaches to support real-time query processing in the system. The simple frame based approach is working over frame data structure; the hierarchical frame based approach builds a multi-level frame data structure from in-memory storage component to achieve better performance. Moreover, we extend previous approaches to allow parallel query processing. We will discuss this part in Sections 5 and 6.

## 4 In-memory storage

In this section, we detail these three data structures as well as their encoding algorithms to store the raw trajectory data into in-memory column-oriented data structure.

### 4.1 Frame-based storage

Different from basic operations (i.e., selection or deletion), most analysis tasks, such as window query and nearest neighbour query, only need to touch few trajectories to get answer. Therefore the row-oriented data structure is not good enough for such tasks, since scanning the whole table is hard to be avoid during query processing without indexing the data. In addition, the column-oriented data structure is known to have better performance in analysis task that compares with row-oriented data structure [31]. To get this advantage, we propose a novel frame based column-oriented data structure to store trajectory data in main memory. Thus, we create a sequence of frames from trajectory data directly. In each frame, we assign a particular time interval, then allocate each trajectory sample point to the related frame based on its recorded time. Therefore, a frame contains all the trajectory sample points that are recorded at the time interval of the frame. Such time interval is called frame rate.

For instance, as Figure 2 shows, if we set the time interval as 1 minute (i.e. the frame rate is 60 seconds), the time period from 9:01 to 9:05 is split into 4 frames.

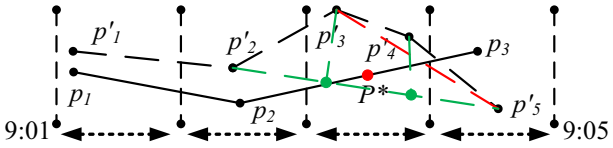



**Figure 2** Trajectory snapshot

To keep simple and tidy, each frame contains at most one sample point for each trajectory. To make this frame structure continuous in temporal space, each frame must be strictly synchronized by same time period (i.e., with same frame rate). Normally, the frame rate will be as same as sampling rate of trajectory. However, a trajectory in a raw trajectory dataset may have different sampling rates compared with other trajectories due to unstable GPS signal. Therefore, this situation makes the whole trajectory dataset become heterogeneous, which may affect the frame structure. To avoid this issue, if there are more than one sample point that belong to a same trajectory, they are aligned to the same frame. Then we calculate the SED [29] distance of each sample point, and keep the sample point with largest SED distance and remove the rest sample points in the frame. This is because the sample point with largest SED distance contains more information than the other points. In Figure 2, $p'_3$ and $p'_4$ are aligned into $Frame3$. According to SED, it partitions the line segment between $p'_2$ and $p'_5$ into 3 equal-length segments. Since the SED distance between $p'_3$ to its corresponding segment point is greater than that of $p'_4$, $p'_3$ is kept and $p'_4$ is removed. In addition, if there is no sample point of a trajectory locates in a frame, we use line interpolation method to add a new sample point of the trajectory into this frame. In Figure 2, we insert $p^*$ into $Frame3$. Finally, we conduct a set of accuracy experiments in section 6 to show the influence of heterogeneous trajectories.

After allocating sample points into frames, converting each frame into column-oriented data structure becomes straightforward. Basically, we only need to insert each frame as a single column into the in-memory database. The ID of each column is named as its own time interval (e.g., 9 : 01-9 : 02) as Table 1 shows. In addition, each sample point (i.e., the object) in column contains its trajectory ID, longitude and latitude. After aligning, we do not need the timestamp information of a sample point any more, since the temporal information can be recovered by the column ID. The advantage is that our frame data structure is a natural column-oriented data structure and synchronized by time. Therefore, storing this data in the column-oriented structure is simple and elegant. We may notice that the interpolation of points can require an increase in memory. If significant interpolation occurs, then the benefits of a compressed data structure may provide limited performance gains. However, if the most of trajectories are collected in high sampling rate, this issue may not affect the performance too much.

**Table 1** Example of frame based structure

| 9:01−9:02 | 9:02−9:03 | 9:03−9:04 | 9:04−9:05 |
|---|---|---|---|
| Frame 1 | Frame 2 | Frame 3 | Frame 4 |
| $(1,p'_1)$ | $(1,p'_2)$ | $(1,p'_3)$ | $(1,p'_4)$ |
| $(2,p_1)$ | $(2,p_2)$ | $(2,p_3)$ | $(2,p_4)$ |

### 4.2 I/P Frame-based storage

For in-memory database, it is good to utilize compression techniques to reduce the size of data and improve query performance, since the compressed data can reduce the footprint of data in the memory, which in turn reduces the searching time in the memory. For trajectory compression, we consider delta encoding technique to compress the trajectory data, which keeps information for the first point and using the $\delta$ value to record the information for the rest of points. Therefore, the delta encoding will not change the sampling rate of trajectories and it allows the storage system to use less bits to store the trajectory data. However, a problems is raised when using delta encoding. For each column, the type of object (point) must be the same, which means the objects in a single column can only be original point or delta encoded point. However, the start time of trajectory and the length of trajectory is arbitrary, so we can not simply use delta encoding. Therefore, we propose a novel I/P frame based data structure to reduce the memory consumption inspired by the idea of video compression picture types [35].

Building an I/P frame data structure includes grouping process and encoding process. First of all, we split the whole sequence of frames into small groups, called frame group, and each frame group contains $n$ continuing frames. Based on the example in Figure 1, if we set $n = 4$, and the $Frame1$ to $Frame4$ are allocated in a group.

After encoding phase, for each group, we keep the first frame raw information as an **I-frame**, and the subsequent frames, called **P-frame**, use delta encoding to be compressed.

The example is shown in Table 2, the $Frame1$ is kept as I-frame $IF_1$. The rest of the frames $Frame2$ to $Frame4$ as $P_1$ to $P_3$, then we encode the sample points in $P_1$ by calculating the difference between $IF_1$ and its related sample points in $P_1$ as a P-frame point $P_1.PF_1$. Again, we continue encoding the sample points in $P_2$ and $P_3$ as P-frame points $P_2.PF_1$ and $P_3.PF_1$ respectively. At the end, we get one I-frame column and 3 encoded P-frames columns as shown in Table 2.

However, this I/P frame encoding algorithm still has some drawbacks. For reconstruction of a segment of a trajectory contained in a frame group, current solution has to scan all of columns in that group, which includes both I-frame and P-frame. It is easy to see this process is expensive, since it has to scan too much memory. Therefore, the performance of query processing can be further reduced since it does not fully utilize the power of column-oriented data structure.

### 4.3 Cache-aware frame-based storage

To improve the trajectory reconstruction performance, the number of sequential scans for columns should be minimized during query processing. The idea of avoiding sequential scan in all columns in a frame group is to reduce the number of scans in P-frame, since the P-frames are highly related to I-frames. Therefore we fix the length of both I-frame column

**Table 2** Example of I/P-frame structure

| Frame 1 | Frame 2 | Frame 3 | Frame 4 |
|---|---|---|---|
| (1,$p$) | (1,$\Delta p$) | (1,$\Delta p$) | (1,$\Delta p$) |
| (2,$p$) | (2,$\Delta p$) | (3,$\Delta p$) | (4,$\Delta p$) |
| (3,$p$) | (3,$\Delta p$) | (4,$\Delta p$) | |
| (4,$p$) | (4,$\Delta p$) | | |

and P-frame columns in a group, which means the length of each P-frame column equals to the length of I-frame column. As a result, for the same segment of trajectory in a frame group, each P-frame point of that trajectory shares the same index with the point in I-frame. Meanwhile, if a trajectory segment has not fulfilled a frame group such as a trajectory segment has 1 I-frame point and 2 P-frame points in a $n = 4$ frame group, we use a $Nil$ code as a place-holder in the rest of P-frames. Hence, to reconstruct a trajectory segment, we only scan the points in I-frame and access the P-frame points in P-frames directly in memory by adding the offset from I-frame point. Moreover, in each P-frame point, the trajectory ID is no longer needed, since it can be recovered from I-frame point. For example, as Table 3 shows, to reconstruct trajectory $T_4$, we scan I-frame first to get its I-frame point, then we access the related P-frame points directly by adding an offset from the beginning address of each P-frame column.

Moreover, in modern architectures of computer system, accessing data from memory to CPU is via a hierarchical memory structure. Typically, CPU has built-in cache memories, which is connecting to system main memory directly. On the other hand, the reading action from memory to CPU is parametrized by CPU cache line size (typical is 64 bytes), which is the basic transferring unit. For a reading action, CPU will read 64 bytes data from memory no matter these data could be fully used or not. Therefore, to increase the performance, it requires to fill up the CPU cache line as much as possible with useful data for each memory access to reduce the total number of accessing from CPU. However, this task is challenging as we need to store the P-frame point of a trajectory segment continuously and cannot break the column-oriented data structure at the same time.

Therefore, we use a hybrid data structure to store the P-frames in a frame group as they are highly related. Our hybrid data structure is based on a two dimensional array, which is created as an $array[x][y]$, where $x$ equals to the number of I-frame points in this frame group, and $y$ equals to the number of P-frame columns in this frame group. So, the value of $x$ in array is the index of I-frame points; the value of $y$ indicates the column number of P-frame group; and the object at $array[x][y]$ is the P-frame point. For example, $array[1][2]$ is the P-frame point $PF_{1,2}$, which is shown in Figure 3, and 1 for first I-frame point and 2 for second P-frame column. As a result, CPU can fetch an I-frame point with its related P-frame points and fill in a cache line block by a single memory accessing. When CPU tries to access the next P-frame point, it is already in the CPU cache, and CPU can access them directly without the need to search memory again. Consequently, it increases the performance of query processing. For example, as Figure 3 shows, assuming a memory access reads 2 points vertically from P-frame array. It will cost two memory seeking requests to decode a trajectory segment in a frame group for non-cache optimization data structure, since the other points in the cache line is not related to this segment and is not usable. On the other hand, it only needs to cost one memory access for decode this trajectory segment in the cache optimized data structure, since the related four points can be read from CPU cache directly.

**Table 3** Example of cache-aware frame-based structure

| 9:01−9:02 | 9:02−9:03 | 9:03−9:04 | 9:04−9:05 |
|---|---|---|---|
| $(1,p)$ | $(1,\Delta p)$ | $(1,\Delta p)$ | $(1,\Delta p)$ |
| $(2,p)$ | $(2,\Delta p)$ | Nil | Nil |
| $(3,p)$ | $(3,\Delta p)$ | $(3,\Delta p)$ | Nil |
| $(4,p)$ | $(4,\Delta p)$ | $(4,\Delta p)$ | $(4,\Delta p)$ |

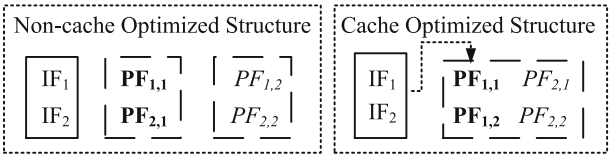



**Figure 3** Example of cache-aware I/P-frame structure

## 5 Basic trajectory query processing

There are two fundamental query types, window query and top-$k$ nearest neighbour ($k$NN) query. Formally, a trajectory $T$ is a sequence of points $\{p_1, \ldots, p_n\}$ where $p_i.t$ is the timestamp of $p_i$. For window query, it finds all trajectories in the data set that are active during a given period and passing a given region. For $k$NN query, it finds top-$k$ trajectories in the trajectory data set that are close to a given point and active during a given period of time.

**Definition 1** (**Window Query**) Given a time period $[HS, HE]$ and a query range $R$, a trajectory or a sub-trajectory $T = \{p_1, \ldots, p_n\}$ is reported as result if and only if $\forall p_i \in T$, we have $p_i$ locates inside $R$ and $HS \leq p_i.t \leq HE$.

**Definition 2** ($k$**NN Query**) Given a time period $[HS, HE]$ and a query point $q$, the $k$NN query reports $k$ trajectories or sub-trajectories that have the smallest distances to $q$, and in the meantime for each result $T = \{p_1, \ldots, p_n\}$, $\forall p_i \in T$ we have $HS \leq p_i.t \leq HE$. The distance between $q$ and $T$ is the minimum perpendicular distance from $q$ to all line segments $\overline{p_i p_{i+1}}$ in $T$.

In this section, we propose simple frame based approach and hierarchical I/P frame data structure based approach proposed to answer these queries. A parallel version for such approaches is discussed in Section 5.3.

### 5.1 Simple frame based approach

For query processing, it is necessary to reduce the number of reconstruction trajectory segments in the frame group as much as possible. In other words, the unnecessary frame groups should be pruned as much as possible. Therefore, we deploy a two-phase processing that includes pruning and refining. In the pruning phase, we first filter out the frame group columns, which are out of the given time period. For I-frame point (i.e., original trajectory sample point) in each frame group, we calculate the maximum moving distance $D_{max}$ for its P-frame points members. The $D_{max}$ equals to user defined maximum possible moving speed multiple the time duration of this frame group. Then we use $D_{max}$ as the boundary value of this trajectory segment for pruning based on different query types (i.e., Window query or NN query). In the refining phase, we decode and reconstruct trajectory segments from I-frame point with its related P-frame points and find the final answer based on given query type. More specifically, we detail the difference of query processing between window query and kNN query.

**Window query** For window query, at the pruning phase, we use $D_{max}$ to estimate the moving area of its related frame group (i.e., trajectory segment). If the moving area is overlapping with query window, we decode the P-frame points to find the actual answer of window query.

***k*NN query** For the $k$NN query, we calculate the distance from I-frame point to query point and minus the $D_{max}$ as the lower bound. If the lower bound is greater than the current $k^{th}$ best true distance, which means this trajectory segment cannot be closer than current best examined trajectory, we can prune it safely. Otherwise, we put this trajectory segment into refining phase to find its true distance to the query point. Moreover, since we are processing on each frame group (i.e., the trajectory segment), not for the complete trajectory, we need to avoid the different frame groups with same trajectory ID to be pushed into result set. This situation can lead several trajectory segments with same trajectory ID (i.e., these segments belongs to one trajectory) to be consider as final results, which causes the number of final results (i.e., the returned trajectories) to be less than $k$, thus makes this query fail. Therefore, we extend the priority heap to a hashed priority heap, in which its elements are hashed by trajectory ID. When updating a new frame group, the system will first check the elements in the results set whether there is an element containing same trajectory ID as the new frame group. If it exists, system will only update the existing element by using this new frame group rather than creating a new element in the results set.

## 5.2 Hierarchical frame based approach

During the query processing, the most time consuming part is the sequential searching in I-frame columns. To reduce the searching time, we propose a new hierarchical frame structure that expands from simple frame based approach.

As Figure 4 shows, we build this structure from bottom to top based on the sample I/P frame data structure, which is used to reduce the number of I-frame columns visited during query processing. To reduce the complexity of the whole multi-layer structure, we still use frame group as the basic unit in the structure. First of all, we extract the I-frame columns from current top level to build a new I/P frame structure layer, which becomes an upper level of current level. Then we assign every $n$ I-frame columns into a frame group, where $n$ is the number of frames in a frame group. Afterwards, we use the same encoding methods

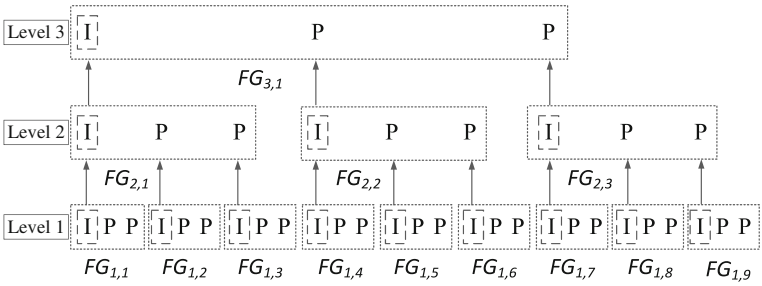



**Figure 4** Hierarchical I/P-frame data structure

that are proposed in the Section 3 to encode the I-frame columns in the new frame groups. That means, in each new frame group, we keep the first I-frame column unchanged as the new I-frame column and re-encode the rest I-frame columns as the new P-frame columns. Figure 4 shows the process to build level 2 structure from level 1 structure and build level 3 structure from level 2 structure respectively. During the processing, we keep building new layers until the time duration of frame group becomes greater than the average time duration of trajectories at current level. This is because if we do not stop building at this level, most trajectories will be represented as a single frame point at the upper level, and such level cannot assist in improving query processing.

Although the encoding technology reduces the space consumption substantially, keeping the I-frame columns information in each level is still space consuming and will limit the number of layers in hierarchical structure due to the memory space limitation. When we build a new layer, for each new I-frame point that needs to copy from lower level, we only copy the memory address from original I-frame point instead of copy full information. The full information of such I-frame points are kept in the bottom level only. Therefore, we can reduce the memory consumption by using this data structure. In addition, we use the same strategy to copy/encode new level P-frame points. Hence, when algorithm accesses such P-frame point, it gets the information directly without need to decode this P-frame point, and increases traversing speed of algorithm.

At the same time, in the high level of hierarchical structure, the frame points of each trajectory become sparse, which leads to not only making the time duration between each frame column very large, but also the distance between two frame point becomes greater. Hence, it is easy to see the bound $D_{max}$ will be too large to do the pruning. This is because a large value of $D_{max}$ will increase the searching space dramatically, which reduces the performance of query processing or even makes it as same as the exhausted search. To solve this issue, for each frame group started at the bottom level, we calculate its MBR information and embed it into its I-frame point to instead of calculating the $D_{max}$ during query processing. As the upper level MBR information is calculated based on the MBR information of its lower level frame groups, so it covers all the MBRs in the lower level frame groups used to build this frame group. For example, the MBR of $FG_{2,1}$ is the superset of MBRs that include frame groups from $FG_{1,1}$ to $FG_{1,3}$.

In addition, since the structure of each layer in whole hierarchical structure is the same, we modify our two phase processing with only a few changes. The best first algorithm is used for traversing the hierarchical structure. First, we initiate a priority heap and push the first level frame groups within query time range into it. Then if a popped frame group is already in the bottom level, we use the algorithm same as simple frame based approach to update the results set. Otherwise, we travel into next level by decoding the popped frame group directly, and filter out decoded frame groups in next level that are out of the time range in query. Finally, we examine filtered frame groups and update the priority queue. Now we detail the process for each type of query:

**Window query** For each selected frame group, we check the MBR information in its I-frame point. If the MBR is overlapping with query window, we decode them. For example, in Figure 4, if frame group $FG_{3,1}$ at level 3 is popped from candidate list, we then select frame group from $FG_{2,1}$ to $FG_{2,3}$ at level 2, which are used to build frame group $FG_{3,1}$. The refining phase is same as simple frame based approach.

***k*NN query** For the $k$NN query, we use the minimum distance from MBR to query point as the new lower bound for pruning phase, and then we calculate its lower bound and push

it into the priority heap. If a frame group is already in the bottom level, we calculate its true distance from query point to this frame group. If the true distance is less than best so far distance, we push this frame group with its true distance into our hashed priority heap as update in the result set.

### 5.3 Parallel computing

A general configuration of modern servers typically contains multiple CPUs with multi-cores built-in in each CPU. Therefore, it now becomes the standard that supports parallel computing for query processing, which can fully release the power of parallel computing. The problem of paralleling query processing is like a divide and conquer problem, for instance, how to divide a query into sub-queries to send to different threads and how to combine the sub-results into final result. Based on the advantage of I/P frame data structure, where each frame group is highly independent and "share nothing" with each others, the divide and conquer process is simple, and very stable. First, we create a thread pool to manage threads. Then we assign each frame group column into different threads to process and push results to the queue. Afterwards, we do parallel decoding to get the next level frame groups from popped frame group and assign them into different thread. In each single thread, we run the algorithm proposed in hierarchical frame based approach to check and keep traversing strategy. Finally, we merge the results from each single thread. Now we detail our approach for parallel window query, and parallel kNN query.

**Window query** To store the results in parallel, we create an $array[x][y]$ to organize the result trajectories; and each $array[x]$ represents a result trajectory and each $array[x][y]$ represents a sample point of a result trajectory. If a thread finds its results containing new trajectories (new IDs), it creates new arrays in this array. When creating the new array for a new result, this thread locks this object to block new array creation request from other threads until finished, which is to avoid creating multiple arrays for one trajectory.

***k*NN query** The challenge of parallelization kNN query is that the value of the best so far (the minimum distance from current candidate trajectory to query point) needs to keep updating efficiently to shrink the spatial search area during query processing. Therefore, during query processing, each single thread uses the current best so far value for pruning and returns a new trajectory id with its distance if any better candidate is found. Once a single thread finds a better candidate, it first applies a lock on both the best so far variable and the hashed priority queue. Then, this thread updates the best so far value if the distance of this candidate is better than current value, and updates the hashed priority queue. At the end, this thread releases the locks and sends a signal to system to wait for assigning next frame group. System continues assigning frame group(s) to free threads in the thread pool until all the frame groups in the candidate list are investigated.

## 6 Trajectory similarity search

In this section, we first introduce the problem definition of trajectory similarity search, which considers both spatial and temporal information in Section 6.1. Then, we discuss the algorithms for processing the trajectory similarity search on the I/P frame structure in Section 6.2.

### 6.1 Problem definition

The general definition of trajectory similarity search is that given a query trajectory, the algorithm will find trajectories in trajectory database, which are similar with query trajectory (i.e., the similarity between query trajectory and result trajectories are high). In general, the similarity is calculated based on the distance between query trajectory and the trajectories in database, and the distance is calculated by similarity measures such as DTW (Dynamic Time Warping) [4, 43] and LCSS (Longest Common Subsequence) [40, 41] etc. However, this general definition is not suitable for the large scale trajectory data. There are two major issues, the first is that a trajectory may contain several months or even several years records for only one moving object. In practical, most similarity trajectory analytic tasks are focusing on particular time period, for example, finding the different pattern of traffics between the peak time and off-peak time everyday. Therefore, the traditional trajectory similarity search cannot return useful results for such trajectory analytic tasks when using large scale dataset. In addition, another issue is about the variable length of trajectory data. As we discussed before, the length of each trajectory in database is different. For example, if we compare the similarity between a two hours trajectory and a two months trajectory, it does not make any sense for similarity analysis. To solve these two issues, we extend and refine the definition of trajectory similarity search in the following, where we assume all of the trajectories are frame encoded.

**Definition 3** (**Trajectory Similarity Query**) Given a time period $[HS, HE]$, a query trajectory $T_q$ and a threshold $\alpha$, a trajectory similarity query is to find a set of trajectory segments $T_i S_j \in T_i$, where the length of $T_i S_j$ equals $T_q$ and $T_i$ is the original trajectory in trajectory database. Meanwhile, for each frame point $p_m^j \in T_i S_j$ and $p_m^q \in T_q$, the distance $d(p_m^j, p_m^q) < \alpha$, where $1 \leqslant m \leqslant l$ and $l$ is the length of $T_q$, and we call trajectory segment $T_i S_j$ matches $T_q$. Moreover, the time duration of both $T_i S_j$ and $T_q$ must be within given time period $[HS, HE]$.

Consequently, we do not use the minimum distance between two trajectories (i.e., the minimum distance of sample point pairs) to measure the similarity between two trajectories, since such measures are point based measures and cannot reflect the similarity between two trajectories. For instance, as Figure 5 shows, the trajectory $T_1$ is similar with $T_q$, however if we use the minimum distance to measure the similarity, the $T_2$ will be considered more similar than $T_1$ with $T_q$, since the minimum distance between $T_2$ and $T_q$ is less than $T_1$ and $T_q$. Therefore, this is the reason why we compare the distance for each point pair to make sure the result trajectories segment is similar with query trajectory practically. That means if a trajectory segment $TS$ is similar to $T_q$, the distance from $p_1^{TS}$ to $p_1^q$ must be less than $\alpha$

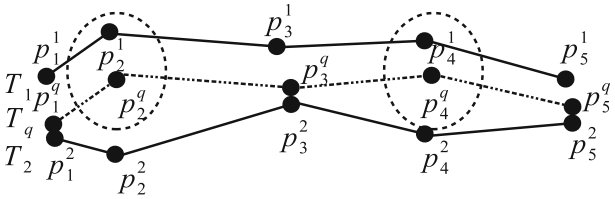



**Figure 5** Similar trajectory

and the distance from $p_2^{TS}$ to $p_2^q$ must be less than $\alpha$ and so on. Based on this measure, the trajectory $T_1$ now is selected as a result and $T_2$ is not, since the distance between $p_2^q$ and $p_2^2$ and the distance between $p_4^q$ and $p_4^2$ are larger than $\alpha$ (the $\alpha$ is shown as the circle).

We give more examples in Figure 6, assuming we have a trajectory $T_1$ and a query trajectory $T_q$. For similarity query, we set the time period $[HS, HE]$ from $t_1$ to $t_7$ and the threshold is $\alpha$, where $HS = t_1$ and $HE = t_7$. For trajectory $T_1$, we can see that a sub-trajectory from $p_1^1$ to $p_3^1$, which denotes as $T_1S_1$, and another sub-trajectory from $p_5^1$ to $p_7^1$, which denotes as $T_1S_2$, are match the query trajectory $T_q$. Therefore, both trajectory segments $T_1S_1$ and $T_1S_2$ are selected as the results for trajectory $T_1$. Moreover, if we change the time period from $t_1$ to $t_5$, although the sub-trajectory $p_5^1$ to $p_7^1$ matches the query trajectory, the timestamp of $p_6^1$ and $p_7^1$ is out of the given time period. Hence, the sub-trajectory $T_1S_2$ cannot be returned as a result.

## 6.2 Query processing on I/P frame structure

The naive way to solve this query is to decode the trajectory from frame structure first, and then find the results directly. It is easy to see that this naive solution is not an efficient way to process this query and does not utilize the advantages of the frame structure as well. Therefore, to keep the efficiency of the trajectory similarity search on the frame structure, we need to solve three challenges. The first challenge is to prune the unnecessary the trajectories (i.e., frame groups) effectively and efficiently, which means decoding frame should be avoided in pruning processing as it will cost extra running time. To calculate the actual distance, the candidate frame groups need to be decoded to original sample point. In this case, it raises the second challenge, which is required to decode the frame to original sample points very quickly. The last challenge is to find the trajectory segments from candidate trajectory set efficiently. Hence, to solve these challenges, we propose a novel method, which includes three phases, pruning, decoding and searching. We discuss our three-phases method in the following:

### *6.2.1 Sliding window based approach*

In order to prune the frame groups and process the trajectory similarity search, we first synchronize the query trajectory by using trajectory calibration techniques [38], if the query trajectory does not come from the same dataset. This is because, if the trajectories are heterogeneous, the similarity measure may suffer the accuracy problem, which is discussed in [38]. After the query trajectory is calibrated, we select frame groups that are within a given time period, and start the first phase of query processing, which is the pruning phase.

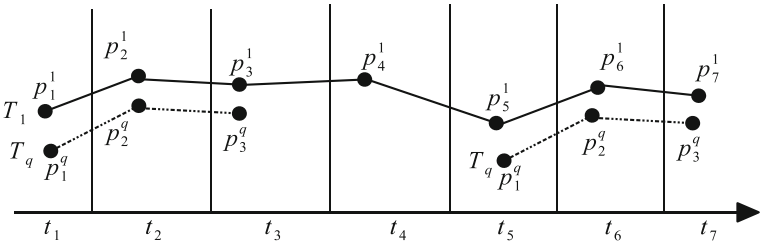



**Figure 6** Example of similar trajectory

In the pruning phase, we can use the maximum moving distance $D_{max}$ to prune the frame group. So, we only need to test the I-frame point first. If the minimum distance between I-frame point and query trajectory is larger than $D_{max} + \alpha$, it means this frame group does not contain any sample point and its distance to any point in query trajectory can be smaller than $\alpha$, then we can prune this frame group safety. After all of frame groups are scanned, the filtering processing is executed on these candidate frame groups.

In the filtering process, we sort the candidate frame groups by trajectory ID. After that, we can get the length of each continuing frame group (i.e., number of sample points) and compare with length of query trajectory. If the length of a continuing frame group is less than the length of query trajectory, these frame groups can be filtered out, since their previous frame group and next frame group have been pruned. For example, assuming a trajectory contains 5 frame groups from $FG_1$ to $FG_5$ and $n$ is set to 8. After the pruning process, the frame group $FG_1$ and $FG_5$ are pruned, the rest of frame groups $FG_2$, $FG_3$ and $FG_4$ are put into candidate set. In this filtering process, we calculate the total length of $FG_2$, $FG_3$ and $FG_4$ (i.e., the sample points that these frame groups contains). If the total length of these frame groups is less than length of query trajectory, we can filter out the frame group $FG_2$, $FG_3$, $FG_4$ directly as they cannot contain the result trajectories.

In the decoding step, we can decode the candidate frame groups directly to re-construct the candidate trajectories. Then, we move to the searching phase. In order to find the exact similar trajectory segments, we use a sliding window to check each candidate trajectory segment. The size of the sliding window equals to the length of query trajectory. During query processing, for each candidate trajectory, we move this window among this trajectory, and calculate the actual distance of each point pair to check whether there is a segment of candidate trajectory that meets the requirement. For example, we first test the trajectory segment, which is contained in the window, and then we slide the window into the next point to repeat the computation task again. We keep the sliding window moving until it reaches the end of the trajectory. If there is more than one possible result such as the trajectory segment from $p_4^1$ to $p_6^1$, and another trajectory segment from $p_5^1$ to $p_7^1$ both meet the requirement with query trajectory as Figure 7 shows. In this case, we calculate the average distance from query trajectory to each trajectory segment and select the one with minimum average distance. In this example, trajectory segment from $p_4^1$ to $p_6^1$ is returned as result. Otherwise, if two trajectory segments are not connected, then both segments are returned as the final results.

However, this algorithm still has some drawbacks. First of all, the boundary value $D_{max}$ could be very large if $n$ is large, which can cause the low performance of pruning phase, while the searching space is now very large. The second is that the decoding phase still needs more optimization to reduce the processing time. In the last, using sliding window is time-consuming. This is because, in most cases, the length of query trajectory is much shorter than candidate trajectory, which means the time complexity of search phase is $O(ab)$, where $a$ is the length of candidate trajectory and $b$ is the length of query trajectory. Therefore, this algorithm needs further optimization to improve the performance.

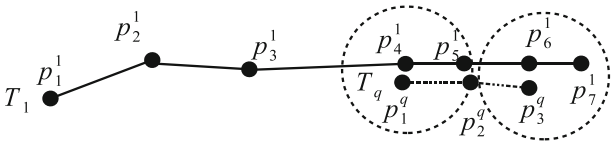



**Figure 7** Example of connected trajectory

#### 6.2.2 MBR based approach

As we discussed before, each frame group has been embedded into the MBR. Therefore, in the pruning phase, we calculate the MBR information of query trajectory firstly. And then we use the minimum distance between MBR of query trajectory and MBR of each frame group as the new boundary value, which is also combining with $\alpha$, to prune the frame groups. Using MBR information can improve the performance significantly as it will reduce the search space a lot. Meanwhile, we can also invoke parallel processing on filter phase, since each frame group is independent that it is the one of advantage of the frame structure. Following by this, each frame group can be assigned to a different thread to check the bound value. In addition, the filter step still remain the same, since it can be done very quickly.

In the decoding step, we can also invoke parallel processing to further accelerate the speed of decoding. Based on this, each frame group will be assigned into different threads for decoding. To continue improving the performance of searching phase, we utilize the benefits of the frame structure. As all of the trajectories are synchronized, we can transform trajectories as a text, and each frame point (i.e., trajectory sample point) in that trajectory is considered as a word. Therefore, if the distance between two sample points is less than $\alpha$, these two points can be looked as the same word. Moreover, it allows us to use the text similarity measures to do the similarity search on trajectory. The text similarity searching has been extensively studied, we decide to use KMP algorithm [25]. There are two advantages that we use KMP to do similarity search, the first is the KMP algorithm is easy to extend to support the distance calculation, the second is the time complexity of KMP algorithm is linear $O(a + b)$, where $a$ is the length of candidate trajectory and $b$ is the length of query trajectory. Hence, it is better than sliding window computation.

## 7 Experiments

In this section, we conduct extensive experiments on real trajectory datasets and a synthetic trajectory dataset to study the performance of the proposed data structure and query processing methods.

### 7.1 Experimental setup

We use two real world taxi trajectory datasets (A: Beijing, B: Nanjing), which are collected by a third party cooperation and cannot be open published due to a confidentiality agreement. Similarly, for testing the influence of heterogeneous trajectory dataset in our system, we generate a synthetic trajectory dataset; and the sampling rate distribution will be based on normal distribution with different mean and standard deviation of sampling rate. The results are discussed in Section 6.3. The detailed statistics of the datasets are given in Table 4,

**Table 4** Trajectory dataset

| Dataset | # Sample Points | # Trajectories | Size | % Raw Sample Points |
|---|---|---|---|---|
| Dataset A | 0.7 billion | 243,194 | 14.6 GB | 93% |
| Dataset B | 80 million | 28,784 | 2.52 GB | 91% |
| Synthetic | 3 billion | 729,582 | 80GB | 76% |

where the last column indicates the percentages of raw sample points in the datasets after interpolation for default value of time interval for frame.

We compare the time cost of the proposed column-oriented data structures against the **row-oriented** data structure. The row-oriented data structure is segmenting the trajectory and storing them in-memory, which is implemented by ourselves and called segmented trajectory database. In the segmented trajectory database, a trajectory is split into small segments, and each segment contains a fixed number (or no more than a fixed number) of sample points. Each segment is described as a MBR, which also includes its start time and end time to speed up the query processing. For number of sample points in each segment, we experimented with different values, and finally chose 50, which achieves the best performance in our experiments. Moreover, we also have tried to use R-tree. However, it costs much more additional main memory compared with our method, thus we do not discuss it in experiments.

Table 5 shows the default value and the range of each parameters. Due to the space limitation, we only examine the kNN query with $k = 5$, since we are more interested in how length of time interval of query can affect the performance. The $T_s$ is the sampling rate of the trajectory datasets, so $2T_s$ means the time interval of each frame is two times with trajectory sampling rate. The default value of time interval for frame is $T_s$, which is to avoid the accuracy issue to make a fair competition with baseline method. For our hierarchical based approach, we set the building stop condition as equal to the time duration of each frame group when it reaches average length of trajectory. Meanwhile, we use 10 cores for parallel model testing. For each set of experiment, we generate 100 queries and calculate the average running time. Each query is generated randomly with selected value of parameters. All the algorithms including segmented trajectory database are implemented in Java and run on a sever with two Intel 8-cores CPUs and 192GB memory.

## 7.2 Performance evaluation

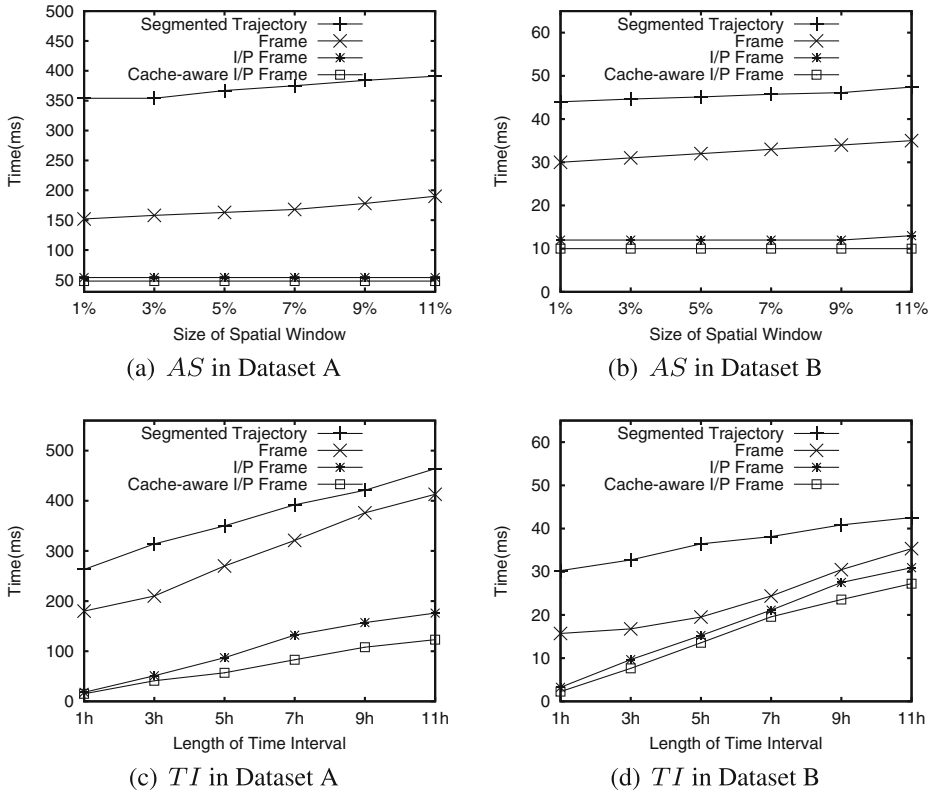
### *7.2.1 Frame storage evaluation*

In this section, we compare the different performance between segmented trajectory database, frame storage, I/P frame storage and cache-aware I/P frame storage on real world datasets. To minimize the effect from query processing approach, for I/P frame storage and cache-aware I/P frame storage, we only use the simple frame based approach; for frame storage, we apply sequential searching on each frame column.

**Performance of window query** We first evaluate the efficiency of these four approaches with different window radius with default time interval. The results are shown in Figure 8, we can see that the performance of column-oriented based data structure are very stable. This is because the main cost of query processing in column-oriented based data structure

**Table 5** Parameters setting

| Parameter | Default value | Range |
|---|---|---|
| # frames per frame group $n$ | 16 | 8−32 |
| Time interval per frame | $T_s$ | $1/4T_s$-$3T_s$ |
| Area of spatial window ($AS$) | 3% | 1%-11% |
| Length of time interval ($TI$) | 3h | 1h − 11h |



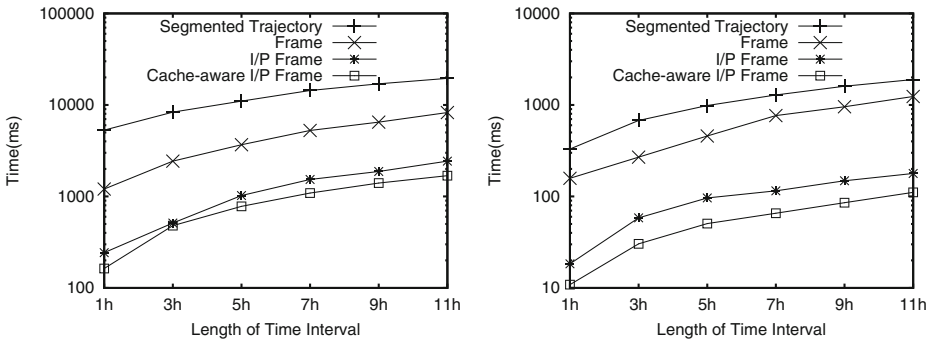

(a) $AS$ in Dataset A (b) $AS$ in Dataset B

(c) $TI$ in Dataset A (d) $TI$ in Dataset B

**Figure 8** Performance of window query in frame storage evaluation

is searching on the I-frame columns; and the number of I-frame columns that need to be searched are fixed since the time interval of queries are the same in this experiment. Hence, the performance of I/P frame storage is better than frame storage, since less I-frame columns need to be searched. Moreover, the cache-aware I/P frame storage optimizes the decoding performance, so it gets the best performance. Then, we conduct another experiment by changing the size of time window of each query, the results are also shown in Figure 8. Based on results, we can see that the length of time interval is the main factor that can affect query performance, especially for column-oriented based data structure, since the length of time intervals decide how many frames need to be searched in I/P frame based storage.

**Performance of kNN query** We investigate the query performance with regard to the length of time interval for each query. The results are shown in Figure 9. Similar to window query, the length of time interval of each query is also the main factor to affect query performance. Moreover, different to window query, the kNN query needs to investigate more frame points to get the final answers. Therefore, for I/P frame based storage, the cost of decoding P-frame points can also be much larger than window query. At the same time, the cache-aware I/P frame storage can reduce the decoding cost as it is optimized for CPU cache to increase decoding performance. Hence, the cache-aware I/P frame storage is much faster than I/P frame storage, when compared with window query. Finally, we can see that the



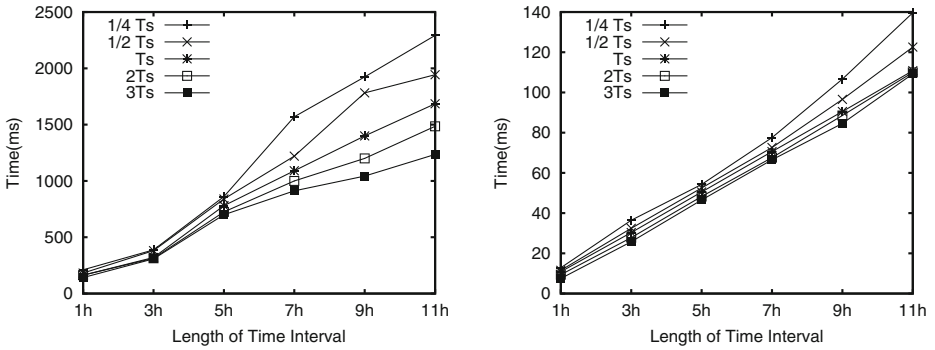

(a) Dataset A (b) Dataset B

**Figure 9** Performance of kNN query

I/P frame storage can improve the performance to at least 10 times faster than row-oriented database.

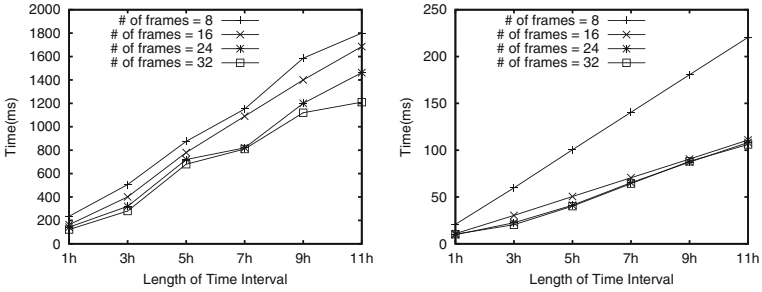
**Effect of frame rate** Next we study the query performance when the frame rate is varying. We use the running time of kNN query on the dataset A as the measurement. As the frame rate will cause the same effects in the three frame storages, we only examine the cache aware I/P frame storage. The results are present in Figure 10. Without considering the accuracy, the experiment with parameter setting $3T_s$ has the best performance since the size of whole dataset has been reduced significantly. In addition, the increasing frame rate will make the distance between two sample point longer and increasing search area of each frame group. Therefore, it causes more candidate frame groups to be pushed into candidate list and reduce the performance in the refining phase.

**Effect of number of frames for each frame group** Finally, we evaluate the query performance with different $n$. We use the same measurement method as the effect of frame rate. The running time is reported in Figure 11. As frame storage do not have frame group structure; and both I/P frame storage and cache-aware I/P frame storage use the same frame group structure, we also only examine the cache-aware I/P frame storage. We can find the




(a) Dataset A (b) Dataset B

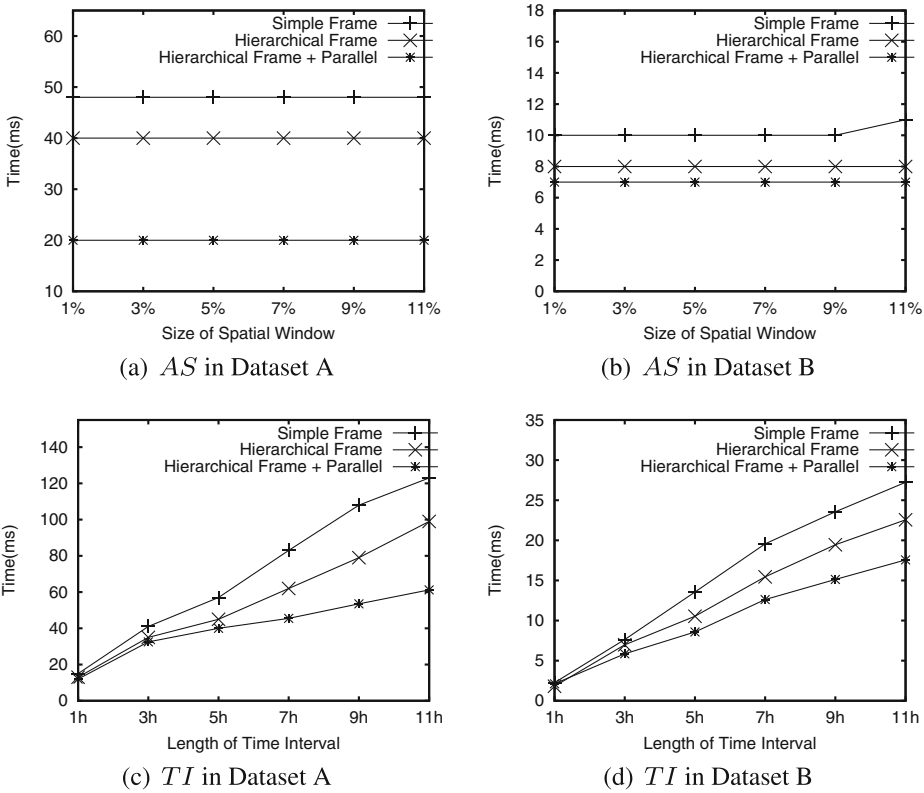
**Figure 10** Effect of frame rate




(a) Dataset A (b) Dataset B

**Figure 11** Effect of number of frames

performance is increasing when the number of $n$ is increasing. This is because, in the query processing on P-frames, the system can access the P-frame points directly without need to search the P-frame columns and send to CPU to decode, which can reduce the search time and increase the performance. Meanwhile, the system can get more benefits for the compression technical, which can reduce the consumption of memory bandwidth.




(a) $AS$ in Dataset A (b) $AS$ in Dataset B

(c) $TI$ in Dataset A (d) $TI$ in Dataset B

**Figure 12** Performance of window query in query processing evaluation

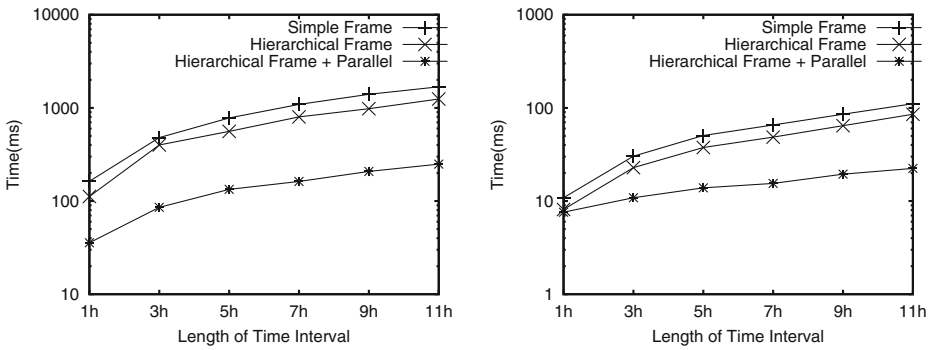
### *7.2.2 Query processing evaluation*

In the next set of experiments, we look at the different performance of our query processing approaches, which are simple frame based approach, hierarchical frame based approach and hierarchical frame based approach with parallel computing. We will test the performance of window query and kNN query.

**Performance of window query** Similar to previous experiments, the results are illustrated in Figure 12. Without surprise, the hierarchical frame based approach is better than simple frame based approach, which shows using the hierarchical structure and MBR can reduce the search space significantly. On the other hand, the parallel computing technology can improve the performance when the time interval is increasing. This is because, the overhead cost of parallel computing is more than single thread, since it has to create a thread pool to manage the threads at beginning, which leads to lower increase if the investigated I-frame columns are small (i.e., in the short length of time interval). But this overhead cost is constant, so the parallel computing approach can achieve large increasing of performance when the search space is large.

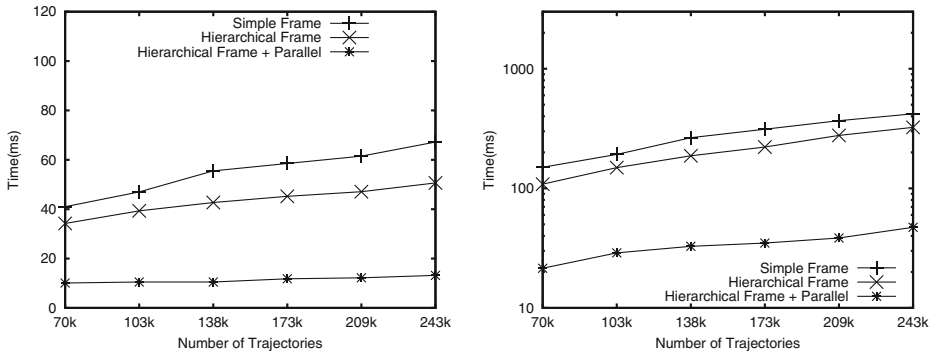
**Performance of kNN query** We now evaluate the performance of kNN query with changing the length of time interval, which is shown in Figure 13. We can see the hierarchical frame based approach can improve the performance around 30%. As the kNN query needs to investigate more frames, the parallel computing approach boosts the performance close to the theoretical performance improvement, which is near 10 time faster compared with single thread.

**Scalability evaluation** We also evaluate the scalability of all the approaches under two queries/operations. To achieve this, we random select trajectories in the dataset A with different number from 70k to (approx.) 243k. Therefore, when we increase the number of trajectories, the density of trajectory in certain area (i.e., the area is covered by whole trajectory dataset) also be increased to the higher value. The running time is reported in Figure 14. It is illustrated that the time cost of all three storages increase linearly/sub-linearly with respect to the size of dataset. Hierarchical frame based approach with parallel computing achieves the best performance in this experiment.




(a) Dataset A (b) Dataset B

**Figure 13** Performance of kNN query




(a) Window Query (b) $k$NN Query

**Figure 14** Scalability

**Parallel computing vs. single thread** We evaluate the performance of parallel computing compared with single thread, and also enlarge the number of cores to see the corresponding performance. In Table 6 we can see, when we enlarge the number of cores used for computing from 4 to 12, the boost rate increases from 2.5X to 8.39X.

### *7.2.3 Accuracy and compression ratio evaluation*

The accuracy testing is designed for frame storages particularly, as we mentioned above, time interval of each frame (i.e., frame rate) will affect the accuracy in frame encoding methods if time interval of each frame is larger than sampling frequency. At the same time, the compression ratio is calculated by the *size of encoded data in the memory* divided by the *size of original data in the memory*

**Effect of frame rate** We evaluate the accuracy by changing the time length of each frame. To measure the accuracy, we first randomly select 200 points; and find their kNN trajectories as ground truth, where $k = 100$. Then we use these points as the query points to do same query processing on the our approach. After we get the results, we use recall to measure the accuracy. For example, for a query point, its kNN ($k = 5$) ground truth is 1, 2, 3, 4, 5, and result of our approaches is 1, 2, 3, 4, 6, and its recall is 80%. The results are shown in the Table 7 with the compression ratio. As expected, increasing time length of frame will cause a low accuracy rate, but the compression performance is increased in this case. Moreover, when the time length of each frame is less than sampling rate of trajectory, there is no improvement in accuracy, but the need for memory is increased.

**Table 6** Parallel computing vs. single thread

| Cores | Parallel | Single thread | Boost rate |
|---|---|---|---|
| 4 | 534 ms | 1342 ms | 2.5X |
| 6 | 335 ms | 1342 ms | 4.0X |
| 8 | 192 ms | 1342 ms | 7.0X |
| 12 | 160 ms | 1342 ms | 8.39X |

**Table 7** The results of accuracy

| | Dataset A | | Dataset B | |
|---|---|---|---|---|
| Time length | Recall | Ratio | Recall | Ratio |
| $1/4\ T_s$ | 100% | 69.3% | 100% | 69.3% |
| $1/2\ T_s$ | 100% | 36.3% | 100% | 36.3% |
| $T_s$ | 100% | 18.3% | 100% | 18.3% |
| $2T_s$ | 82% | 11.2% | 86% | 11.2% |
| $3T_s$ | 76% | 7.4% | 77% | 7.4% |

**Effect of number of frames for each frame group** Then we investigate the compression ratio with regard to the number of frames in a frame group. The results are shown in Table 8 for both datasets. We observe that for different value of $n$, definitely, the higher $n$ is set, the higher compression ratio we can achieve. This is because increasing $n$ will reduce the total number of I-frame columns in the whole storage system; and P-frame point costs much less than I-frame point.

### 7.3 Synthetic trajectory data evaluation

The range of mean and standard deviation of sampling rates is shown in Table 9. Meanwhile the range of sampling rate is from 30 seconds to 240 seconds during generation. Moreover, in this experiment, we set the frame rate equal to the mean sampling rate, and use the same settings ($k$NN query) as previous experiment for accuracy and performance testing.

**Effect of standard deviation** Figure 15a illustrates the effect of standard deviation of sampling rate. As we can see, the heterogeneous trajectory can cause an impact on accuracy of query processing, especially when the standard derivation is high. This is because, if sampling rate of a part of the trajectory is higher than mean sampling rate, some sample points will be removed during frame encoding. Therefore, the higher standard deviation, the more sample points will be removed. This case will decrease the accuracy rate, which is similar as the evaluation of effect of frame rate. However, we can see that our system has a good tolerance for low standard deviation. On the other hand, there is no effect on performance for different standard deviation since it does not affect the frame rate.

**Effect of mean sampling rates** Figure 15b shows the results of different mean sampling rates. We can find that performance is increasing with lower mean sampling rate, since less frame group columns will be scanned during query processing. This is because, the larger mean sampling rate has longer frame rate, which reduces the number of frame group column in a certain time period. In the same time, based on our evaluation settings, the smaller mean sampling rate can increase the accuracy. The reason is the smaller frame time interval in each frame, which means the number of sample points that need to be removed is lower than the larger frame time interval for high sampling trajectories.

**Table 8** The results of compression ratio

| | $n = 8$ | $n = 16$ | $n = 24$ | $n = 32$ |
|---|---|---|---|---|
| Dataset A | 21.1% | 18.3% | 16.1% | 15.0% |
| Dataset B | 21.1% | 18.3% | 16.1% | 15.0% |

**Table 9** Synthetic evaluation parameters setting

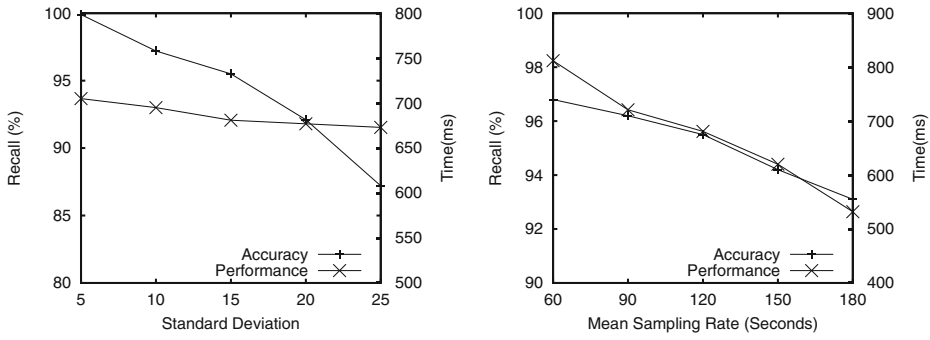
| Parameter | Default value | Range |
|---|---|---|
| Mean sampling rate (seconds) | 120 | 60−180 |
| Standard deviation | 15 | 5–25 |

## 7.4 Evaluation of trajectory similarity search

In this section, we implement four algorithms with different processing strategies. We divided our algorithm into two main parts. The first part includes pruning phase and decoding phase and the second part includes calculation phase. The details of implemented algorithms are listed in the following Table 10, where $S$ is denoted as single thread based algorithm, $P$ is denoted as multi-thread based algorithm and $RC$ is denoted as reconstructing algorithm (i.e., decoding algorithm). Moreover, we still use the same real world datasets for this experiment, and the additional parameters settings are listed in the Table 11.

**Effect of query time interval** The query time interval is a main factor that can affect the performance of query processing. The results of these algorithms are shown in Figure 16 for both datasets. The running time of naive algorithm is increasing very fast while the duration of query time is increasing, since the longer query time will require more frame groups (i.e., more sample points) that need to be tested. The performance of single thread MBR-KMP algorithm is better than the naive algorithm due to better pruning algorithm and distance calculating algorithm, however, the trend of performance is similar with naive algorithm. For multi-thread MBR-KMP algorithm, we can see its performance is same as the single thread MBR-KMP algorithm. This is because the overhead cost of multi-thread is more than single-thread, such as thread pool management and thread allocation management. But, when the duration of query time is increasing, the power of parallel processing starts to release, as we can see, the performance of multi-thread MBR-KMP algorithm is stable. Therefore, the performance of multi-thread MBR-KMP dominates other two algorithms.

**Effect of $\alpha$** The next experiment studies the different value of $\alpha$, since it is another factor to affect the performance of queries. The results of these algorithms are reported in Figure 17. Comparing multi-thread MBR-DP algorithm and multi-thread MBR-KMP algorithm, the multi-thread MBR-DP algorithm is faster 20% than multi-thread MBR-DP algorithm, which




(a) Standard Deviation (b) Mean

**Figure 15** Synthetic evaluation

**Table 10** Implemented algorithms

| Algorithm | The first part | The second part |
|---|---|---|
| Naive algorithm | $S\text{-}D_{max} + S\text{-}RC$ | Sliding window |
| S-MBR-KMP | $S$-MBR $+S\text{-}RC$ | KMP |
| P-MBR-SW | $P$-MBR $+P\text{-}RC$ | Sliding window |
| P-MBR-KMP | $P$-MBR $+P\text{-}RC$ | KMP |

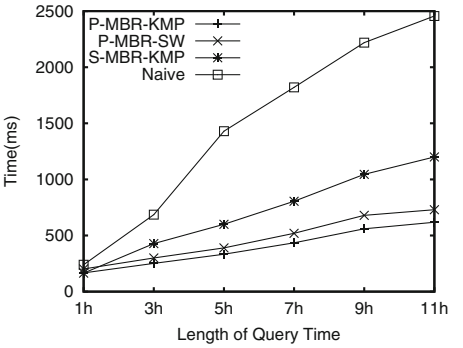
means KMP algorithm can reduce the computational cost compare with the sliding window algorithm. However, as we can see, the running time of multi-thread MBR-KMP algorithm is 5 times faster than naive method, which shows the advantage of our pruning method on frame structure as it can prune the unnecessary frame groups effectively, and reduce the pressure of last calculating step significantly. The performance of the naive method decreases very quickly, since larger $\alpha$ will lead the pruning method of naive method lose efficacy and increase the calculation workload. Moreover, the performance of multi-thread MBR-KMP algorithm is stable, and shows the parallel processing can keep the performance in these tasks very well.

**Effect of query length** The impact of varying query length is shown in Figure 18. There is no doubt that the longer query trajectory needs more running time to get the results. This is because, the longer query trajectory may insect with more trajectories in the trajectory dataset; and this case will increase the number of trajectories to be pushed into the calculation phase. First of all, we notice that the performance of multi-thread MBR-KMP algorithm is two times faster than single-thread MBR-KMP algorithm, which provides the evidence that the parallel computing can obtain many benefits from our novel frame structure, even if our algorithm is partial paralleled (i.e., the pruning phase and decoding phase is paralleled, but the calculation phase is not). Moreover, it also shows the parallel processing can exactly improve the performance for our pruning and decoding algorithms. In addition, the running time of sliding window based algorithm is increasing faster than KMP based algorithm as longer query length will take more calculations in calculation phase.

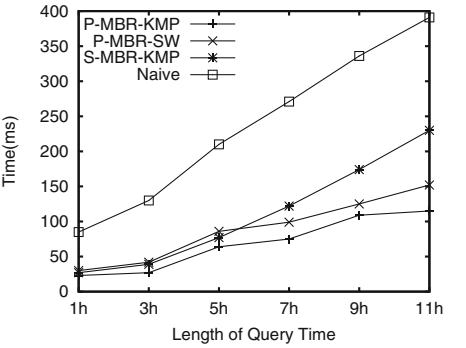
**Effect of $n$** In this test, we investigate the performance with different $n$. The results are shown in Figure 19. When $n$ ($n \leqslant 24$) is increasing, the performance of these algorithms are also increasing. The reason is that the larger $n$ is set in I/P frame structure, the less number of frame groups will be checked in pruning phase. Moreover, a frame group with more P-frame points will also have better performance than a frame group with less P-frame points, since decoding P-frame points is much faster than seeking frame groups (i.e., I-frame points) in the memory. It is obvious that the number of frame groups can be reduced by setting a large $n$, which means the performance can increase a lot with a large $n$. Therefore, the number of $n$ can affect the performance of similarity search more significantly than window query and $k$NN query, since similarity search requires decoding more frame groups. In addition, the large $n$ will also increase the size of MBR or $D_{max}$ of each frame group and then reduce

**Table 11** Parameters setting on similarity tests

| Parameter | Default value | Range |
|---|---|---|
| # simple points of query | 150 | 30−330 |
| length of $\alpha$ (meters) | 100 | 50−300 |

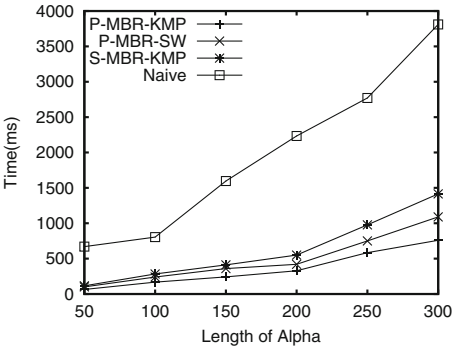



(a) Dataset A

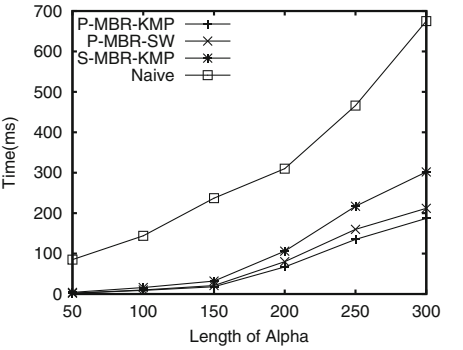



(b) Dataset B

**Figure 16** Effect of query time interval

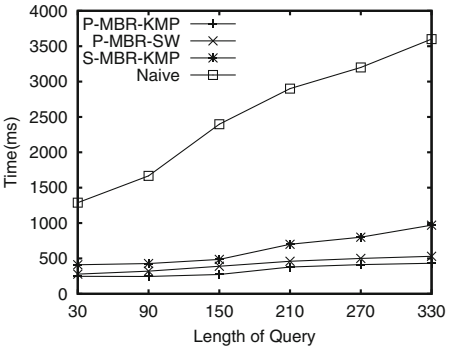



(a) Dataset A

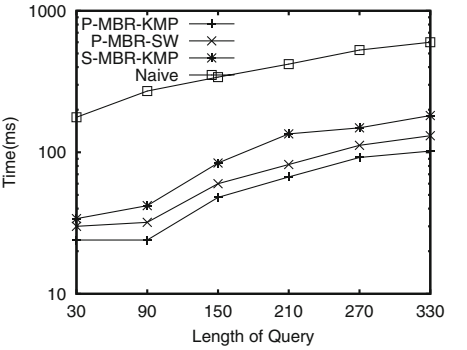



(b) Dataset B

**Figure 17** Effect of $\alpha$

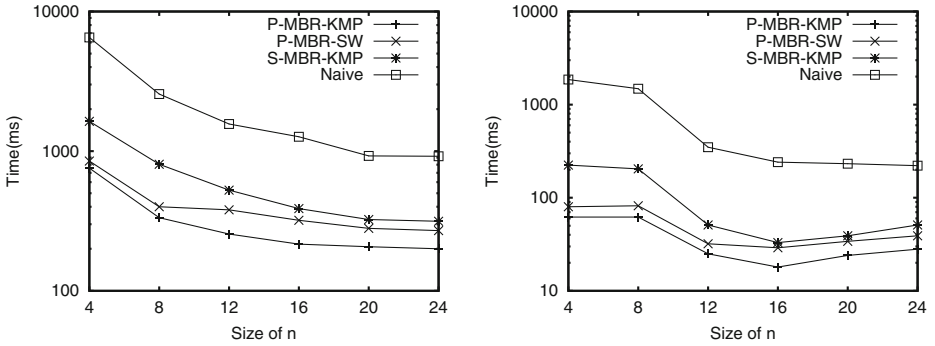



(a) Dataset A




(b) Dataset B

**Figure 18** Effect of query length




(a) Dataset A (b) Dataset B

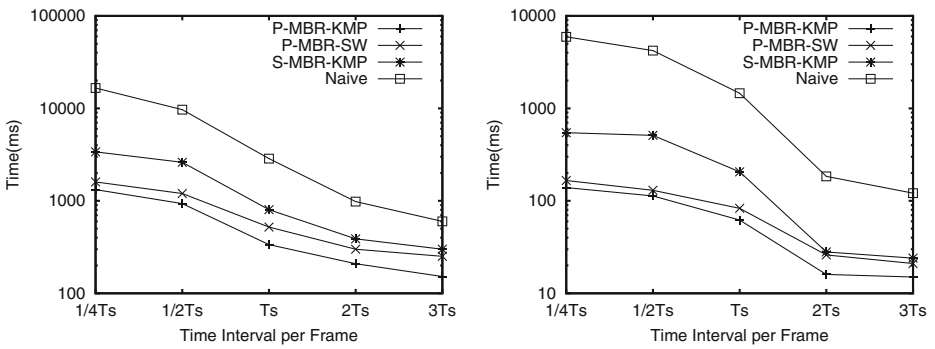
**Figure 19** Effect of $n$

the efficiency of pruning process. As we can see, when $n$ is larger than 24, the increased performance of decoding has been counteracted by inefficient pruning processing.

**Effect of $T_S$** The results of similarity search are similar to the results of window query and $k$NN query, which is shown in Figure 20. The lowest frame rate (i.e., $3T_s$) has the best performance. Moreover, the performance of multi-thread MBR-KMP algorithm is much better than naive algorithm, when the frame rate is set to highest (i.e., $1/4\ T_s$). As the highest frame rate has the largest data size in this experiment, it shows the parallel computing is a good idea for large data processing.

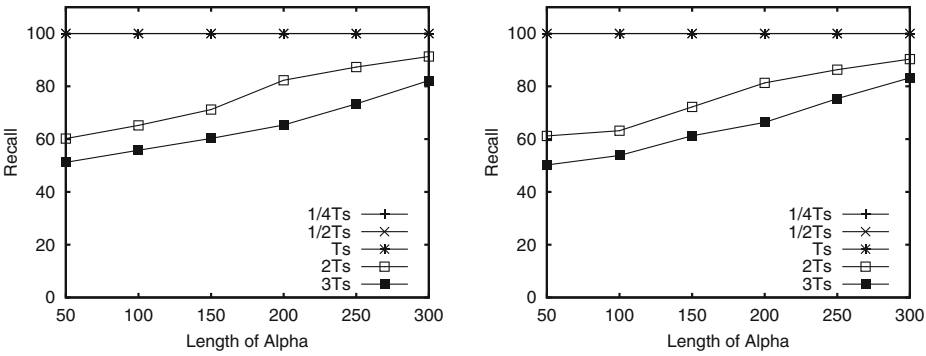
### 7.5 Accuracy evaluation of trajectory similarity search

Based on previous experiment, we know that changing the frame rate in I/P frame structure can affect the accuracy of results significantly. In this experiment, we set five different frame rates, which are same as previous experiment, with different $\alpha$ to test the accuracy of similarity search. We set the $\alpha$ is from $50m$ to $300m$. The accuracy is also measured by recall, and the results are shown in Figure 21. These is no doubt that the accuracy will be reduced, when we set the frame rate is same as the original trajectory sampling rate. As




(a) Dataset A (b) Dataset B

**Figure 20** Effect of $TS$




(a) Dataset A (b) Dataset B

**Figure 21** Accuracy

expected, if the frame rate is larger than original trajectory sampling rate (e.g., $1/4\ T_s$ and $1/2\ T_s$) and the $\alpha$ is small, the accuracy of results will not be also affected. This is because, the interpolated sample points do not change the shape of the trajectory, and no information will be lost due to the interpolation. Similar to window query and $k$NN, the accuracy can be affected significantly, when the frame rate is lower than original trajectory sampling rate as the shape of trajectory has been changed due to many sample points of trajectories have been removed. Finally, as we can see, the accuracy is very low with small value $\alpha$. Therefore, it has a trade-off between data size and accuracy. If the application is more sensitive about accurate results, it is better to choose higher frame rates.

## 8 Conclusion

In this paper, we presented SharkDB, a new in-memory column-oriented storage system for storing and querying trajectory data. We developed an I/P frame based column-oriented data structure with CPU-cache optimization to provide an efficient storage system. To support efficient query processing, we proposed several algorithms that using both hierarchical structure and parallel computing, which fully utilize the of I/P frame data structure. Extensive experimental results based on both real and synthetic datasets demonstrate that the proposed method outperforms several baseline algorithms significantly with good scalability.

**Acknowledgements** This work is partially supported by Natural Science Foundation of China (No. 61502324 and No. 61532018).

## References

1. Ammann, A.C., Hanrahan, M., Krishnamurthy, R.: Design of a memory resident DBMS. In: COMPCON, pp. 54–58 (1985)
2. Aßfalg, J., Kriegel, H.P., Kröger, P., Kunath, P., Pryakhin, A., Renz, M.: Similarity search on time series based on threshold queries. In: International Conference on Extending Database Technology, pp. 276–294. Springer (2006)
3. Baulier, J., Bohannon, P., Gogate, S., Gupta, C., Haldar, S.: DataBlitz storage manager: main-memory database performance for critical applications. In: SIGMOD, pp. 519–520 (1999)

4. Bernad, D.: Finding patterns in time series: a dynamic programming approach. Advances in knowledge discovery and data mining (1996)
5. Berndt, D.J., Clifford, J.: Using dynamic time warping to find patterns in time series. In: KDD Workshop, vol. 10, pp. 359–370. Seattle, WA (1994)
6. Binnig, C., Hildenbrand, S., Färber, F.: Dictionary-based order-preserving string compression for main memory column stores. In: SIGMOD, pp. 283–296 (2009)
7. Bitton, D., Hanrahan, M., Turbyfill, C.: Performance of complex queries in main memory database systems. In: ICDE, pp. 72–81 (1987)
8. Boncz, P.A., Zukowski, M., Nes, N.: Monetdb/X100: hyper-pipelining query execution. In: CIDR, pp. 225–237 (2005)
9. Botea, V., Mallett, D., Nascimento, M.A., Sander, J.: PIST: an efficient and practical indexing technique for historical spatio-temporal point data. GeoInformatica **12**(2), 143–168 (2008)
10. Chakka, V.P., Everspaugh, A.C., Patel, J.M.: Indexing large trajectory data sets with SETI. In: CIDR (2003)
11. Chen, L., Ng, R.: On the marriage of Lp-Norms and edit distance. In: Proceedings of the 13th International Conference on Very Large Data Bases-Volume 30, pp. 792–803. VLDB Endowment (2004)
12. Chen, L., Özsu, M.T., Oria, V.: Robust and fast similarity search for moving object trajectories. In: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data, pp. 491–502. ACM (2005)
13. Cudre-Mauroux, P., Wu, E., Madden, S.: Trajstore: an adaptive storage system for very large trajectory data sets. In: ICDE, pp. 109–120 (2010)
14. Faloutsos, C., Ranganathan, M., Manolopoulos, Y.: Fast subsequence matching in time-series databases, vol. 23. ACM (1994)
15. Forlizzi, L., Güting, R.H., Nardelli, E., Schneider, M.: A data model and data structures for moving objects databases, vol. 29. ACM (2000)
16. Frentzos, E., Gratsias, K., Pelekis, N., Theodoridis, Y.: Algorithms for nearest neighbor search on moving object trajectories. Geoinformatica **11**(2), 159–193 (2007)
17. Frentzos, E., Gratsias, K., Theodoridis, Y.: Index-based most similar trajectory search. In: IEEE 23Rd International Conference On Data Engineering, 2007. ICDE 2007, pp. 816–825. IEEE (2007)
18. Gawlick, D., Kinkade, D.: Varieties of concurrency control in IMS/VS fast path. DEB **8**(2), 3–10 (1985)
19. Gowanlock, M., Casanova, H.: In-memory distance threshold queries on moving object trajectories. In: Proceedings of the 6th International Conference on Advances in Databases, Knowledge, and Data Applications, pp. 41–50 (2014)
20. Guttman, A.: R-trees: a dynamic index structure for spatial searching. In: SIGMOD, pp. 47–57 (1984)
21. Hadjieleftheriou, M., Kollios, G., Tsotras, V., Gunopulos, D.: Efficient indexing of spatiotemporal objects. Advances in Database Technology—EDBT **2002**, 251–268 (2002)
22. Héman, S., Zukowski, M., Nes, N.J., Sidirourgos, L., Boncz, P.: Positional upyear handling in column stores. In: SIGMOD, pp. 543–554 (2010)
23. Ivanova, M.G., Kersten, M.L., Nes, N.J., Gonçalves, R.A.: An architecture for recycling intermediates in a column-store. TODS **35**(4), 24:1–24:43 (2010)
24. Keogh, E.: Exact indexing of dynamic time warping. In: Proceedings of the 28th International Conference on Very Large Data Bases, pp. 406–417. VLDB Endowment (2002)
25. Knuth, D.E., Morris, J.H., Jr. Pratt, V.R.: Fast pattern matching in strings. SIAM J. Comput **6**(2), 323–350 (1977)
26. Krueger, J., Kim, C., Grund, M., Satish, N., Schwalb, D., Chhugani, J., Plattner, H., Dubey, P., Zeier, A.: Fast upyears on read-optimized databases using multi-core CPUs. PVLDB **5**(1), 61–72 (2011)
27. Lemke, C., Sattler, K.U., Faerber, F., Zeier, A.: Speeding up queries in column stores. In: Dawak, pp. 117–129 (2010)
28. Manegold, S., Boncz, P., Kersten, M.L.: Generic database cost models for hierarchical memory systems. In: PVLDB, pp. 191–202 (2002)
29. Meratnia, N., By, R.: Spatiotemporal compression techniques for moving point objects. In: EDBT, pp. 765–782 (2004)
30. Pfoser, D., Jensen, C.S., Theodoridis, Y., et al.: Novel approaches to the indexing of moving object trajectories. In: Proceedings of VLDB, pp. 395–406 (2000)
31. Plattner, H.: A common database approach for OLTP and OLAP using an in-memory column database. In: SIGMOD, pp. 1–2 (2009)
32. Plattner, H.: SanssouciDb: an in-memory database for processing enterprise workloads. In: BTW, vol. 20, pp. 2–21 (2011)
33. Rao, J., Ross, K.A.: Making B+- trees cache conscious in main memory. In: SIGMOD, pp. 475–486 (2000)

34. Rasetic, S., Sander, J., Elding, J., Nascimento, M.A.: A trajectory splitting model for efficient spatio-temporal indexing. In: Proceedings of VLDB, pp. 934–945 (2005)
35. Setton, E., Girod, B.: Video streaming with Sp and Si frames. In: Visual Communications and Image Processing 2005, pp. 59,606F–59,606F. International Society for Optics and Photonics (2005)
36. Simonas Saltenis, C.S.J., Leutenegger, S.T., Lopez, M.A.: Indexing the positions of continuously moving objects. In: SIGMOD, pp. 331–342 (2000)
37. Stonebraker, M., Abadi, D.J., Batkin, A., Chen, X., Cherniack, M., Ferreira, M., Lau, E., Lin, A., Madden, S., O'Neil, E., O'Neil, P., Rasin, A., Tran, N., Zdonik, S.: C-store: a column-oriented DBMS. In: VLDB, pp. 553–564 (2005)
38. Su, H., Zheng, K., Wang, H., Huang, J., Zhou, X.: Calibrating trajectory data for similarity-based analysis. In: SIGMOD, pp. 833–844 (2013)
39. Tao, Y., Papadias, D., Sun, J.: The TPR*-tree: an optimized spatio-temporal access method for predictive queries. In: PVLDB, pp. 790–801 (2003)
40. Vlachos, M., Gunopulos, D., Kollios, G.: Robust similarity measures for mobile object trajectories. In: Proceedings of the 13Th International Workshop On Database and Expert Systems Applications, 2002, pp. 721–726. IEEE (2002)
41. Vlachos, M., Kollios, G., Gunopulos, D.: Discovering similar multidimensional trajectories. In: Proceedings of the 18th International Conference on Data Engineering, 2002, pp. 673–684. IEEE (2002)
42. Wang, H., Zheng, K., Xu, J., Zheng, B., Zhou, X., Sadiq, S.: SharkDB: an in-memory column-oriented trajectory storage. In: CIKM, pp. 1409–1418 (2014)
43. Yi, B.K., Jagadish, H., Faloutsos, C.: Efficient retrieval of similar time sequences under time warping. In: Proceedings of the 14th International Conference on Data Engineering, 1998, pp. 201–208. IEEE (1998)