THE UNIVERSITY OF QUEENSLAND

AUSTRALIA

SHARKDB: AN IN-MEMORY STORAGE SYSTEM FOR LARGE SCALE TRAJECTORY DATA MANAGEMENT

Haozhou Wang

Bachelor of Information Technology (Honours)

A thesis submitted for the degree of Doctor of Philosophy at

The University of Queensland in 2016

School of Information Technology & Electrical Engineering

Abstract

The rapid development of location-based technologies including animal tracking sensors, GPS devices embedded in taxis and buses, and smart phones carried by people has quickly led to the capability of collecting spatio-temporal information about almost any kind of moving object, resulting in huge volumes of spatio-temporal data in the form of trajectories. Business intelligence now has more interest in analysing large amounts of trajectory data rather than the data on hard disk, since querying on such trajectory data can reveal useful information. Due to high access latency and low I/O operations of hard disks, the disk-based storage systems (with traditional data structures) have been challenged by modern applications (e.g. location-based services), which require real time responses when querying large scale trajectory datasets. Therefore, novel data structures and query algorithms need to be designed to meet this requirement. In this thesis, a series of concrete and challenging problems about storing, managing and analysing large scale trajectory data are studied. A complete in-memory column-oriented storage system called SharkDB is implemented to address these problems and support real time computing for trajectory queries. Below is a brief description of contributions.

First of all, a preliminary study has been conducted to identify the trajectory synchronisation problem on large scale trajectory dataset. Based on this observation, a novel data structure, which is called a frame based data structure, is proposed to synchronise trajectories based on their temporal information. Meanwhile, to improve performance of trajectory queries, the frame based data structure is optimised by implementing this data structure into main memory with compression and CPU cache-optimisation techniques.

After implementing a frame based data structure, challenges with regard to trajectory query processing are investigated. To address these challenges, the trajectory queries are divided into three categories, i.e. basic operations, advanced operations and analytic operations. For each category, a naive algorithm is proposed first. To improve the query performance, for the category of basic operations, a parallel computing technique is used to speed up the running time of the query. For the category of advanced operations, a hierarchical I/P frame structure based approach is proposed. For the category of analytic operations, a MBR+KMP algorithm is presented.

To evaluate SharkDB, a comprehensive experimental study including operation level evaluation and system level evaluation is conducted. In the operation level evaluation, query processing using the proposed algorithms are compared to a traditional trajectory data structure. The extensive experiments demonstrate that the newly designed algorithms can guarantee real-time trajectory query processing on large scale trajectory dataset. A set of workload models that reflect real world workloads is proposed in the system level evaluation. The experiments on such workload models also verify the superiority of SharkDB over traditional data structures.

Finally, as in-memory database management systems are receiving more attention today, some commercial in-memory database management systems have been released. Hence, in the collaboration with SAP, SharkDB is migrated into SAP HANA. To achieve this, the data structures of SharkDB are re-designed to suit the architecture of SAP HANA. A set of experiments are conducted to show that SharkDB can beat other popular traditional data structures in regard to trajectory query processing.

Publications during candidature

Journal papers:

- Han Su, Kai Zheng, Jiamin Huang, Haozhou Wang and Xiaofang Zhou. Calibrating Trajectory Data for Spatio-temporal Similarity Analysis. *The VLDB Journal*, 2015, pp. 93-116.

Conference papers:

- Haozhou Wang, Han Su, Kai Zheng, Shazia Sadiq, Xiaofang Zhou. An Effectiveness Study on Trajectory Similarity Measures. In *ADC 2013*, Adelaide, pp. 13-22. (Best Paper Award)
- Haozhou Wang, Kai Zheng, Han Su, Jiping Wang, Shazia Sadiq. Efficient Aggregate Farthest Neighbor Query Processing on Road Networks, In *ADC 2014*, Brisbane, pp. 13-25.
- Haozhou Wang, Kai Zheng, Hoyoung Jeung, Shane Bracher, Asadul K. Islam, Wasim Sadiq, Shazia Sadiq, Xiaofang Zhou. Storing and Processing Massive Trajectory Data on SAP HANA. In *ADC 2015*, Melbourne, pp. 66-77.
- Haozhou Wang, Kai Zheng, Jiajie Xu, Bolong Zheng, Xiaofang Zhou, Shazia Sadiq. SharkDB: An In-Memory Column-Oriented Trajectory Storage. In *CIKM 2014*, Shang Hai, pp. 1409-1418.
- Han Su, Kai Zheng, Haozhou Wang, Jiamin Huang, Xiaofang Zhou. Calibrating trajectory data for similarity-based analysis. In *SIGMOD 2013*, New York, pp. 833-844.
- Jiping Wang, Kai Zheng, Hoyoung Jeung, Haozhou Wang, Bolong Zheng, Xiaofang Zhou. Cost-Efficient Spatial Network Partitioning for Distance-Based Query Processing. In *MDM 2014*, Brisbane, pp. 13-22.
- Haozhou Wang, Kai Zheng, Xiaofang Zhou, Shazia Sadiq. SharkDB: An In-Memory Storage System for Massive Trajectory Data. In *SIGMOD 2015*, Melbourne, pp. 1099-1104. (DEMO)
- Han Su, Kai Zheng, Jiamin Huang, Tianyu Liu, Haozhou Wang, Xiaofang Zhou. A crowd-based route recommendation system-CrowdPlanner. In *ICDE 2014*, Chicago, pp. 1178-1181. (DEMO)

Publications included in this thesis

Haozhou Wang, Kai Zheng, Hoyoung Jeung, Shane Bracher, Asadul K. Islam, Wasim Sadiq, Shazia Sadiq, Xiaofang Zhou. Storing and Processing Massive Trajectory Data on SAP HANA. In *ADC 2015*, Melbourne, pp. 66-77. - incorporated as Chapter 6.

Contributor	Statement of contribution
Haozhou Wang (Candidate)	Designed experiments (75%) Wrote the paper (60%) Designed algorithms (60%) Defined the problems with motivations(60%) Proofreading of the paper (30%) Joined the discussion (30%)
Kai Zheng	Wrote the paper (40%) Designed algorithms (30%) Designed experiments (25%) Defined the problems with motivations(20%) Proofreading of the paper (30%) Joined the discussion (20%)
Hoyoung Jeung	Joined the discussion(10%)
Shane Bracher	Joined the discussion(10%)
Asadul K. Islam	Joined the discussion(10%)
Wasim Sadiq	Joined the discussion(5%)
Shazia Sadiq	Proofreading of the paper (10%) Joined the discussion (5%)
Xiaofang Zhou	Proofreading of the paper (30%) Designed algorithms (10%) Defined the problems with motivations(20%) Joined the discussion (10%)

Haozhou Wang, Kai Zheng, Jiajie Xu, Bolong Zheng, Xiaofang Zhou, Shazia Sadiq. SharkDB: An In-Memory Column-Oriented Trajectory Storage. In *CIKM 2014*, Shanghai, pp. 1409-1418. - incorporated as Chapter 3, 4 & 5.

Contributor	Statement of contribution
Haozhou Wang (Candidate)	Designed experiments (80%) Wrote the paper (60%) Designed algorithms (60%) Proofreading of the paper (30%) Joined the discussion (30%)
Kai Zheng	Wrote the paper (40%) Designed algorithms (30%) Designed experiments (20%) Proofreading of the paper (30%) Joined the discussion (20%)
Jiajie Xu	Joined the discussion (10%)
Bolong Zheng	Joined the discussion (10%)
Xiaofang Zhou	Proofreading of the paper (20%) Designed algorithms (10%) Joined the discussion (20%)
Shazia Sadiq	Proofreading of the paper (20%) Joined the discussion (10%)

Haozhou Wang, Han Su, Kai Zheng, Shazia Sadiq, Xiaofang Zhou. An Effectiveness Study on Trajectory Similarity Measures. In *ADC 2013*, Adelaide, pp. 13-22. - incorporated as Chapter 3.

Contributor	Statement of contribution
Haozhou Wang (Candidate)	Designed experiments (80%) Wrote the paper (60%) Designed algorithms (60%) Proofreading of the paper (30%)
Han Su	Designed experiments (20%)
Kai Zheng	Wrote the paper (40%) Designed algorithms (30%) Proofreading of the paper (30%)
Shazia Sadiq	Proofreading of the paper (10%)
Xiaofang Zhou	Proofreading of the paper (30%) Designed algorithms (10%)

Contributions by others to the thesis

In all of the research problems in this thesis, Professor Xiaofang Zhou, as my principal advisor, and Dr Kai Zheng, assisted in providing technical guidance for formulating the problems, refinement of ideas as well as reviewing and polishing the drafts.

Statement of parts of the thesis submitted to qualify for the award of another degree

None.

Acknowledgments:

This thesis represents a major portion the research the has been done over the past three and a half years since I joined DKE group. During this period, there are many people supporting and helping me in my research and living, and without their contribution, I can hardly complete my PhD study. It is a pleasure to convey my gratitude to them all in my humble acknowledgment.

First and foremost, I would like to show my sincerest gratitude to my principal advisor, Professor Xiaofang Zhou, who has provided me with professional support and guidance in every stages throughout my research. Without his consistent and impressive kindness, all the paper that have been published or submitted, as well as this thesis, would not have been completed or written. His vigorous academic observation enlightens me not only in this thesis but also in my future study.

I wish to express my deep gratitude to my advisor, Dr. Kai Zheng, who not only brought me into the exciting area of spatial-temporal databases, but also has been remarkably helpful in various stages of my research. His role in my life will be very profound. I gratefully acknowledge my advisor, Prof. Shazia Sadiq, for her valuable advice in each group discussion and furthermore, sparing his precious time to read my drafts and give constructive comments about them.

It is my pleasure to specially acknowledge Prof. Heng Tao Shen, A/Prof. Xue Li, Dr. Zi Huang, Dr. Sen Wang, Dr. Hongzhi Yin, Dr. Mohamed Sharaf, Dr. Wasim Sadiq, Dr. Hoyoung Jeung, Dr. Shane Bracher and Dr. Asadul Islam for their great effort and help on my research problems.

I really appreciate that I have the opportunity to study in DKE, which is a friendly and harmonious group. I will show my gratitude to Mr. Bolong Zheng, Dr. Han Su, Mr. Lei Li, Mr. Xingzhong Du, Ms. Weiqing Wang, Ms. Ruojing Zhang, Ms. Yanxia Xu, Mr. Tieke He, Ms. Jiping Wang, Dr. Jialong Han, Ms. Wen Hua, Dr. Xiaofeng zhu, Dr. Jiajun Liu, Dr. Xin Zhao, Dr. Jingkuan Song, Dr. Yang yang, Mr. Peng Wang, Mr. Jiewei Chao, Ms. Xuefei Li, Ms. Ling Chen, Ms. Hongyun Cai, Ms. Bingyu Yi, Mr. Chao Li, Mr. Wei Wang and Mr. Litao Yu. Their sincerity and kind heart impressed me to give warm care to every other person.

Last but not least, my deepest gratitude goes to my family, to my dad and mum, who brought me up and encouraged my interest in science research, and who have taught me so much, without which this could not have happened. There is so much more to say, that is hard to express in words.

Keywords

spatio-temporal database, trajectory database, in-memory database, trajectory compression, query processing, algorithm, performance

Australian and New Zealand Standard Research Classifications (ANZSRC)

ANZSRC code: 080604, Database Management, 100%

Fields of Research (FoR) Classification

FoR code: 0806, Information Systems, 100%

Contents

List of Figures

List of Tables

Chapter 1

Introduction

An increasing amount of motion history data, trajectory data, is being collected from different sources such as GPS-enabled mobile devices, surveillance cameras and social networks. Trajectory data offers an unprecedented amount of information that can be used to help us understand the behaviour of moving objects. Effective and efficient technologies to manage large scale trajectory data are in high demand as the foundation to serve a variety of application domains such as geographical information systems, location-based services, vehicle navigation and so on. Despite the demand for efficient trajectory data processing, none of the existing relational database management systems (RDBMSs) have built-in support for trajectory data and operations/queries on this data. This is mainly due to the heterogeneity of trajectories including lengths and sampling rates, which makes it difficult to fit into the relational schema with a fixed number of columns. Moreover, most traditional RDBMSs adopt disk-based storage with significant I/O overhead when processing large amounts of data, which may not allow query processing to meet the real-time requirements in many trajectory related applications such as map services, trip planning and early event detection. Thanks to the increasing availability of larger RAM at lower costs, it has become possible and affordable to store and process the entire (or at least a significant portion of the) dataset within main memory, which can be orders of magnitudes faster than the traditional disk-based database systems.

Motivated by this demand, this thesis explores potential opportunities to boost the performance of trajectory data processing by designing a novel trajectory storage system within main memory, which has been named SharkDB. In contrast to most existing trajectory indexing methods that keep consecutive samples of the same trajectory in the same disk page, the SharkDB database is partitioned into frames in which the positions of all moving objects at the same time instant are stored together and aligned in main memory. This column-oriented based storage has been found 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 misses. The independence between frames also makes them natural working units when parallelising data processing in a multi-core environment.

1.1 Background

1.1.1 Spatio-temporal Database

A collection of data is usually stored in a database in the information technology field. A database management system (DBMS) is the kind of software that is used to store and analyse collection of data. Spatial database systems are designed to support storing and analysing spatial data such as geometric based data. A spatial database management system has been defined by R.H. Guting [36] to have three key elements. The first one is that a spatial database system should be a DBMS, which means that the spatial database needs to support basic queries on the data such as SELECT, JOIN and UPDATE. The second one is that spatial data types need to be supported in both the data model and the query language. Normally, there are three fundamental abstract data types, point, line and region, that are used to model single spatial objects. A point is usually used to store a single location in space such as a bus station, a house or a landmark. A line object represents a spatial object, which is moving though the space such as a road or a river. And a two dimensional area or surface in space is represented by a region, which is the last fundamental spatial data type in a spatial database. For example, a region object can be used to represent a lake, forest or city. The last key element of a spatial database management system is such that spatial data types need to be supported by the spatial database system with efficient query processing (which includes both the indexing structure and the algorithms).

The status of spatial objects can change over time. For example, a moving object changes its location frequently or a river may change its route very occasionally due to a flood. Therefore, a spatio-temporal database system, an extension of the spatial database system, is designed for spatio-temporal data, which means it can capture both the spatial and temporal aspects of data at the same time. Not only can spatio-temporal data be stored and represented in a spatio-temporal data system but it can also be analysed (by using data model, indexes and queries). Therefore, spatio-temporal data systems are now playing major roles in many real world applications such as trip-recommendation [132, 105, 135], urban computing [128, 129], and traffic monitoring [138, 134, 136]. Based on this, there is much research interest focused on spatial/spatio-temporal queries, which include:

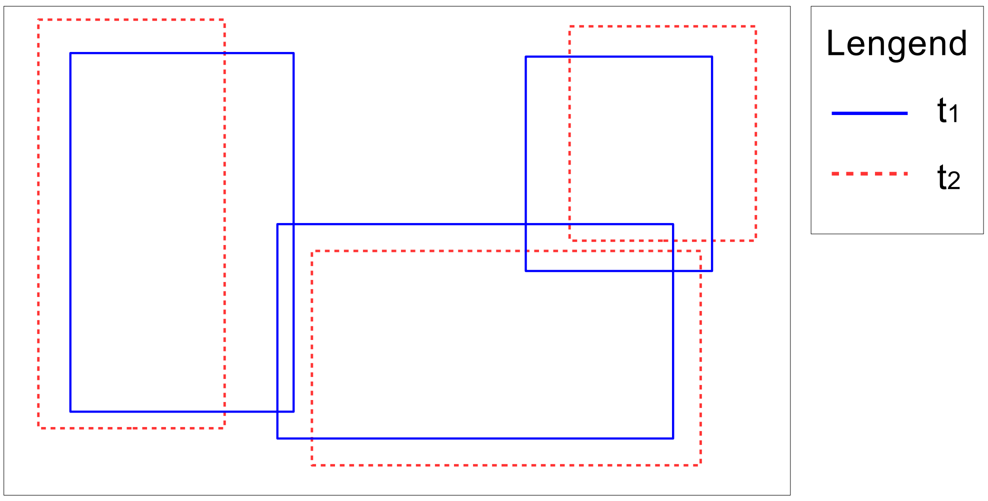
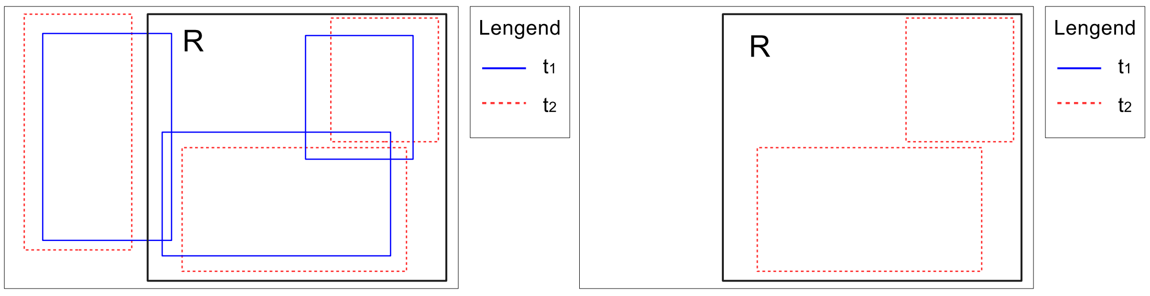


FIGURE 1.1: Example of Spatio-temporal Objects

- **Range(Window) Query** [74] which finds the objects that are contained in a given region during a given time period. For example, as Fig. 1.1 shows, rectangles with a blue solid line indicate the status of spatial objects in this area at time t_1 and rectangles with a red dashed line demonstrate the status of such objects at time t_2. Given a spatio-temporal range query with spatial window R, which is shown in Fig. 1.2(a); and a time period T, where $T = t_2$, a range query will then retrieve all spatial objects within R at time t_2, as shown in Fig. 1.2(b).

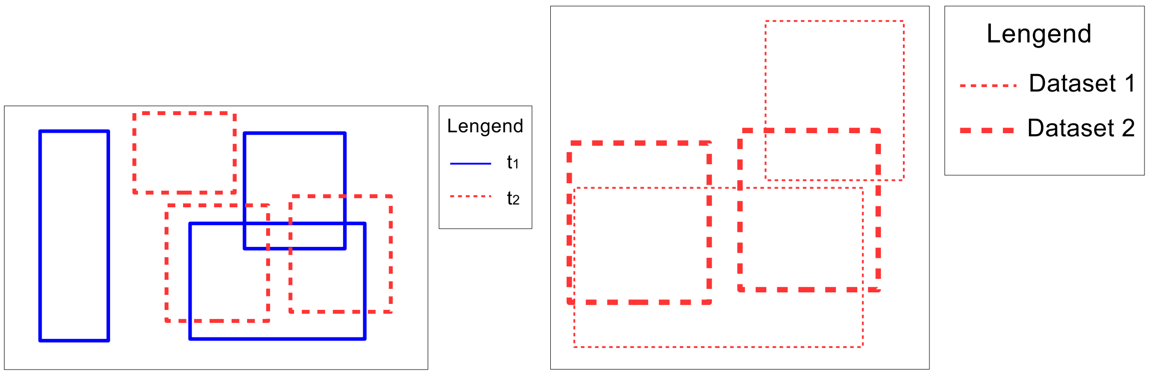
- **Spatio-temporal Join Query** [15, 65, 75] which given a predicate, spatio-temporal join query is used to find the pair of objects that satisfy this predicate in a given time period. Given a dataset (denoted as dataset 2, as shown in Fig. 1.3(a)), a spatio-temporal join query



(a) Query Example (b) Query Results

FIGURE 1.2: Example of Range Query

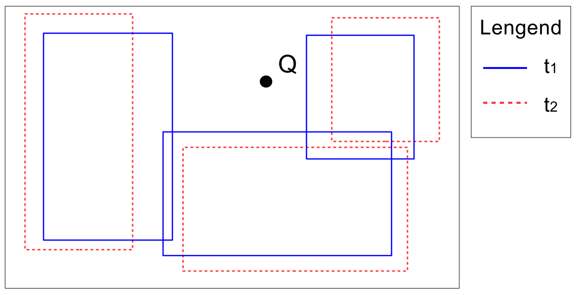
can be used to find the overlap between spatial objects of these two datasets at time t_1. The results are shown in Fig. 1.3(b) where spatial objects with a bold red dashed line belong to dataset 2 and the rest of spatial objects belong to original dataset (denoted as dataset 1).



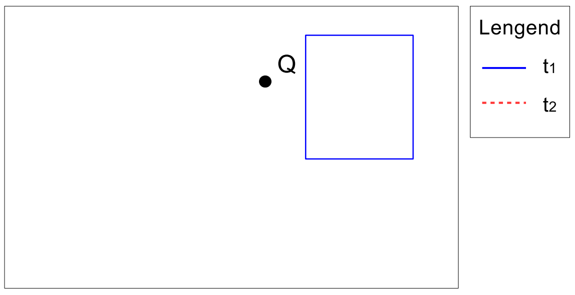
(a) Query Example (b) Query Results

FIGURE 1.3: Example of Spatio-temporal Join Query

- **Nearest Neighbour (NN) Query** [41, 7, 94] The NN query is used to find an object, which is closest to a given query in a given time period. Given a query point Q as Fig. 1.4(a) shows, a NN query returns the closest spatial object from this query point, as shown in Fig. 1.4(b) with time period $T = t_1$. In addition, a popular version of the NN query is the top-k NN query (kNN), which is used to find k closest objects.



(a) Query Example



(b) Query Results

FIGURE 1.4: Example of Nearest Neighbour Query

1.1.2 Moving Objects and Trajectories

Moving objects data (MOD) and trajectory data are both spatio-temporal data and can be stored in the spatio-temporal database. As discussed previously, a data object stored in a spatio-temporal database contains not only the spatial information, but also the related temporal information. Therefore, a spatio-temporal database can be used to store the dynamic status of a spatial object. For example, changing area of a lake can be stored as a spatio-temporal object. The dynamic status of spatial objects are very important, since people would like to know the status of a spatial object at different times. Meanwhile, such data allows us to be able to go back to any particular time to retrieve the status of spatial objects at that time.

Both moving objects data and trajectory data are used to represent the dynamic status of a moving object. Although these data types are often used as synonymously, MOD is about the current and near future positions, and trajectory data is about the complete history of moving objects as differentiated in [30]. This does introduce a few differences between these two types of data. MOD data also typically records additional information such as velocity (i.e. the current moving direction and speed), and it is important to handle the streaming nature of new position data on which some data correction algorithms such as map-matching should be applied. For trajectory data, on the other hand, the focus is on efficient processing of a large volume of moving object history data.

This thesis considers trajectory data only. The definition of a trajectory is as follows: A position $l = (x, y)$ is a spatial point in a geographical space with longitude x and latitude y. A trajectory point is a spatial-temporal point $p = (x, y, t)$ that represents location $l = (x, y)$ at time t. Let each moving object be identified by its unique id number, and T_i be the trajectory of object i. T_i is a sequence of trajectory points p_1, p_2, . . ., p_{n_i} ordered by time (i.e. $\forall j = [1..n_i), p_j.t < p_{j+1}.t$). An example of a trajectory is shown in Fig. 1.5. The blue points represent the sample points.

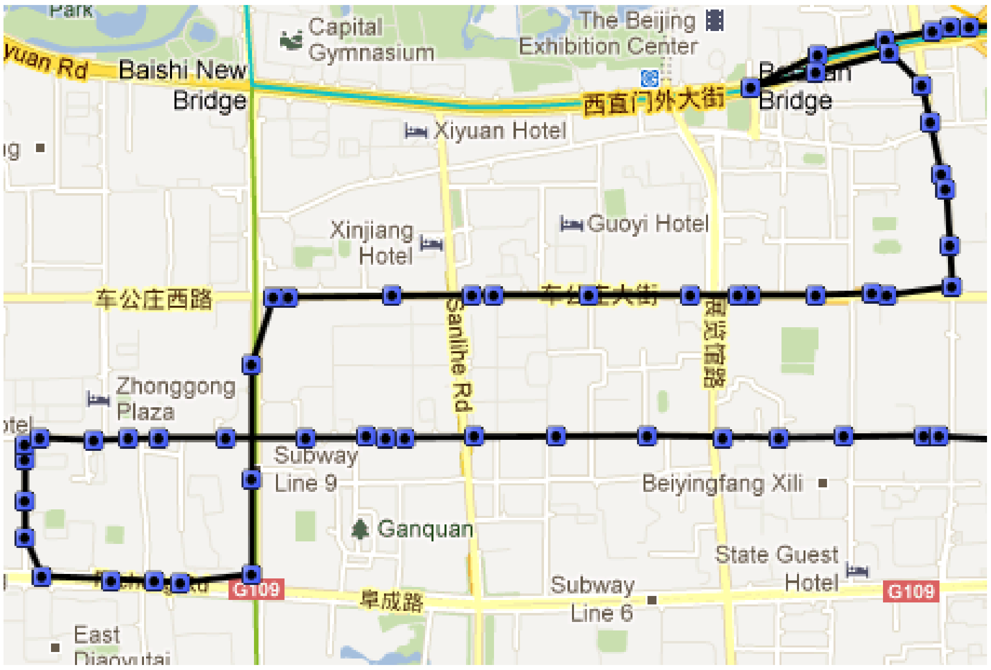


FIGURE 1.5: Trajectory Example

The trajectory data is not only used for storing and representing purpose, but also for analysing purposes. In general, there are three basic trajectory based query types: window query, k nearest neighbour (kNN) query and trajectory similarity search. More specifically, the window query is used to find all the trajectories that through a given region within a given period of time, for example, a window query can be used to find the trajectories that pass through a given region between 8:00 am and 8:15 am. A kNN query, is used to find the top-k trajectories that are close to a given point and active during a given period of time. For example, a NN query can be used to find the closest trajectory to a given spatial point that was active between 8:00 am and 8:15 am. The trajectory similarity search is used to find a set of trajectories that are similar to a given trajectory. The similarity between each pair of trajectories is calculated by a pre-defined similarity measure. For example, similarity search query can be used to find similar trajectories moving from the city to the airport via the airport highway between 8:00 am and 10:00 am. Therefore, the trajectory data can be used in many important applications in various areas.

1.1.3 In-memory Data Management

Owing to the development of cheap RAM-based storage technology, modern computing hardware can afford much larger main memory. Consequently, traditional database systems can be re-designed to store and manage all the data in main memory permanently. Such types of in-memory database systems are attracting increasing attention from both academia and industry due to their outstanding performance in processing large amounts of data. Currently, the bottleneck of the traditional disk-oriented database system is the I/O cost, since the I/O performance of hard disks is very limited. To alleviate this bottleneck, in-memory database systems move all data into main memory instead of storing the data on the hard disk, which means all data are accessed and maintained in the main memory directly, since the main memory can provide high speed, random access. The hard disk in the in-memory database systems will be used as data backup and to maintain the log files. Therefore in-memory database systems will need different optimisation methods to structure and organise the data as well as to make sure that the data is reliable.

To improve the computational performance of the CPU, engineers have put multi-level caches inside the CPU. This structure is called a memory hierarchy, which is a key concept of in-memory database system. The memory hierarchy includes TLB, L1 cache, L2 cache, L3 cache and main memory. The top level memory structure is faster than the bottom level memory structure, but is much smaller in size than the bottom level memory structure. In modern computer architectures, the data can only be accessed from the TLB directly. When a CPU tries to access a data record, it will start by first searching in the TLB, and if the record is not found, the CPU will search each level of the memory hierarchy until such record is found. Then the CPU transmits this data, level by level, to the TLB for accessing, a process which is called cache missing. Moreover, the CPU cache line is the smallest unit of transferring data from main memory to the CPU so, for example, if the size of a CPU cache line is 64 bytes, the CPU will load 64 bytes of continuous data from memory even if the size of required data is less than 64 bytes. As we can see, a cache missing can reduce the performance a lot in the in-memory database. Therefore, utilising the memory hierarchy (i.e. CPU cache) is necessarily for in-memory database systems. A general idea is to reduce the cache missing during query processing. It is required the related data be stored in the memory continuously to fill the cache line as much as possible, which means that the data structure needs to be optimised to allow possible related data stored together in the memory space.

Recently, there have been some in-memory database management systems proposed, such as SAP HANA, Hyper, VoltDB, MongoDB, Memcached and so on. The newest in-memory based and column-oriented database management system, SAP HANA [85, 86] was developed and implemented by SAP. The goal of SAP HANA is to handle heavy loads of data and the real-time complex query processing. SAP HANA supports many programming languages such as SQL Script, R and C (via ASL package). Even though SAP HANA supports both row-oriented and column-oriented data structures, the column-oriented store is recommended to use to store the data, since the column-oriented store can provide better performance for analytic queries and the data can be highly compressed.

1.1.4 Column-oriented Data Structures

TABLE 1.1: Sample Table

ID	Name	Country
1	Paul	Australia
2	Bob	Australia
3	John	Canada
4	Lean	USA

In a traditional relational database management system (RDBMS), the data is stored in a row-oriented structure. Given a table such as in Table 1.1, in a row-oriented structure, all attributes of a tuple (or row) are stored continuously and sequentially, which means the data is stored row by row as shown in Fig. 1.6. The row-oriented data structure is designed for transaction-based applications such as banking systems, online shopping systems and log management systems. Because in the modern storage hardware, sequential access is faster than random access and the transaction operations need to select/insert/update entire rows. For example, to select Paul's information, the database system only needs to scan three cells to retrieve the results. It is easy to see how the row-oriented data structure can respond to such operations quickly, since it can access each entire row instantly.

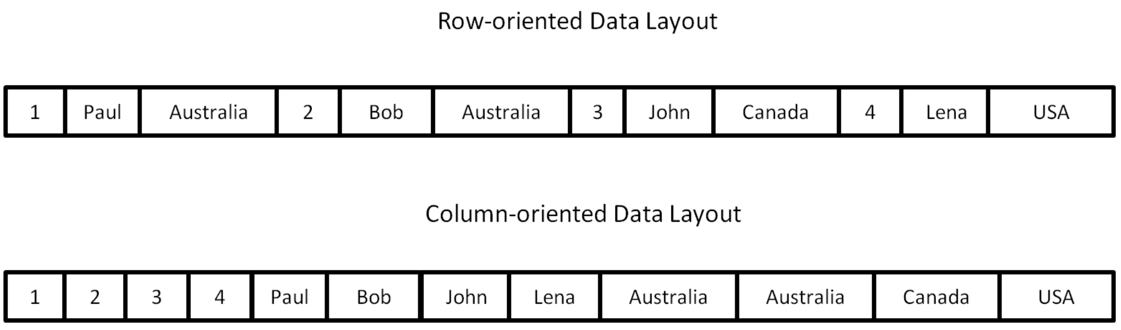


FIGURE 1.6: Row-oriented Structure versus Column-oriented Structure

On the other hand, analytic queries only need to aggregate a few columns to get the results. For example, to count the number of people living in Australia in Table 1.1, the database system needs to retrieve only the information from the column"Country" to get the answer. However, in the row-oriented data structure, this operation needs to scan and access this table three times to get the result, which is time-consuming. Hence, a column-oriented data structure is proposed to increase the performance of analytic queries, as discussed in [104]. They indicate that analytic queries can achieve better performance on column-oriented data structure. Meanwhile, they also mention that the row-oriented data architecture could not store data for such a system efficiently, since the random accesses will consume a lot of bandwidth between the CPU and the hard disk. In a column-oriented data structure, the data objects of one column are stored together as shown in Fig. 1.6. Following the previous example, it is easy to see that this query can be done very quickly on a column-oriented data structure as it only needs to access the table once to get the answer.

In addition, the compression techniques can be invoked in the column-oriented data structure to reduce the size of the data at the same time. This is because the objects in a column are similar; and they are stored sequentially in the storage space. Meanwhile, the data compression techniques will also improve the query performance, since the system can load more data into memory for query processing. Currently, there are two common light-weight compression methods that are used with column-oriented data structures. The first one is dictionary encoding, in which a dictionary is built to index frequent elements on a selected column. Then the elements of this column can be converted to the index number of dictionary, which means that less space is required to store the information for this column. Another common compression method is the run-length encoding, which is an improvement on dictionary encoding. The run-length encoding records the dictionary code with its last index in the column to save even more space than the dictionary encoding method alone.

1.2 Problem Statements and Methodologies

1.2.1 Problem Overview

Trajectory data offers the opportunity to learn movement behaviour of moving objects such as human beings, vehicles and animals. However, it is hard to store and manage trajectory data in traditional database systems, since its variable lengths and asynchronous sampling rates do not fit disk-based and tuple-oriented structures, which are the fundamental structures of traditional database systems. In this thesis, a novel trajectory storage system that is motivated by the success of column-oriented data structures and the recent development of in-memory based databases is implemented. In this storage design, the performance of query processing for trajectory data can be boosted via exploring the potential opportunities. To achieve this, three challenges need to be addressed. The first is how to convert the original trajectories into a column-oriented structure. The second is how to compress trajectory data to reduce the space consumed. The last challenge is how to process trajectory queries efficiently on the column-oriented structure with compressed trajectory data.

Methodology: In this thesis, an in-memory storage system to store and manage trajectory data, which is called SharkDB, is proposed. To solve such challenges, SharkDB contains two main components, the in-memory storage component and the trajectory query processing component. As shown in Fig. 1.7, the duty of the in-memory data storage component is to encode and compress the raw trajectories, and then to store them in the column-oriented data structure. The trajectory query processing component is responsible for processing trajectory data and answering trajectory-related queries in real time. Based on this design, the SharkDB implementation is split into three parts: data structure design, query processing algorithms implementation, and system verification (via comprehensive experiments). Moreover, it is also important to apply SharkDB to a commercial DBMS to support real applications. Therefore, this task is also included in the SharkDB parts are listed in the below:

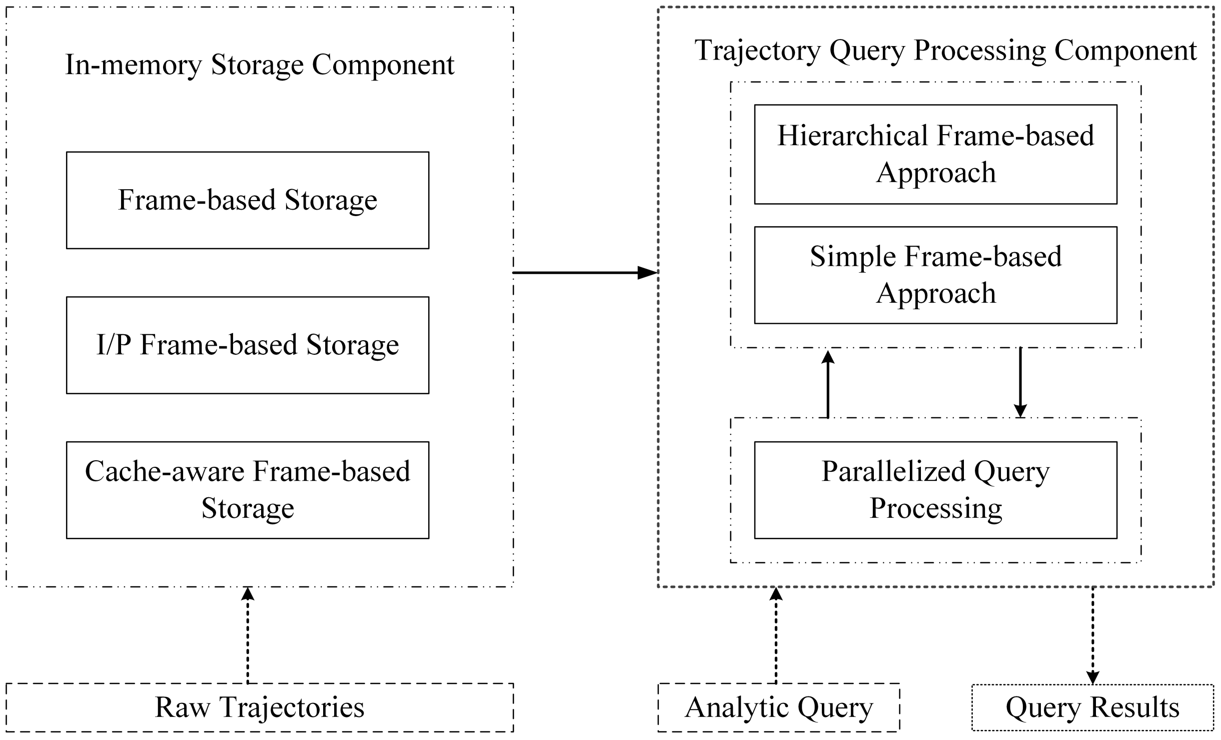

In-memory Storage Component
Frame-based Storage
I/P Frame-based Storage
Cache-aware Frame-based Storage
Trajectory Query Processing Component
Hierarchical Frame-based Approach
Simple Frame-based Approach
Parallelized Query Processing
Raw Trajectories
Analytic Query
Query Results

FIGURE 1.7: System Overview

1.2.2 SharkDB Design

Currently, many commercial relational database management systems (RMDBSs) have started to offer additional components or extensions to support spatial data types and operations. However, only a few simple spatial data types such as points, lines and polygons are considered and supported in these RDBMSs. On the other hand, more complex data types are needed to store trajectories because they contain continuous time information and the length of each trajectory varies (i.e. the number of sample points of each trajectory is different). Traditionally, to solve this problem, it is easy to consider the trajectory of a moving object as one database object. While this approach has a clear semantic meaning for each trajectory record, it has several obvious drawbacks. Firstly, most RDBMSs cannot handle records of variable lengths well. Secondly, many trajectory processing operations deal with, explicitly or implicitly, some segments of a trajectory only. For example, to find a point of interest (POI) closest to a trajectory, ideally only one part of the trajectory (that contributes to the identification of the resultant POI) needs to be used should that part be identified (using some indexing and filtering strategies). Treating a trajectory as whole cannot support this type of operation efficiently, as it needs to access the whole trajectory and then discard all but the relevant segment(s).

Methodology: To address this problem, a study on heterogeneous trajectory data was first completed, since trajectory data in real world database applications are heterogeneous by nature as the GPS signal is not always reliable or accurate. A heterogeneous trajectory dataset means the sampling rate of each trajectory is not consistent. For example, in such trajectory dataset, assuming the sampling rate of a trajectory is 30 seconds (i.e., reports its location every 30 seconds) and the sampling rate of another trajectory could be 60 seconds. If the time duration of these two trajectories is one hour, then the first trajectory contains 60 sample points and the second trajectory contains 120 sample points. This, however, can be problematic when these heterogeneous trajectories are processed directly, since most of trajectory query processing algorithms [5, 118, 69, 51] are designed for synchronised trajectory data, which means the sampling rate of each trajectory in the whole trajectory dataset must be consistent (e.g. the sampling rate of each trajectory is 30 seconds). To provide the evidences of these issues, a preliminary study on heterogeneous trajectory data was completed before the implementation. In this study, six common similarity measures are evaluated on asynchronous trajectory datasets to test how heterogeneous trajectory data can affect the effectiveness of similarity measures. The results show that the heterogeneous trajectory data can reduce the effectiveness of similarity measures. Having witnessed the limitations of existing spatial database systems and the issues of heterogeneous trajectory data, it requires a novel trajectory data storage system to support varied length and multi-dimensional trajectory data as well as to calibrate the heterogeneous trajectory data to synchronise the data.

Along with this evidence, SharkDB, an in-memory column-oriented storage system for storing massive amounts of trajectory data is introduced. As discussed before, in relational database systems, the column-oriented data structure is known to have better performance in analytic tasks that in comparison with row-oriented data structure, since analytic tasks only need a few columns from databases for obtaining the results. Similarly, trajectory queries could also be processed by scanning some segments of trajectories. Therefore, the advantages of column-oriented data structure in relational database systems provide an idea that the trajectory data can also be converted to a column-oriented data structure to avoid scanning the whole trajectory during query processing. In the SharkDB design, a novel read-optimised storage structure, which combines the advantages of in-memory computing and column-oriented data structure for analytical tasks, is proposed. Aligned with common in-memory database designs, where data compression is a key factor, SharkDB also supports the effective compressing of trajectory data and allows analytical query processing on the compressed data directly without the need to decompress the whole dataset. The data structure, called a cache-aware I/P frame data structure, is a carefully designed data structure with two-phase data processing, which combines both a column-oriented data structure and compression techniques to store and manage huge amounts of trajectory data. More specifically, the trajectory allocation phase will calibrate the trajectory data and convert the calibrated trajectory data into a column-oriented data structure. In next phase, the calibrated trajectories will be encoded and compressed as the elements of a cache-aware I/P frame structure in order to support efficient query processing. The details of the SharkDB design are discussed in Chapter 3.

1.2.3 Query Processing in SharkDB

The main reason for storing trajectory data in the frame based data structure is to answer trajectory based queries (e.g. window, kNN and similarity search), which can be used for many real applications. Similar to other analytic queries, most trajectory analytic queries only need to touch a few of trajectory segments to get the required answers. Therefore, treating a trajectory as whole does not support this type of operation efficiently, as it needs to access the whole trajectory and then discard all but the relevant segments. Thus, some efficient approaches to support trajectory analytic queries are required. There are two challenges in this problem, the first is how to perform efficient query processing on frame-based data structures in main memory. The second one is how to support query processing on compressed data without sacrificing much performance.

Methodology: Initially, the queries/operations that are supported by SharkDB are divided into three categories: basic operation, advanced operation and analytic operation. The basic operation category contains standard database operations such as INSERT, DELETE, SELECT and

APPEND. The advanced operation category includes window and kNN, which are fundamental trajectory based queries. The trajectory similarity search belongs to the analytic operation category. Meanwhile, to support these queries efficiently, several approaches are proposed for each category.

In particular, parallel based algorithms are proposed to support basic operations. For processing analytic operations, a simple frame based approach and a hierarchical frame based approach are presented. Moreover, as multi-core CPUs have become standard configuration for both servers and PCs, parallel versions for these proposed approaches are also implemented. For processing trajectory similarity searches, a sliding window based approach and a MBR+KMP based approach are introduced. The details of these approaches are presented in Chapter 4.

1.2.4 Performance Evaluation

Having established the new design and query processing approaches, it is important to have a set of experiments to evaluate the performance of SharkDB. First of all, as discussed previous, several approaches have been proposed for query processing, hence, it is necessary to compare the differences in performance of these approaches and demonstrate the performance gain. Secondly, since the storage system may have different workloads under different circumstances, the overall performance of SharkDB under different usages must be tested. Therefore a benchmark with different usages of workload models is required for evaluating SharkDB. However, there is no common benchmark is designed for a trajectory data storage system.

Methodology: Motivated by this, comprehensive experiments are conducted to illustrate the performance of SharkDB. The experiments contain two levels, operation level and system level. At the operation level, extensive experiments are executed on several large real world and synthetic trajectory datasets. At the system level, a carefully designed benchmark evaluation is proposed, which includes three categories to represent different system usages in the trajectory data storage systems. In addition, each category contains mixed operations with different weights. The first category is a typical workload, where the basic operations have the maximum weight. This category simulates the system working under a transaction workload model. The second category is an operational workload, which simulates the system serving trajectory based applications. Hence, advanced operations occupy maximum weight in this category. The last category is an analytic workload, which means this workload focuses mainly on analytic operations. All workload evaluations are based on real world datasets and the evaluation results of SharkDB and traditional trajectory data structures are also discussed. All experiments are included in Chapter 5.

1.2.5 SharkDB Implementation using SAP HANA

A further step is taken to tailor the I/P frame-based data structure into the relational database model and provide built-in support for trajectory query processing in SAP HANA, an in-memory platform developed by SAP for processing high volumes of operational and transactional data in real-time. SAP HANA can offer many benefits for SharkDB as it is a powerful in-memory database system and supports a column-oriented data structure. For example, a front-end system (i.e. a user interface) can be provided to users for using SharkDB directly, and APIs based on SAP HANA can be implemented to connect with real applications. Therefore, in this task, two major challenges need to be addressed when migrating this data structure into SAP HANA. The first challenge is how to fit the frame based data structure into a relational database model. The second one is how to process the query efficiently using the queries language (SQL like) supported by SAP HANA.

Methodology: To address the above challenges, a carefully designed data structure is presented, which is inherited from the previous I/P frame data structure. This redesigned I/P frame data structure retains the benefits of the I/P frame data structure in SharkDB, and is perfectly compatible with traditional table formats, which can work well in the SAP HANA. Moreover, extensive experiments also are conducted to evaluate this new approach. The implementation details are described in Chapter 6.

1.3 Contributions

To sum up the contributions of this thesis, first of all, this thesis proposes a preliminary study to observe that the trajectory data is naturally heterogeneous and such heterogeneous trajectory data can affect the performance of trajectory data analysis tasks. In this preliminary study, a set of reasonable transformation functions for the original trajectory data is devised, the variance of which is controlled by parameters. We then evaluate the similarity between original and transformed trajectories, and study how the similarity is reflected in six different distance measures. This thesis also identifies some problems encountered by the relational database system in handling large scale trajectory data as well as the fact that the heterogeneous trajectory data is hard to store in an uniform data structure. To solve these issues, a new frame based column-oriented data structure is proposed to calibrate the trajectories by time and support compact trajectory data storage. Then this frame data structure is extended into an I/P frame data structure. The I/P frame data structure uses P-frame to compress the trajectory data and reduce the memory consumption significantly.Finally, the I/P frame data structure is upgraded to a cache-aware frame data structure, which is optimised for CPU cache and can support the storage of large scale trajectory data in the main memory.

Secondly, several algorithms with parallel computing techniques are proposed to process trajectory queries efficiently. For basic trajectory queries (i.e. SELECT, DELETE, INSERT and UPDATE), the parallel processing is invoked to increase the query speed. Then an efficient hierarchy frame structure with parallel processing is discussed to answer advanced trajectory queries (i.e. window and kNN). For analytic trajectory query (i.e. trajectory similarity search), a MBR-KMP algorithm is implemented to increase the performance of similarity searches.

Thirdly, a set of experiments that include operation level evaluations and system level evaluations are conducted. The extensive operation level experiments with a range of trajectory data (up to 3 billion spatio-temporal points) show significant performance gain can be achieved. In the system level evaluations, a common benchmark is designed to evaluate the performance of SharkDB. To make a comprehensive evaluation, three categories of workload models are implemented. Experiments based on this benchmark with several real world large trajectory datasets are executed. The results demonstrate that SharkDB can achieve high performance compared with traditional database structures offering the promise of real time computing.

Finally, the previous I/P frame data structure is modified to fit into a relational database model, which is then used to store trajectory data in SAP HANA. This thesis also provides efficient built-in data types and operators for trajectory data using the query languages supported by SAP HANA. Extensive experiments are then conducted with large-scale real trajectory datasets. The results demonstrate that the new data storage model can achieve superior performance in ranges of spatial queries compared with traditional trajectory data storage structures.

1.4 Thesis Outline

The rest of the thesis is organised as follows: Chapter 2 is the literature review. Chapter 3 presents the design of SharkDB. The introduction and discussion of the high performance query processing of SharkDB is covered in Chapter 4. The evaluation of SharkDB is detailed in Chapter 5. The implementation details of SharkDB into SAP HANA are provided in Chapter 6. Finally, the conclusions of this thesis and future work directions are presented in Chapter 7.

Chapter 2

Literature Review

In this chapter, an overview of most of the important problems that are related to this thesis is provided First of all, since the trajectory data contains spatial information, and the techniques of the spatial databases are the basis of trajectory data management, therefore, the fundamentals of spatial databases are reviewed in Section 2.1. Furthermore, this thesis focuses on managing massive amounts of trajectory data, hence, the related work of managing trajectory data and querying trajectory data are covered in Section 2.2 and 2.3. In addition, this thesis also invokes the techniques of in-memory database management and column-oriented data structure, so the related work of in-memory database management and column-oriented data structures is presented in Section 2.4 and Section 2.5, respectively.

2.1 Fundamentals of Spatial Databases

Spatial data is usually a multi-dimensional (dimensions $\geqslant 2$) data type, hence it requires a special database system to manage such data, called spatial database. R.H. Guting [36] gives a formal definition of a spatial database as having three key elements, 1) a spatial database system must belong to the family of database systems, 2) spatial data types need to be supported in both the data model and the query language, and 3) spatial data types need to be supported by a spatial database system with efficient query processing (including both the indexing structure and algorithms). Based on this definition, Fig. 2.1 shows a paradigm of spatial databases.

First of all, for spatial data modelling, the most common definition of spatial data types is

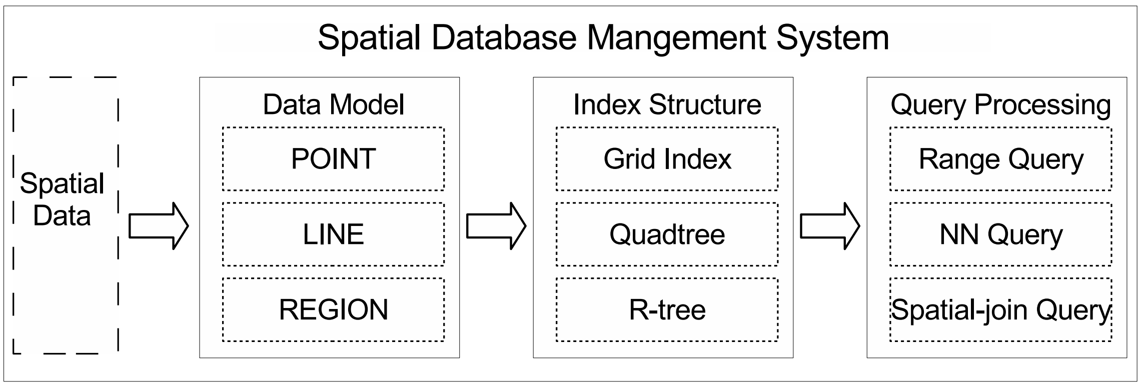


FIGURE 2.1: Paradigm of Spatial Database

proposed by [35]. In their work, three fundamental spatial abstractions named POINT, LINE and POLYGON, are defined. A POINT represents the position of a spatial object. A LINE is a finite sequence of points, which is used to record a curve in the spatial space. An area is modelled as a POLYGON, which is a chain of points. However, a POLYGON can only support simple objects (i.e. a region without a hole inside the region). On the other hand, [33] suggest that using only one spatial type to represent all spatial objects can reduce the complexity of query processing. Therefore, they propose a new spatial data structure, called SHARP, which is a pixel based data type. Subsequently, [37] propose a ROSR algebra, which extends POLYGON to REGION to support complex area based spatial objects. Nowadays, POINT, LINE and REGION have become the most common definitions of spatial data types.

For spatial index structures, the grid index is one of earliest well-known index structures, as proposed by [71]. The basic idea of a grid index is to partition the whole spatial space into small grid cells, and each cell uses a bucket to store all spatial objects in the cell. For example, given a set of spatial objects, such as Fig. 2.2 shows, the grid distribution and index structure are shown in Fig. 2.3. The buckets are stored and organised as an array, which can be accessed directly during query processing. However, grid index is not good at indexing spatial objects if the distribution of these spatial objects is highly skewed.

Quadtree [97] is a common indexing structure for spatial databases. Quadtree has been used in many applications and research works [96, 99, 119]. It has an unbalanced tree structure, and each leaf node in Quadtree represents a region or a point. A Quadtree internal node always has four points, which indicate four directions, NW, SW, SE and NE, and each point connects to

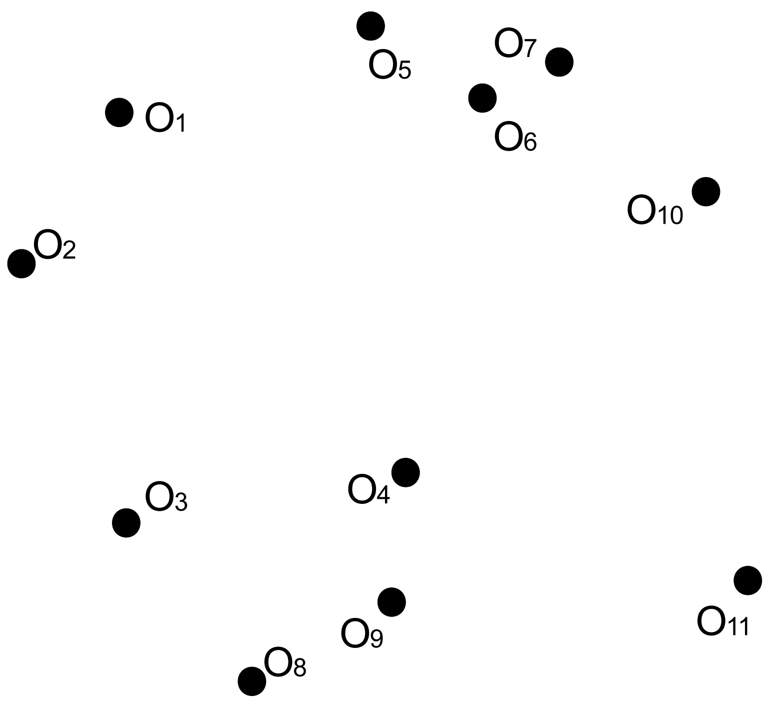


FIGURE 2.2: Example of Spatial Objects

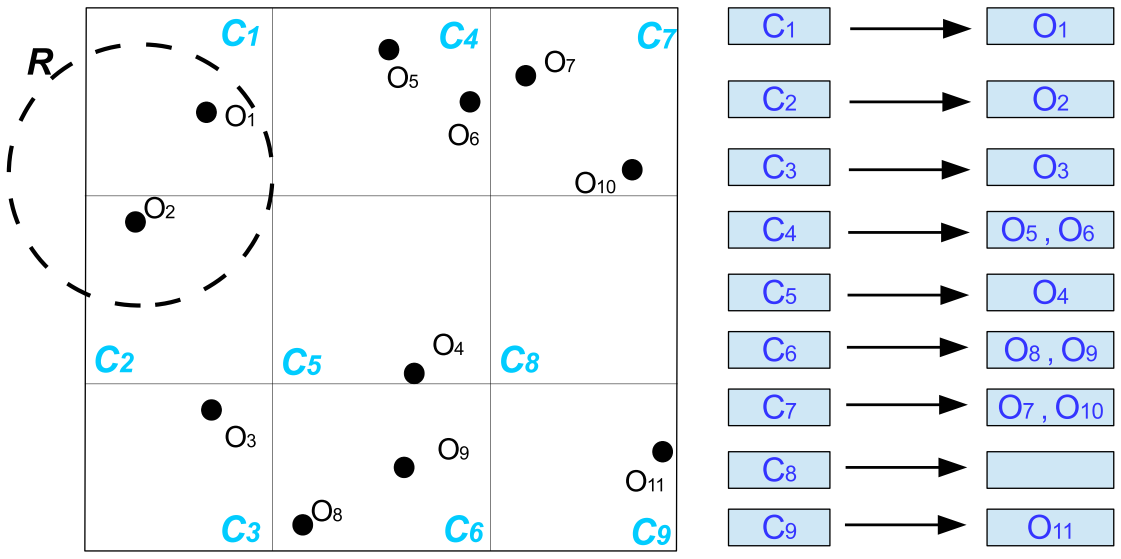


FIGURE 2.3: Grid Index Structure

a child node. Therefore, when inserting a new point to into a Quadtree structure, if the node is full (i.e. already has four objects), this node will decompose into four new child nodes. Therefore, Quadtree can support the indexing of highly skewed spatial datasets very well. For example, given a set of objects such as shown in Fig. 2.2, if a Quadtree index structure is used to build an index, the resultant region distribution and index structure are show in Fig. 2.4 and Fig. 2.5. As we can see, each node of Quadtree contains four points, where each point can link to a new entry or object or be empty. During the construction of the index structure, if the node E_2 contains more than one object in the same direction, then the node E_5 decomposes into four new child nodes to store the objects O_5, O_6, and O_7. Therefore, a Quadtree is built from top to bottom. Based on our example, it is easy to see that a Quadtree is not a balanced tree structure. The drawback of a Quadtree is that Quadtree is not a balanced tree, therefore, the query may need to search all entries in the Quadtree in a worst case scenario.

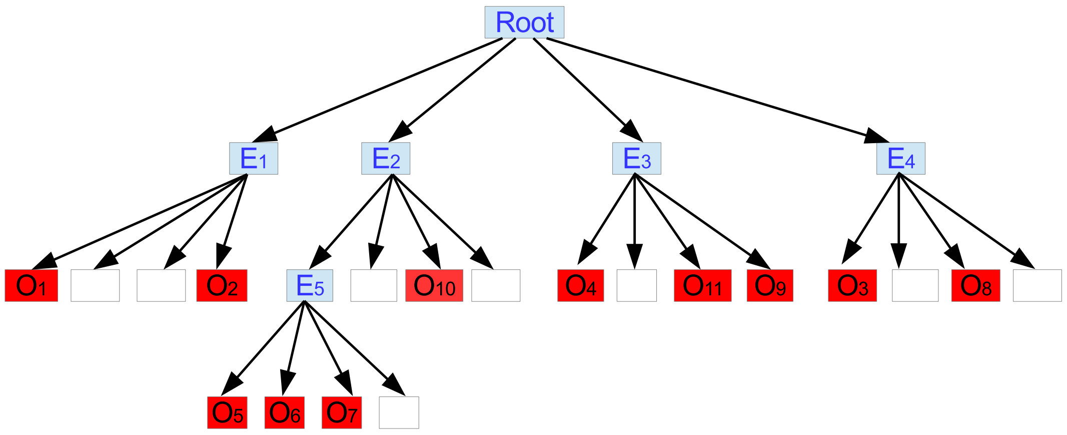


FIGURE 2.4: Example of Quadtree Region

Another popular index structure is the R-tree [38]. R-tree is the most common spatial index structure and has been widely supported in many commercial database management systems. In contrast to a Quadtree, an R-tree is a height balanced tree, which is an efficient tree structure. To achieve this, R-tree uses a minimum bounding rectangle to group and represent spatial objects. For example, as Fig. 2.6 and Fig. 2.7 show, the R-tree is built from bottom to top. As we can see, the object O_1 and O_2 is represented as the MBR entry $E5$, and E_1 is the bounding box of the lower level MBR entries E_3, E_4 and E_5. Therefore, the R-tree can be built as a balanced tree and all objects (i.e. leaf nodes) are stored at the same level. Many variants of the R-tree have been proposed to reduce the overlap between each MBR; and the R*-tree [4] has the best performance to minimise the overlapping of MBRs.

The first fundamental query type of a spatial database is the range query, which is to find all the objects that are within a given region. Generally, a spatial index structure can support the efficient

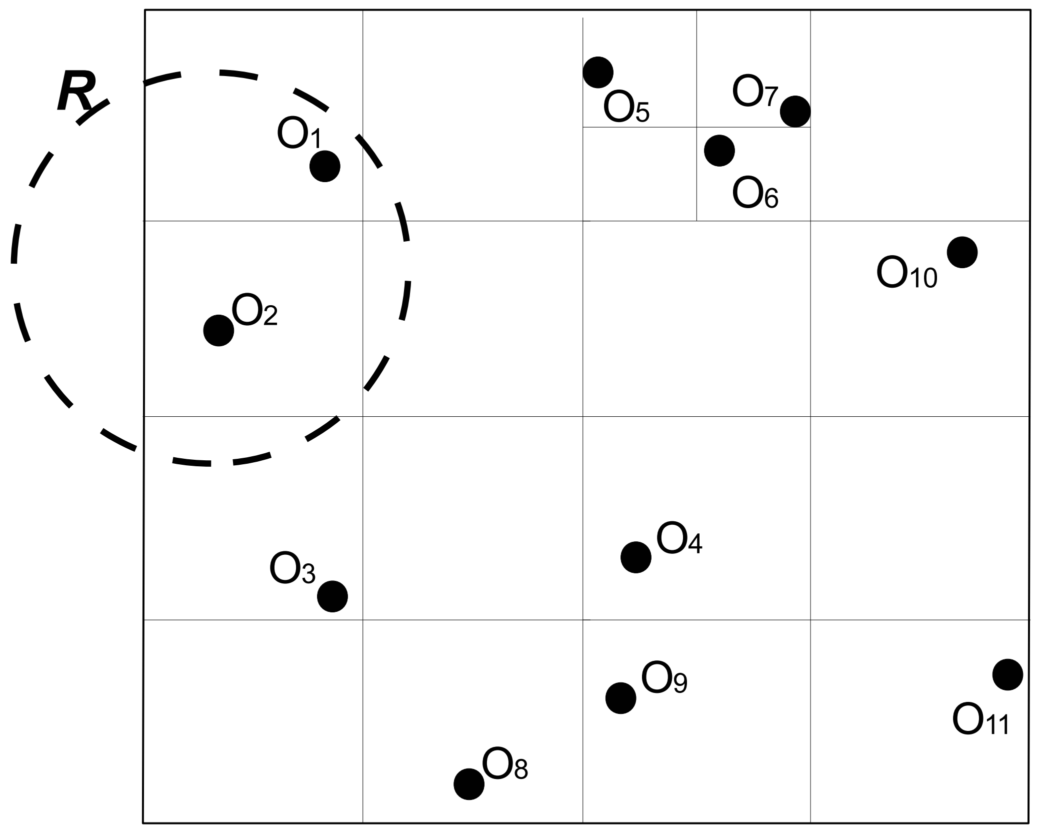


FIGURE 2.5: Example of Quadtree Structure

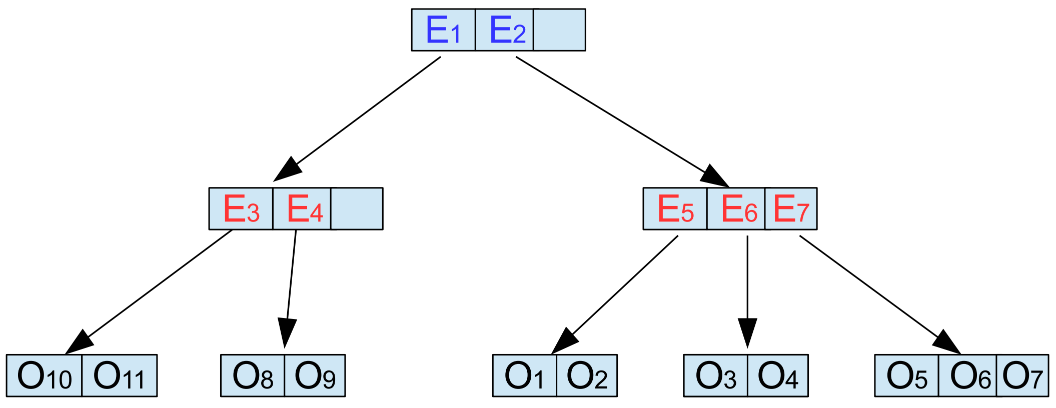


FIGURE 2.6: Example of R-tree Region

processing of a range query with spatial indexes. For example, for a grid index [71], given a query window, a grid index search will retrieve all grid cells that are either contained by or overlap the query window. Then grid index searches all objects in the retrieved grid cells to get the results. For instance, as Fig. 2.3 shows, given a range query R, the grid index will return grid cells C_1 and C_2

as a candidate set, and then it will search all the spatial objects (i.e. O_1 and O_2) that are contained in these cells and return the refined results. Quadtree indexes are also used for range queries to find all objects within a given spatial window. To retrieve these objects, Quadtree starts the searching task from the root, and if a node does not overlap the given spatial window, Quadtree will not continue to access that node and its children. Otherwise, the node will be visited and the objects within node will be investigated. Continuing the example from Fig. 2.5, given a circle range query R, Quadtree will start to traverse from the root, and then examine the next level nodes to determine whether they have an overlap with R. So E_4 and E_5 are checked and hence, O_1 and O_2 are found as the results. Range query processing on an R-tree starts at the root of the R-tree. Then each node (i.e. MBR) will be examined against query region, and if a MBR overlaps the query region, the R-tree will continue to visit its children until it reaches the leaf node. Otherwise, this node will be pruned out. Continuing our example from Fig. 2.7, the R-tree first searches at the root, and finds that E_2 does not overlap with R. Then E_2 with its contained MBRs and objects can be pruned safely and dismissed from the search. Then the R-tree continues to search E_1's leaf node, and E_7 and E_6 are pruned. Finally, objects O_1 and O_2 are found as results.

An alternative to the range query, the NN (kNN) query is another fundamental query type of a spatial database. Given a spatial point, the NN query is designed to find the closest point in the database. For processing kNN queries, Roussopoulos et al. [94] proposed an early algorithm based on an R-tree. Two important metrics, MINDIST and MINMAXDIST are defined in this work. MINDIST and MINMAXDIST are used as the lower bound and upper bound for the branch-and-bound based algorithm. Two search strategies, depth-first [94] and best-first [41] search can also be supported by R-tree.

One variate of the NN query is the reverse nearest neighbour (RNN) query. Given a query point, the RNN query is designed to find all the spatial objects in the database, which consider this query point as their nearest neighbour. The problem of the RNN query was first studied by [56], and they proposed an R-tree based algorithm to process RNN queries. After this, a series of works [123, 62, 109, 117, 126] to solve the RNN query are proposed in both euclidean space and road networks. Aggregate nearest neighbour query (ANN) is another important type of NN variations. The difference between an ANN query and an NN query is that an ANN query is designed to find the minimum aggregated distance from a point in the dataset to a set of query points, and the

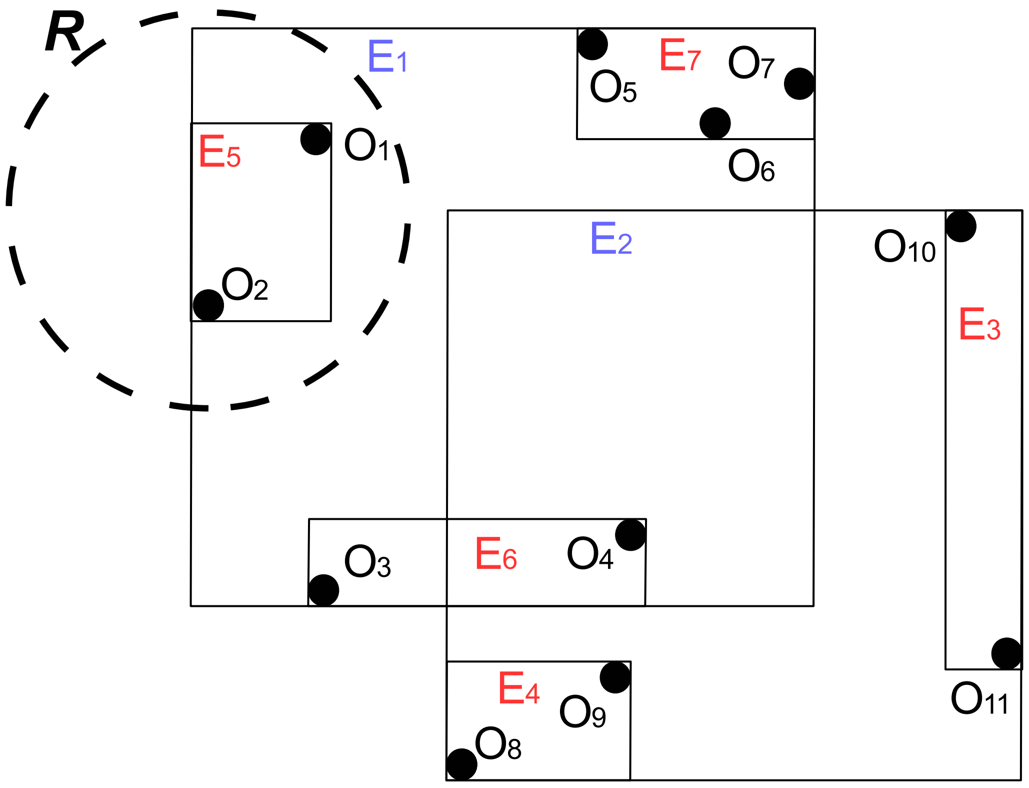


FIGURE 2.7: Example of an R-tree Structure

aggregated distance is calculated by a user-specified aggregate function (e.g. max, sum). There are many works that focus on ANN queries in both euclidean space [76, 77, 122, 72, 61] and road networks [125, 31, 78].

The last fundamental query is the spatial-join query. Given two spatial datasets D_1, D_2 and a predicate, a spatial-join query is designed to find the pair of objects O_i, O'_j that satisfy this predicate, where $O_i \in D_1$ and $O'_j \in D_2$. Brinkhoff et al. [15] studied spatial-join queries on both R-tree and R*-tree structures. They investigated the performance of different settings for R-trees and proposed two new algorithms to reduce I/O performance and CPU time. Subsequently, Huang et al. [42, 43] used distribution functions as the cost model to optimise the I/O performance of R-tree traversals. Another cost model for a spatial join, which is based on an analytical formula [113], was proposed by Theodoridis et al. [114].

Finally, as the paradigm shows, a spatial database management system (spatial DBMS) firstly needs to support the spatial data models (i.e. POINT, POLYGON and REGION). Then a spatial

DBMS must provide some kind of spatial indexing method to help to manage such spatial data. And, importantly, a spatial database system must be able to efficiently process a set of spatial queries. The earlier versions of spatial DBMSs were implemented as independent DBMSs. There are two types of such spatial DBMSs, the first type builds a spatial DBMS as a top layer on a traditional DBMS, such as [8, 18]. These spatial DBMSs use traditional data types to store the spatial objects, which will limit the performance of the spatial DBMS, and the spatial index structure is difficult to implement on these spatial DBMSs. Other early designs of spatial DBMSs [116, 1] built the spatial DBMS first as a subsystem and then integrated it with a traditional database via a middleware component. Hence, such spatial DBMSs can support both spatial data and spatial index structures very well. However, it is very hard to extend these spatial DBMSs to support new query types and spatial index structures, since they are already built into the system. Currently, most commercial DBMSs such as MYSQL, SQLSERVER and SAP HANA use the spatial extension plug-ins to provide spatial DBMS functionally, which can provide a flexibility for a spatial DBMS. Therefore, a new spatial component (e.g. index structure or query algorithm) can be easily added to the DBMS.

2.2 Trajectory Data Management

As shown in Fig. 2.8, topics related to on trajectory data management are reviewed in this section. Trajectory data is a multi-dimensional data, which contains two aspects information, spatial information and temporal information and the basis of trajectory data management is the index structures, which provide an efficient way to store and query trajectory data. Hence, Section 2.2.1 first reviews the trajectory data indexing structure. As trajectory data is a kind of spatio-temporal data, spatio-temporal indexing structures, which can support indexing of trajectory data, are also reviewed. But, the size of this trajectory data poses a physical storage challenge. Therefore, compressing trajectory data is becoming an area of significant research interest and related works are reviewed in Section 2.2.2. The most common and fundamental types of trajectory queries include range query, kNN query and trajectory similarity search. Meanwhile, the trajectory range query can be processed efficiently and directly by trajectory indexing techniques. In addition, the rest of the trajectory queries including kNN query and similarity search are reviewed separately in Section

2.3. These queries can be used in many real applications and in research areas such as trajectory clustering and trajectory mining.

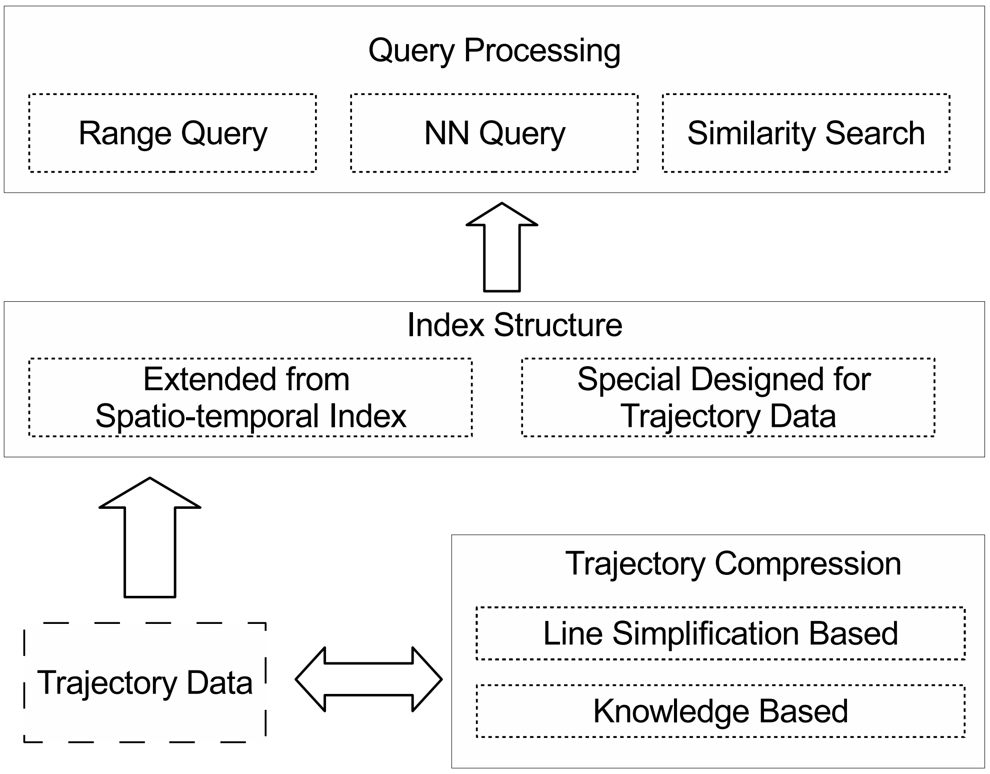


FIGURE 2.8: Trajectory Data Management

2.2.1 Trajectory Indexing Structure

To search and query trajectory efficiently and effectively, substantial data structures are deployed to store and index trajectory data. The most popular data structure is the R-tree [38]. However, the original R-tree structures are not good at supporting trajectory data, since trajectory data requires a particular spatio-temporal data structure, where the R-tree structure is proposed for spatial data and not specifically for spatio-temporal data. Therefore, to make the R-tree based structure support spatial-temporal queries, many optimisations are proposed. Early versions of R-tree based index structures for spatio-temporal data are Historical R-trees (HR-trees) [70] and 3D R-trees [55]. HR-tress are based on timestamps and a single R-tree will be created for a timestamp to index all objects that belong to that timestamp. Meanwhile, to save space, the same objects that appear in different times are only stored once, and all R-trees share the same entry point. In contrast to HR-trees, 3D R-trees maintain only one R-tree structure and the temporal information is stored as a new dimension. In general, there are two ways in which to index trajectory data, one way is based on sample points of each trajectory; and another way is based on the whole trajectory. If an R-tree index structure is based on a sample point, then the closest sample points will be moved together to fit onto the same disk page. Although, this type of index can handle searches on a large trajectory set with accepted performance, the cost of building the index and re-constructing the trajectory would not be acceptable. For an index based on the whole trajectory, the performance will be very limited by large overlaps in the bounding rectangles for long trajectories. Therefore, to make the R-tree based structures capable of supporting the indexing the trajectory data and processing the trajectory queries efficiently and effectively, many optimisations are proposed.

TB-tree [82] uses a hybrid tree structure to store and index both spatial and temporal information, but it is still not good for processing the long trajectories, since indexing long trajectories can produce large bounding rectangles. And this issue is not addressed yet. TPR-tree [101] and TPR*-tree [111] invoke the prediction model to predict the future positions for the moving objects. These index structures mainly focus on continuously moving objects rather than historical trajectories. On the other hand, partitioning trajectories into segments becomes a new direction to improve the query performance. Rasetic et al. [91] derived an analytical cost model to control the splitting process for a trajectory into segments based on given query. Their system is optimised for trajectory range queries only and not for trajectory kNN queries or trajectory similarity searching. SETI [17] 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 searching performance. PIST [13] partition the sample points rather than partition the trajectories. Similarly, TrajStore [28] proposed a new adaptive storage system that indexes the trajectory data based on Quadtree index and clustering methods.

These algorithms or systems are designed based on a hard disk based system, which means that I/O cost between hard disk to memory is the main concern. However, in a main memory based system, I/O cost is no longer an issue since all of the data are now stored in the memory. That means that these algorithms and systems are not optimal for the in-memory systems.

2.2.2 Trajectory Compression

In this section, existing techniques for trajectory compression techniques are described. An overview of line simplification based trajectory compression methods is first introduced. Then, the knowledge based trajectory compression methods are presented.

Line Simplification Based Trajectory Compression

The first line simplification algorithm, called the DP algorithm, was proposed by Douglas and Peucker [29]. This work is based on a splitting and merging strategy. In addition, the algorithm uses a top-down method to check if the deviation from a straight line is acceptable, where the merging algorithm combines the pair of trajectory segments with the least deviation. The time complexity of the original DP algorithm is $O(N^2)$,and [40], and [84] improve the DP algorithm with a better $O(NlogN)$ time complexity. The main disadvantage of these algorithms is that they may lead to undesirable approximation results.

For trajectory compression, there are many other factors that must be considered during compression processing such as temporal information and the velocity of moving objects. Meratnia et al. [67] proposed an SPT algorithm which is implemented by a greedy approach called an opening-windows approach. Their method considers the trajectories' temporal information as the simplification factor. In their work, they indicate that the previous line based error criteria are not suitable for trajectory data, since such criteria only consider spatial information. Hence, they propose a new error criteria whereby errors are also measured via the distance between pairs of temporally synchronised positions, which is called the synchronous Euclidean distance (SED).

Two single pass approximation methods based on sampling, which take advantage of the spatial locality and temporal timeliness inherent in trajectory data were proposed by Potamias et al. [87]. To achieve this, they not only consider the additional temporal information, but also consider other parameters such as velocities and coordinates. The first algorithm, called a threshold-guided algorithm, uses a safe area of next point to solve the min-$\#$ problem in a greedy manner. The second algorithm, called a STTrace sampling algorithm, is implemented by using a bottom-up strategy, where the SED is minimised in each step.

Muckell et al. [68] proposed an algorithm called the Spatial quality simplification heuristic

(SQUISH), which is based on the priority-queue data structure. This method is an online trajectory compression algorithm, which uses local optimisation to select the best subset of trajectory sample points and permanently removes redundant or insignificant trajectory sample points from the original trajectory. This method can increase compression speed with a higher accuracy.

Buragohain et al. [16] presented a linear simplification algorithm called PieceWise Linear Histogram (PWLH), which is extended from ACPA. In this algorithm, a trajectory is first divided into a sequence of variable-length data segments. All the sample points in each segment are mapped by a line, which minimise the maximum distance from these points to the line.

In the work proposed by Chen et al. [22], more factors are considered in order to achieve a higher trajectory simplification performance, including both the shape skeleton and the semantic meanings of a trajectory. Their algorithm, called a trajectory simplification algorithm (TS), assigns different point headcounts in terms of the product of the average heading change and the distance between each segment. After that the min-ε problem is solved in each segment by using local a weighting process. The main disadvantage of this method is that it is not robust when the sampling rate is not uniform.

Chen et al. [21] introduced a fast $O(N)$ multi-resolution polygonal approximation algorithm. They extend the integral square error (ISE) [80] and the local ISE [25] with SED as the integral square synchronous distance error criterion, where ISE is the sum of square of perpendicular distances between all dropped trajectory sample points. In their algorithm, a bottom-up multi-resolution method and a polygonal approximation method are used to construct the initialised approximated curve under a priority-queue structure. Once the polygonal curve is initialised, two finely tuned algorithms, LSSD and ISSD, are used in order to achieve the desired level of quality. The time complexity of this algorithm is $O(1)$ with pre-computing of $O(n)$ time.

The algorithm for the Path Nearest Neighbour based on Compressed Trajectories query (PNN-CT query) was proposed by Shuo et al. [100]. This algorithm has two main phases: trajectory compression and PNN query searching. For the compression phase, their algorithm is based on a linear prediction model to compress the trajectories with lossy compression. For the PNN query searching phase, they introduce a three-step solution to decompress the trajectory data and perform the PNN query search. In the first step they use the sample points and the corresponding meta-data to specify a tight searching range by adopting the network expansion method and branch-and-bound strategy. In the second step, a reconstruction algorithm is used, which is based on probabilistic models to account for the uncertainty when decompressing the trajectory segments in the candidate set. Finally, they adopt an effective combination strategy to find the PNN with the highest probability and with the highest performance and accuracy.

Knowledge Based Trajectory Compression

Most trajectories are generated by vehicles, which means that for land-based vehicles, road network constraints can be used to compress the trajectory data.

Brakatsoulas et al. [14] proposed two methods to map a trajectory onto a road network by matching geometrics. For the first method, they use an incremental match of the position trajectory sample points by pursuing a local match of geometries, which means matching a portion of the trajectory onto a path in the road network by using a measure composed of distance and angles between the curves. This method can trade off the accuracy and speed of computation. The second method uses the global match mapping for the entire trajectory to a candidate curve in the road network. To achieve this, two similarity measures Frechet distance and weak Frechet distance are used in association with two different map-matching algorithms to guarantee finding a matching curve with optimal distance to the trajectory.

A new approach to trajectory compression under road network constraints is to use the shortest path for compression. Therefore, a trajectory can be coded as a sequence of the shortest path codes. Kellaris et al. [50] proposed an approach to solve the map matched trajectory compression problem (MMTC). They first deploy a Map-Matching algorithm to map the trajectory onto the path in the road network. Then they propose an offline algorithm to compress the trajectory by using shortest path. This approach can achieve the optimal solution, however, the offline algorithm has a high computational cost, since it must calculate every possible shortest path. Hence, they proposed another approximate algorithm which supports online computation and is based on a greedy approach. They use the Minimum Description Length (MDL) as the distance measures to keep the result optimal.

Tao et al. [112] proposed an algorithm to solve the k-skip shortest path problem. They built a k-skip graph to store the simplified road network, which occupies less space than storing underlying road network. The main purpose of this work is to significantly increase the speed of shortest path searching. To achieve this goal, they use the both global reach and local reach algorithm to bound the searching area. There are two possible cases that use this approach: 1) if the start and end point are not in the k-skip graph, then they just insert these two points into the k-skip graph and rebuild the super edge from these two points to its direct neighbours; and 2) if start and end points are in the k-skip graph, then they just find the shortest path directly via the k-skip graph.

The current researches on the trajectory compression do not consider the trajectory synchronisation problem, which the compressed trajectory data sets are heterogeneous, even if after decompressed. Therefore, playing with the heterogeneous trajectory data is very hard, since such data set can reduce the effectiveness of query processing tasks (e.g. trajectory similarity search).

2.3 Trajectory Query Processing

2.3.1 Nearest Neighbour Query over Trajectory Data

The nearest neighbour (NN) query is the traditional and fundamental research area in spatial database, and large numbers of papers have been published in this area in recent years. An overview of the NN query over trajectory data is provided in this section. There are two key approaches for NN query over trajectory data: using a point to find its nearest trajectory called nearest neighbour queries over trajectories; and using a trajectory to find its nearest points called reverse nearest neighbour queries over trajectories.

Nearest Neighbour queries over trajectories

The growing popularity of GPS enabled mobile devices, such as tablet computers and mobile phones, generate millions of trajectories, which opens up a new area of spatial-temporal database applications. The nearest neighbour query over trajectory is used to find a closest trajectory from a given query point in the trajectory dataset. As the most important query, there has been a lot of work done on the solving nearest neighbour query for trajectories.

The first such work on NN query for trajectories by Kollios et al. [54] limited to a 1D space. Thus each trajectory segment is considered to be a linear function with time information. To represent the data, they use a dual transformation based method to map a point into a transformed plane and vice-versa. Moreover, they extend this solution into a “1.5-dimensional case”, which allows the trajectories to be represented in the plane with a restriction on the number of line segments fused to depict movement. Meanwhile, this work uses a HB-tree [63] as the indexing method rather than an R-tree [38] to obtain better performance. Even if this method can be extended to “1.5-dimensional”, it is still very hard to extend this method to a 2-dimensional space, where the trajectories of the points belong to a three-dimensional space.

Using Voronoi diagrams as the index structure, Zheng and Lee [131] proposed a Voronoi diagrams based method to solve the kNN query. But the method is limited to solving a single NN (i.e. k = 1), and it is not easily extended to support kNN (i.e. $k > 1$), and maintaining the Voronoi diagrams is very expensive. Song et al. [102] proposed another method with an R-tree index to process queries with kNN. In this method, an algorithm called a dual buffer search is used instead of the naive branch-and-bound algorithm, and the number of pages accessed in the R-tree is reduced heavily. Benetis et al. [5] proposed an efficient algorithm for solving NN queries and reverse NN queries for trajectories. This method is use a TPR-tree as the indexing structure.

Tao and Papadais [108] provide a solution for the continuous nearest neighbour (CNN) query, which means the trajectories data keeps coming as streaming data and the query point is changing continually. They use an R-tree based method to process both the CNN query and the kCNN query. Subsequently, Tao and Papadais [110] proposed another type of NN query called a Time-parameterised (TP) query, which uses a TPR-tree to index the trajectories. In their study, they extend their method to support CNN queries, but this causes multiple TRP-tree searching to get the correct result. This leads to extra CPU and I/O cost which may increase to very high when the number of NNs (k) is large.

Raptopoulou et al. [90] improved on the previous method in [6], and proposed a new method to more efficiently process NN queries on trajectories. This method can guarantee that only one query is issued per time interval, which can significant reduce the I/O and CPU cost. To achieve this, they make an assumption that the future locations of the current object can be calculated by their current movement status such as velocity, and they assume that the velocity is constant during the time interval. However, this assumption is too strong to support many applications. In reality, velocities reported by GPS devices, are continually being updated in relation to both speed and direction.

The validity region is used by Zhang et al. [130] to answer the NN query, where the results can be re-evaluated when the trajectory's sample points are in the same validity region. This approach can reduce the computation load significantly, but is limited by the fact that it focuses only on stationary objects.

Xiong et al. [121] proposed an algorithm, called a Shared Execution Algorithm, to continuously solve a collection of concurrent kCNN queries. The scalability of SEA-CNN is achieved by employing a shared execution paradigm on concurrently running queries. Shared execution entails that all the concurrent kCNNs along with their associated searching regions are grouped into a common query table. Thus, the problem of evaluating numerous kCNN queries reduces to performing a spatial join operation between the query table and the set of moving objects (the object table). Therefore, SEA-CNN does not make any assumptions related to the status of movement of objects such as velocities or the shape of trajectories, and also reduce both the less both CPU and I/O cost.

Two efficient and scalable algorithms were proposed by Yu et al. [127], based on grid indexing, to monitor kNN queries on the moving objects. The first method is dependent on indexing the objects themselves (Object-Indexing), and the second one is based on the indexing the queries (Query-Indexing). In both methods, the index takes the form of a grid structure, which represents a canonical partition of the 2D space.

The previous works assume that each query has a query point only. However, a kNN query may contain more than one query point for each query in many applications. Hence some newer technologies, which support multiple query points in one query, are presented the below.

Chen et al. [24] studied a new query, called k Best-Connected trajectories (k-BCT), to find a trajectory with good connections to given query points. To answer this query, they proposed an incremental kNN based algorithm (IKNN), which searches the nearest trajectory points that relate each query point incrementally, and calculates the aggregate distance from the query points to such trajectory points. In this method, they use both best first and depth first algorithms with R-tree indexing to retrieve the nearest trajectory points, and use the r-NN algorithm to minimise both the upper bound and lower bound. Therefore this algorithm can keep the I/O and CPU cost at a low level and likewise, minimise the memory cost.

The current state-of-the-art trajectory NN query was proposed by Zheng et al. [135], and is called an activity trajectory similarity query (ATSO). It returns k trajectories that cover the query activities and yield the shortest minimum match distance from given a sequence of query points. In their queries, each trajectory sample point contains several activities such as sports, shopping etc. They developed a hybrid grid index structure, called GAT, to organise the trajectory segments and activities hierarchically. By using GAT, the pruning speed can be increased significantly, which means both I/O and CPU remain low. Finally, the authors extend their method to support order-sensitive ATSO queries.

Reverse Nearest Neighbour queries over trajectories

Benetis et al. [5] proposed algorithms for RNN queries on the current and anticipated future positions of points moving continuously in the plane with the assumption that the movement pattern of objects can be predicted using a linear function of time. They use the TPR-tree as the index structure for moving objects and the algorithms proposed are based on the best-first and depth-first paradigm. Later, they revised and extended their work to support the RkNN queries which are the $k > 1$ RNN queries.

The TPR-tree is good for indexing moving objects but is hard to maintain since changing and updating the existing moving objects is very expensive and consequently, its derived methods are inefficient. Xia and Zhang [118] proposed the continuous RNN query (CRNN) without making any assumption about the moving objects. They use a grid index for the moving objects and query points. In addition, they maintain two kinds of monitoring regions with different shapes, called pie-shaped and circled-shaped reigns for query processing. However, there are two disadvantages of this approach. Firstly, they are limited to only monochromatic RNN queries; and secondly they always assume a constant worst case scenario at every time interval.

Kang et al. [48] improved on this work to overcome these drawbacks. They proposed an algorithm called IGREN, which is applicable to both monochromatic and bi-chromatic RNN queries. The main idea of this algorithm is to initially identify a single region R around the query object and a set of objects S such that only R and S need to be monitored to trigger subsequent changes to the answer. The incremental aspect comes from the fact that each execution instance of IGERN updates the shape of R and the objects in S. Then, subsequent executions of the algorithm will need to monitor only R and S rather than the whole space.

The previous work on spatial queries was designed for querying original trajectories (uncompressed trajectories) only and optimised for disk based systems. Therefore, such algorithms cannot be implemented directly into our system.

2.3.2 Trajectory Similarity Search

Given a query trajectory, a trajectory similarity search of a dataset is used to find the trajectory which is located at the minimum distance to this query trajectory. The distance between two trajectories is calculated by a trajectory distance function. Different trajectory distance functions can affect the efficiency and effectiveness of trajectory similarity search in different ways. Hence, in this section, the most popular trajectory distance functions (i.e. distance measure) that across four categories are reviewed.

Euclidean Distance Measure

Euclidean distance, also known as L_2-norm, is a distance measure used for a variety of applications. Given two trajectories T_1, T_2, the Euclidean distance $d(T_1, T_2)$ can be calculated as, $d(T_1, T_2)$ = $\frac{\sum_{i=1}^{n} d(p_{1,i}, p_{2,i})}{n}$, where $d(p_{1,i}, p_{2,i})$ is the distance between the two sample points. Euclidean distance is easy to implement and index with many access methods, and it is parameter-free. In addition, the complexity of the Euclidean distance measure is linear, which means it can handle a large trajectory dataset. Euclidean distance is proposed as a distance measure and is one of most commonly used similarity functions since the 1960s [88, 81, 32, 51]. Later, Euclidean distance was also extended to measure the distance between trajectories [26, 92, 47, 98, 107], since trajectories and time series have similar representations.

Dynamic Time Warping based Measures

Dynamic Time Warping (DTW) is a well-known algorithm for finding similar trajectory patterns between two trajectories. The definition of DTW uses a recursive method to search all possible point combinations between two trajectories for the one with minimal distance, and DTW can be converted to dynamic programming very easily. DTW allows one to find a similar pattern between two given trajectories, which can be of different lengths, and with or without time information. Moreover, the original DTW similarity measure is also parameter-free. For example, if two trajectories are separately generated by a slowly moving object and a fast moving object, DTW can still report their similarity pattern. DTW was first introduced to compute the distance of time series [69]. In the 1980s, [58, 103, 83, 73] introduced DTW to measure trajectory distance. For a huge data set, DTW is time-consuming and I/O-intensive. To speed up the DTW algorithm and reduce I/O cost, several pruning methods have been introduced such as the FastMap method and the lower bound method [95, 124].

Piecewise Dynamic Time Warping (PDTW) [51] is another dynamic time warping based similarity trajectory function, which is an improvement of DTW. PDTW speeds up DTW by a large constant c, where c is data dependent. PDTW uses two steps to calculate a similarity trajectory pattern. The first step, called Piecewise Aggregate Approximation (PAA), cuts a given trajectory into c pieces, where $[p_{c*(i-1)+1}, p_{c*(i-1)+2}, \cdots, p_c * i]$ is i-th piece. For piece i, PAA computes $\bar{p}_i$ as a representative point and transform trajectory T into piecewise approximation $\bar{T} = [\bar{p}_1, \bar{p}_2, \cdots, \bar{p}_N]$. Then, in the second step, PDTW processes the DTW distance to find similar trajectory patterns between transformed trajectories $\bar{T}_1$ and $\bar{T}_2$.

Edit Distance based Measures

Edit distance with Real Penalty (ERP) [19] is an edit distance (ED) based trajectory similarity measure. ERP uses L_1-norm as the distance measure. Introducing L_1-norm makes ERP a metric measure, which is a significant advantage over DTW and LCSS, as metric measures allow for efficient pruning. In addition, ERP distance is defined on normalised trajectory data for amplitude scaling and global spatial shifting. ERP normalize a trajectory T by shifting it by its mean (μ) and scaling it by its standard deviation(σ): $Norm(T) = [\frac{p_1-\mu}{\sigma}, \frac{p_2-\mu}{\sigma}, \ldots, \frac{p_n-\mu}{\sigma}]$.

Edit Distance on Real sequence (EDR) [20] is another edit distance (ED) based trajectory similarity measure. EDR also uses a threshold ε to detect matching sample point, similar to LCSS. Like ERP, EDR also uses normalised trajectory data, in order to be invariant to scaling and shifting. In contrast to ERP, for EDR, for each sample point p_i in T, the position values of x, y are normalised by using the corresponding mean (μ_x), (μ_y) and standard deviation (σ_x), (σ_y), respectively: $Norm(T) = [(\frac{p_{1,x}-\mu_x}{\sigma_x}, \frac{p_{1,y}-\mu_y}{\sigma_y}), \cdots, (\frac{p_{n,x}-\mu_x}{\sigma_x}, \frac{p_{n,y}-\mu_y}{\sigma_y})]$. The matching defined by EDR is $match(p_i, p_j)$ for a pair of trajectory sample points, where $p_i \in T$ and $p_j \in T', T \neq T'$. $match(p_i, p_j)$ is true if and only if $|p_{i,x} - p_{j,x}| \leq \varepsilon$ and $|p_{i,y} - p_{j,y}| \leq \varepsilon$, where ε is the matching threshold. If $match(p_i, p_j)$ is true, the $subcost$ (i.e. edit distance) between p_i and p_j is 0, otherwise the $subcost = 1$.

Longest Common Subsequence based Measures

Some similarity measures work well based on the assumption that the trajectory data is clean. However, the trajectory data generated by GPS devices is not clean enough due to device accuracy limitations, bad GPS signals, and other factors. Therefore, a similarity measure which is more robust for processing low quality trajectory data attracts great research interest. Longest common subsequence (LCSS) a popular measurement used for string similarity, [44, 93] can also be applied as a trajectory similarity measure. For detecting matching sample points like matching string characters, a threshold ε is used, and if the distance between two points is less than ε, they are considered to be a match. The basic idea of LCSS is that it allows some unmatched sample points to match some sequences in trajectories. LCSS is good for processing with low quality trajectory data (i.e. noisy trajectory data), from which similarity trajectories can be determined with a reasonably high accuracy. However, it may lead to some inaccuracy, since it does not consider various unmatched sequences in trajectories.

2.4 In-Memory Database System

The in-memory technologies that are related to this thesis are reviewed in this section. In-memory database systems store their data in the main memory rather than on a hard disk device, which can provide high speed, randomly access. Therefore, in-memory database systems need to use different optimisations to structure and organise data, as well as to make them reliable. There are two main tasks for the in-memory database system, online transaction processing (OLTP) and online analytical processing (OLAP). Since SharkDB is a OLAP system, this section will mainly focus on work that related to OLAP systems, including in-memory indexing, and parallel computing for in-memory database systems.

2.4.1 In-Memory Index

The common tree based index structures like B-trees [27] and R-trees [38] are designed for block oriented storage (e.g. hard disk device), which optimise the sequence reading, lose much their appeal when applied to an in-memory database. Hence, a wide variety of index structures have been proposed and evaluated for in-memory databases. T-tree index structures [59], designed explicitly for in-memory databases, have been widely accepted as a major index structure. T-tree which is an improvement on both AVL tree [2] and B-tree, is a balanced binary tree with nodes containing more than one item. The main difference between T-tree and traditional tree-like index structures is that T-tree stores the data point directly in the node and does not care about the depth of the tree, since traversing trees is much faster in memory than on a hard disk device.

The traditional indexes cover all data equally, even if some data are needed often, and some will never be accessed. Therefore, an adaptive indexing technology is used to index the data in the dynamic workload, which means that such an indexing structure will focus only on the columns and the specific key ranges that are actually queried and avoid the need to incur the cost of full index construction. Idreos et al. [45] proposed a hybrid adaptive indexing method, which combines adaptive merging and database cracking techniques, to index the data in both in-memory and column-oriented databases. Their indexing method is specifically designed for in-memory technology, which uses the CPU cache line optimisation data partition algorithm. Meanwhile, their method can offer a light-weight adaptation to refine the index efficiently by releasing the power of in-memory technology, since an index structure and dataset that is stored in the main memory can be re-constructed very quickly.

During the query processing, the data is loaded into the CPU cache and then sent to the CPU's registers to process, since the data in the main memory can be accessed by the CPU directly. Bernstein et al. [9] studied the performance of several commercial database management systems in main memory and found that a significant portion of the execution time is spent on second level data cache misses and first level instruction cache misses. So, improving cache behaviour is going to be an imperative task in in-memory data processing. Rao and Ross [89] proposed Cache

Sensitive B+-trees, called CSB+-Trees. They optimised the B+-tree to store all the child nodes of any given node contiguously, and to keep only the address of the first child in each node to fit the CPU cache line. The rest of the children can be found by adding an offset to that address. Since only one child pointer is stored explicitly, the utilisation of a cache line is high. There are two types of CSB+-Trees: segmented CSB+-Trees and Full CSB Trees. Segmented CSB+-Trees divide the child nodes into segments, hence nodes within the same segment are stored contiguously and only pointers to the beginning of each segment are stored explicitly in each node. And full CSB+-Trees preallocates space for the full node group and thus reduces the split cost.

2.4.2 Parallel Computing on In-memory Database

In general, join operations are the most important DBMS operations. A Join is a way to combine tuples of two or more tables. Thanks to higher bandwidth of memory than hard disk, processing join operations in parallel is available on multi-core CPUs in in-memory DBMSs. The recent research into parallel joins in in-memory DBMSs mainly focuses on equal-join, which allows the selection of tuples from both relations which satisfy a given equality.

Blanas et al. [11] proposed a new join algorithm, called a no partitioning join, which is a direct parallel version of the canonical hash join. This algorithm does not depend on any hardware-specific parameters, therefore, the partitioning phase requires multiple passes over the data and can be omitted by relying on modern processor features. During processing, both input relations are divided into equal-sized portions and each part is assigned to a worker thread. A shared hash table is populated by worker threads and can be accessed by all worker threads. Because the hash table is shard among all participating threads, concurrent insertions into the hash table must be synchronised. Therefore, the whole hash table is divided into multiple buckets, each bucket is protected by a latch that a thread must obtain before if can insert a tuple, but there is no latch needed for threads reading the hash table. However, this mechanism may reduce the performance when multiple threads wish to write a tuple into same bucket, since only one thread can write at one time and rest of the threads have to wait.

The previous algorithms do not consider the hardware very carefully, but another category join algorithm, called a partitioned join, partitions the hash table into cache-sized chunks to reduce cache misses and improve performance. After partitioning in the hash table, the number of chucks is very large since the size of cache is small and this could be cause a different type of cache problem, which may cause TLB missing. To reduce both cache missing and TLB missing, a hierarchical partition based join algorithm, called radix partitioning was proposed by [66], which partitions the input data in multiple passes. In practice, each pass looks at a different set of bits from the hash function. Kim et al. [52] proposed a parallel version of the radix join algorithm, which subdivides both input relations into sub-relations that are assigned to individual threads. There are two phases in this algorithm on the thread-level parallelism, parallelised partition and parallelised join. In the parallelised partition phase, all threads need to simultaneously perform partitioning, during the first level of partitioning. Then each thread can operate on a single partition without any explicit communication with other threads. In the parallelised join phase, a three step parallelisation scheme is used to reduce overheads and can efficiently utilise the memory bandwidth and computation cores.

2.4.3 Other Research Interests of In-memory Database

Data recovery is also a concern for the in-memory database systems, because the memory is normally volatile. Therefore, backups of memory must be maintained on hard disk devices or other stable storage devices. When a failure occurs, the in-memory database system must restore the data from its backup and then bring it up-to-date using the log. However, if the database is large, simply transferring the data from the disk may take a long time. Therefore, two solutions were proposed to reduce the recovery time after a failure has occurred. The first solution, proposed by Gruenwald and Eich [34], is to load blocks of the database "on demand" until all of the data has been loaded. However, it is only suitable for light-load systems, since heavy-load systems will receive millions of transactions in a second after the database has been recovered, and it is impossible to load such a large amount of data in a second from hard disk device. Another solution introduced by Patterson et al. [79], is to use multiple hard disk devices (e.g. the disk arrays) to backup the database and read the devices in parallel to reduce the loading time. But it will be very expensive to build such disk arrays to achieve effective performance.

Several in-memory DBMSs have been proposed or implemented. Baulier et al. [3] implemented an commercial in-memory database system, which is called DataBlitz Storage Manager. This system supports application access to the data in the memory directly via its configurable, multi-level API. To minimise the storage overhead, they use concurrent index management based on both hashing and tree structures. Meanwhile, they only store the direct point of data in the index instead of store the page information. Plattner [85] tested an OLTP (Online Transactional Processing) system and an OLAP (Online Analytical Processing) system by using an in-memory database system. The results show that the in-memory database outperformed the traditional database system even though the in-memory database did not use any indexing method. Meanwhile, the in-memory system can fully release the power of multi-core CPUs and parallel computing. Bining et al. [10] proposed a dictionary-based order-preserving string compression algorithm in the in-memory column system. In their work, they presented a new approach to indexing a dictionary of string values to leverage an order-preserving encoding scheme efficiently. For searching on the compressed data, they proposed a concrete leaf structure for the string values, which can be used by the indexes of a dictionary to efficiently encode and decode string values.

So far, most of the work in the in-memory database system field focuses on moving the traditional DBMS to an in-memory database system (i.e. for one dimensional data only). Yet, there is no any work focusing on the trajectory in-memory database system ,neither trajectory indexing for in-memory database approach nor trajectories query processing on the in-memory database system.

2.5 Column-oriented Data Structures

In contrast to a row-oriented data structure, a column-oriented data structure is used to store the data column by column. Therefore, a column-oriented data structure is suitable for analytic based queries or DBMSs as such query types always request aggregation operations on same column. Based on this, a column-oriented data structure can fetch the data from same column quickly and increase the performance of aggregation operations during analytic query processing. In this section, a set of column-oriented data structures are reviewed.

Boncz et al. [12] developed the modern in-memory database system, called MonetDB, along with the MIL query language to fully support the column-oriented store. Each column contains [oid, value] combinations and is stored in a Binary Association Table (BAT). A BAT is a 2-column table where the left column is called the head and the right column the tail, making up what is called a vertically fragmented data model. In this model, the execution primitives know only about the columns they operate on without having to know about the overall table layout. Therefore, the bandwidth requirements between CPU and memory are reduced based on this model. To further reduce bandwidth requirements, they use a lightweight compression to compress the data in each column.

Most current major DBMSs implement record-oriented storage systems, so the records (data) are stored continuously in the hard disk, which is called row-oriented architecture. In row-oriented architecture, a single disk write will suffice to push all of the fields of a single record out to disk. Therefore, it can achieve high writing performance as a write-optimised system. However, the row-oriented architecture is suitable for OLTP-style applications, however, Stonebraker et al. [104] indicate that system oriented towards ad-hoc querying of large amounts of data should be read-optimised, for example, customer relationship management systems and electronic library card catalogues. For such systems, the row-oriented store architecture could not be efficient as the query has to scan whole table and thus will consume a lot of bandwidth between CPU and hard disk. Hence, they proposed a new database system, called a C-Store, which improves on the column-oriented store architecture. To save the bandwidth, the C-Store physically stores a collection of columns, each sorted on some attribute, and then compresses these columns via aggressive compression techniques. There are two levels of this system, and the top level is called the Writeable Store component, which is architecture to support high performance inserts and updates. There is also a much larger component called the Read-optimised Store (RS), which is capable of supporting very large amounts of information. However, their system considers the column-oriented store in the disk based system only and focuses on optimized the bandwidth between CPU and hard disk and does not consider the in-memory approach or the optimisation of memory.

The column-oriented data structure is good for read-only tasks, since it does not need to scan the whole database to get the results. However, column-oriented architecture will lead to reduced performance for updating and deleting operations because such operations have to scan each column to find the data and perform the update or delete. Meanwhile, most column-oriented storage architectures use compression techniques to save the space, which makes updates/deletes computationally more expensive and complex since data needs to be de-compressed, updated/deleted and re-compressed. Therefore, Héman et al. [39] proposeed a new column-oriented data structure, called a Positional Delta Tree (PDT). A PDT is designed to make merging in of these updates fast by providing the tuple positions where differences have to be applied at update time. The PDT requires less I/O and CPU time based on positional merging than does value based merging. This work still looked at optimising the I/O between hard disk and CPU, even if they used the MonetDB, which is an in-memory system approach. To improve the queries performance, Lemke et al. [60] proposed a new algorithm to process queries such as scan and aggregation operations on the compressed column-oriented data structures. Ivanova et al. [46] studied a new architecture to provide an intermediate in the column-oriented structures, which uses a lightweight mechanism. Their approach can also improve the query performance significantly. Krueger et al. [57] presented a linear merge algorithm to utilise the power of parallel computing for fast updating in compressed in-memory column-oriented structures.

However, even if the trajectory query processing is performed on an OLAP system, these column-oriented data structures cannot support trajectory queries well. This is because such structures are designed for relational data only, and trajectory data is not relational data. Due to the underlying use of the relational model in these column-oriented data structures, analytical trajectory queries cannot be well-supported.

Chapter 3

SharkDB Design

3.1 Introduction

Driven by the 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 being generated at an 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 commercialisations[1], they were designed to support basic spatial types only such as points, lines and polygons. Trajectories, on the other hand, are not easy to fit into a relational table with a pre-defined schema since each tuple (i.e. trajectory) has a different number of attributes (i.e. time-stamped points). To fix this problem, the entire sequence of points can simply be treated as a single attribute and stored in one column. But this kind of storage will severely deteriorate query performance since it will make it almost impossible to utilise the spatio-temporal locality amongst trajectories. Having witnessed the limitations of existing spatial database systems, in the past decade researchers have dedicated significant efforts in proposing novel techniques for

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

trajectory data management. What lies in the core of these techniques is 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 a tree-based index) or cell (in a grid index).

The evolution of modern technology allows large amounts of memory to be installed in a computer system, thus providing potential opportunities to significantly improve the efficiency of managing large amounts of data by shifting more or even all data from disk-based storage into main memory. This has triggered widespread research interests in in-memory data management from both the database and data mining community, ranging from memory-based indexing techniques [89] to in-memory database systems [85]. 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.

In order to take advantage of a column-oriented data structure and compression techniques for storing and analysing trajectory data, two main challenges need to be solved. The first one is how to convert trajectory data into a column-oriented data structure, which not only supports varied length and multi-dimensional trajectory data, but can also perform efficient query processing in the main memory. The second challenge is how to deploy compression techniques on the column-oriented data structure, which can further support query processing on compressed data without sacrificing much performance. Therefore, in this chapter, the importance of calibrated trajectory data is first investigated by a preliminary study. Based on this investigation, a novel frame based structure is proposed to address the above challenges. The proposed frame based structure is a column-oriented structure, which is a read-optimized data structure for storing and processing massive amounts of trajectory data. This data structure combines the merits of high throughput of main memory and the benefits of a column-wise store for analytical tasks. Meanwhile, this design also supports effective trajectory data compression and query processing on the compressed trajectory data.

3.2 A Preliminary Study on Heterogeneous Trajectory Data

3.2.1 Motivation

A fundamental ingredient of such trajectory analysis tasks is the distance/similarity measure that can 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 with both spatial and temporal attributes, which needs to be considered for similarity measures. More than ten distance/similarity measures have been proposed in the literature, for example, Euclidean Distance (ED) [47], Dynamic Time Warping (DTW) [103], Piecewise Dynamic Time Warping (PDTW) [51], Distance based on Longest Common Subsequence (LCSS) [49], Edit Distance with Real Penalty (ERP) [19], Edit Distance on Real Sequence (EDR) [20]. 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. The details of six similarity measures have been described in the literature review chapter. We list the core definition of such similarity measures in the below. To define such similarity measure, assuming there are two trajectories $T = [p_1, p_2, ..., p_m]$ and $T' = [p'_1, p'_2, ..., p'_n]$, core definitions are:

Euclidean Distance Measure

$$Euclidean(p_i, p'_j) = d(p_i, p'_j) = \sqrt{(p_i.x - p'_j.x)^2 + (p_i.y - p'_j.y)^2} \tag{3.1}$$

DTW/PDTW The PDTW uses the Piecewise Aggregate Approximation(PAA) algorithm to simplify the trajectory, but the core definition of PDTW is same as DTW.

$$DTW(p_i, p'_j) = \begin{cases} 0, & if \quad m = n = 0 \\ \infty, & if \quad m = 0 \; or \; n = 0 \\ d(p_i, p'_j) + min(DTW(p_{i-1}, p_{j-1}), DTW(p_{i-1}, p_j), \\ DTW(p_i, p_{j-1})), \; otherwise \end{cases} \tag{3.2}$$

EDR The α is the match function of EDR. If the $d(p_i, p_j)$ is less than the given threshold, α is equal to 0, otherwise, α is equal to 1.

$$EDR(p_i, p'_j) = \begin{cases} n, & if \quad m = 0 \\ m, & if \quad n = 0 \\ min(EDR(p_{i-1}, p_{j-1}) + \alpha, EDR(p_{i-1}, p_j) + 1, \\ EDR(p_i, p_{j-1}) + 1), \; otherwise \end{cases} \tag{3.3}$$

ERP

$$ERP(p_i, p'_j) = \begin{cases} p_i, & if \quad n = 0 \\ p'_i, & if \quad m = 0 \\ min(ERP(p_{i-1}, p_{j-1}) + d(p_i, p'_j), ERP(p_{i-1}, p_j) + p_i), \\ ERP(p_i, p'_{j-1}) + p'_j), \; otherwise \end{cases} \tag{3.4}$$

LCSS The ϵ is the given threshold of LCSS.

$$LCSS(p_i, p'_j) = \begin{cases} 0, & if \quad m = 0 \; or \; n = 0 \\ LCSS(p_{i-1}, p_{j-1}) + 1, & if \quad |p_i - p'_j| \le \epsilon \\ max(LCSS(p_{i-1}, p_j), LCSS(p_i, p_{j-1})), & otherwise \end{cases} \tag{3.5}$$

Given the multitude of competitive techniques, a good understanding of the effectiveness of various similarity measures is important. Very often a newly introduced distance measure has claimed a particular advantage over some others by using an exemplified explanation. Also most of those works focused on evaluating the efficiency of their pruning and searching algorithms, while leaving the effectiveness study, i.e. how their proposed distance measure truly reflects the similarity between trajectories under different circumstances, inadequate or even completely omitted. In this light, we argue that there is a strong need for an empirical study on the effectiveness of trajectory similarity measures. More specifically, in this study we have implemented six widely used trajectory similarity measures (shown in the following list), and study their effectiveness in different circumstances using a common real world taxicab trajectory dataset.

- Euclidean Distance Measure
  - Euclidean Distance
- Dynamic Time Warping based Measures
  - DTW
  - PDTW
- Edit Distance based Measures
  - EDR
  - ERP
- Longest Common Subsequence based Measures
  - LCSS

3.2.2 Effectiveness Study

In this section, the trajectory dataset used in this study is first introduced. Then the types of transformations applied to the trajectories; and the experimental observations regarding the effectiveness of the compared similarity measures are also proposed.

Dataset

The dataset of Beijing taxi trajectories [140] is employed for this experimental study. This is a real-world trajectory dataset generated by 30,000 taxicabs in Beijing over a period of three months. The sampling rate of this data set is approximately 30 seconds, which means the time duration between consecutive sampling points is about 30 seconds. Since in this study we mainly focus on effectiveness rather than scalability, 1000 trajectories are randomly selected from the dataset, where each contains at least 100 sampling points.

Trajectory Transformations

Evaluating the effectiveness of different similarity measures objectively is a challenging task due to the lack of a widely recognised benchmark dataset, where the ground-truth distance between any pair of trajectories is known in advance. Therefore, while most previous works put emphasis on the scalability test for the similarity measures, none of them have conducted experiments on the effectiveness. This study tackles this problem from a novel aspect by having the following two observations. First, an identical motion history can be represented by different trajectories due to the variance in sampling time, sampling rate or possible noise. Second, in spite of different representations, they should still have high similarity based on any good similarity measure since they all actually refer to the same motion record.

Based upon this, the evaluation procedure works as follows. It firstly picks up a trajectory as the *original trajectory*. Then several types of transformations on the original trajectory are performed in a controlled way (by using parameters), resulting in a set of *transformed trajectories*. For each transformation, this study evaluates the distance between the original and transformed trajectories and tunes the parameter to see how the distance between the trajectories is affected. The rationale behind this is that, with a reasonable similarity measure, the trajectory with a lower degree of transformation should have a higher similarity to the original trajectory, and vice versa.

Three types of transformation functions are devised, namely re-sampling a trajectory, shifting trajectory points, and adding noise. These transformations are controlled by two parameters, *rate* and *distance*. The parameter *rate* is used to specify the percentage of the trajectory points that will be transformed; for instance, $rate = 0.1$ means that 10% of the trajectory points are to be transformed by the transformation function, and $distance = 0.0001$ means that the trajectory points are to be shifted around 11 metres by the transformation function. The parameter *distance* is a threshold of how far a trajectory point might be shifted in relation to the original point. Table 3.1 summarises all the transformation functions and their parameters.

Re-sampling trajectory. There are two ways to re-sample a trajectory, i.e. increasing sampling rate and decreasing sampling rate. To increase the sampling rate, the $rate$ extra points will be randomly added to the original trajectory. Analogously, to decrease the sampling rate, this experiment randomly remove $rate$ points from the original trajectory. Fig. 3.1 and Fig. 3.2 exemplify

TABLE 3.1: Types of Trajectory Transformations

Transformation Type	**Operation**	**Adjustable Parameters**
Re-sampling	Increase sampling rate (add points)	$rate$
	Decrease sampling rate (remove points)	$rate$
Point shift	Random shift	$rate, distance$
	Synchronised shift	$rate, distance$
Noise	Add noise	$rate, distance$

those two opposite transformations.

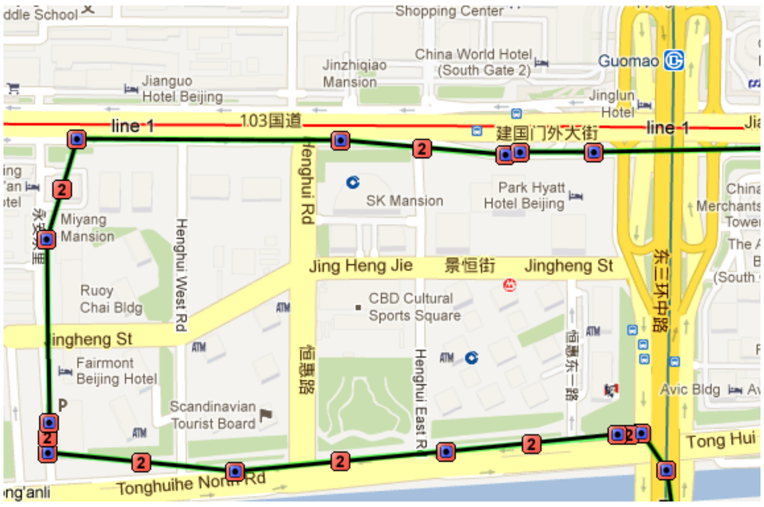


FIGURE 3.1: Increase Sampling Rate Transformation Function

Point shift. Unlike the re-sampling transformation, a point shift does not change the number of trajectory points. Instead, it changes the locations of them. To do so, this study randomly selects the $rate$ of the trajectory points and shifts them by *distance*. There are two ways to shift the points, i.e. random shift and synchronised shift. A random shift will change the position of each selected point arbitrarily without considering the other shifted points, while synchronized shift will translate all the selected points in the same way (same offset and direction). Additionally, a point shift transformation would not change the shape of the original trajectories. Fig. 3.3 and Fig. 3.4 illustrate these two shift transformations.

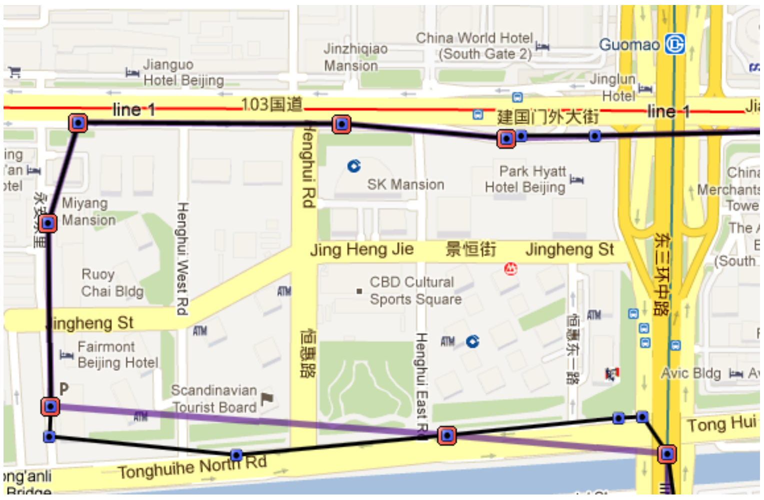


FIGURE 3.2: Decrease Sampling Rate Transformation Function

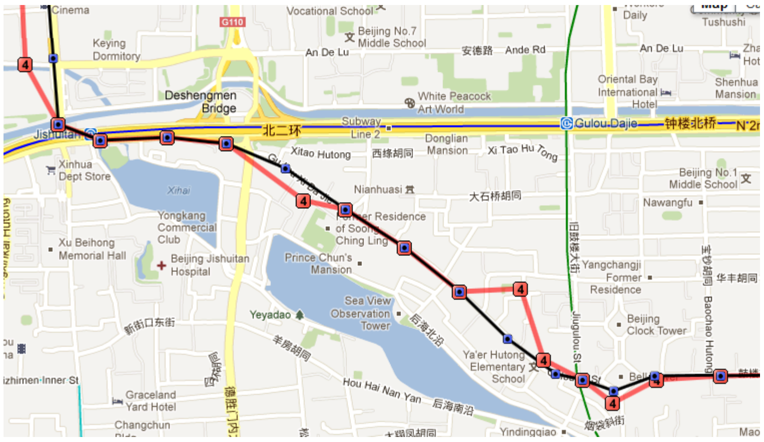


FIGURE 3.3: Random Shift Transformation Function

Adding noise. The last transformation function is to add $rate$ noise/outliers to the original trajectory. The gap between the noisy points and the original trajectory is controlled by the parameter $distance$. An example is used to demonstrate this transformation in Fig. 3.5.

Experimental Observations

In this section, the set of transformations is applied to the original trajectories and the distance/similarity between the original and transformed trajectories are computed based on each similarity measure. Specifically, for each similarity measure, two sets of experiments are conducted. First, the parameter $distance$ is fixed asa constant ($distance = 0.0015$), and then the parameter $rate$ is varied

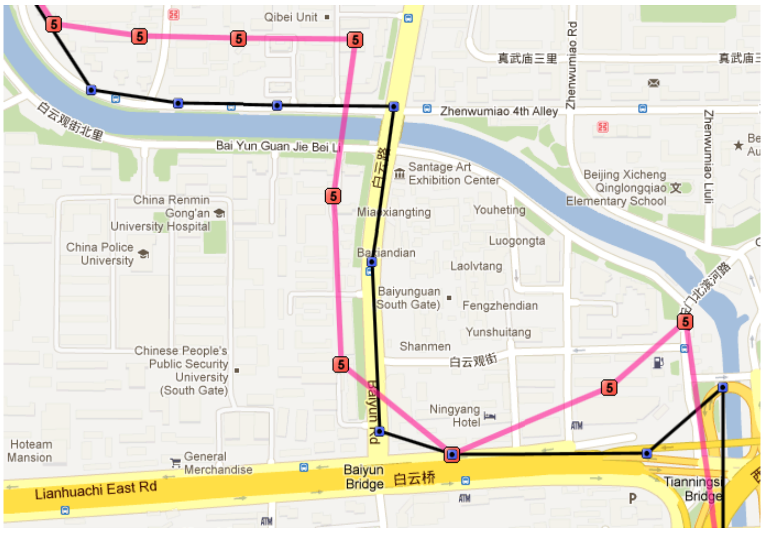


FIGURE 3.4: Synchronized Shift Transformation Function

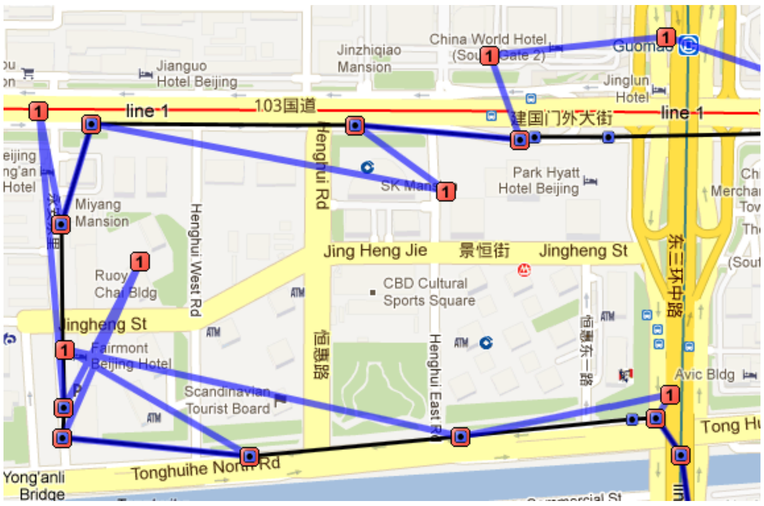


FIGURE 3.5: Add Noise Transformation Function

from 0.1 to 0.6 with a step of 0.1. However, EDR and LCSS measures use another threshold ε to determine the matched pairs of points. The relationship between $distance$ and ε will heavily affect the results. Therefore, two sets of experiments for LCSS and EDR are conducted, i.e. with $\varepsilon = 0.002$ being greater than $distance$, and $\varepsilon = 0.0004$ being less than $distance$. It was found that, adding $10\% - 60\%$ noise points into the trajectory was too much and could change the shape of the original trajectory, hence the transformation rate was reduced by reducing the adding noise function from 0.1 to 0.06 with a step of 0.01, which is one-tenth of the previous parameter $rate$.

Second, the parameter $rate$ is fixed as constant ($rate = 0.3$)[2], and the value of parameter $distance$ was changed from 0.0005 to 0.004 (Euclidean distance in spatial space) with the step of 0.0005. It only changes the transformation distance for random shift, synchronised shift and adding noise as these are the only transformations are affected by this parameter.

For all the similarity measures except LCSS and EDR, the distance between the original and transformed trajectories was reported, with a greater value indicates a lower similarity.

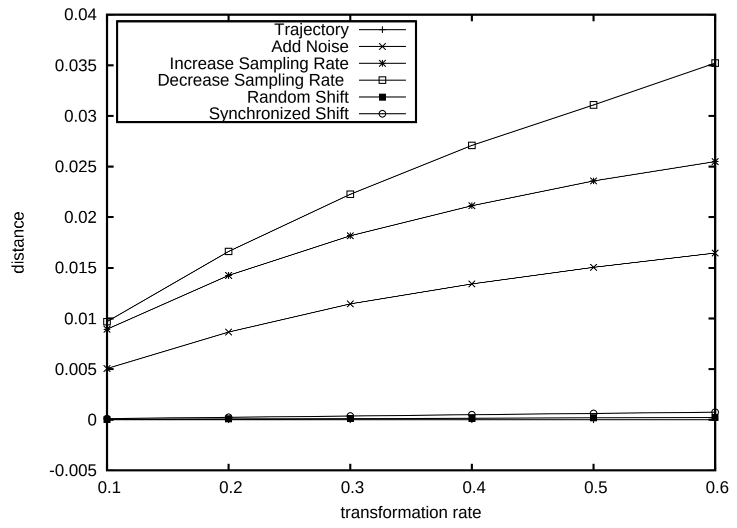


FIGURE 3.6: Result of Euclidean Distance with different transformation rate

Euclidean Distance Measure. The results of Euclidean distance with varying transformation $rate$ are shown in Fig. 3.6[1]. We can see that, the distance between the original and transformed trajectories with re-sampling and noise increases quickly as the transformation rate rises. This implies that the Euclidean distance is sensitive to sampling rate or noise. On the other hand, shifting sampling points within a certain range has little influence on the distance.

The results of Euclidean distance with different transformation distances is illustrated in Fig. 3.7. As expected, the distance between the original and transformed trajectories with point shift gradually increases as the transformation distance increases. But adding noise will make the transformed trajectory completely dissimilar to the original one, which again indicates that Euclidean distance is sensitive to outliers.

[2]Transformation rate of add noise function is set to 0.03

[1]Transformation rate of add noise function is one-tenth of x-axis's value, the follow figures use same setting for transformation rate of add noise. The y-axis distance value is defined by definition of trajectory similarity measures

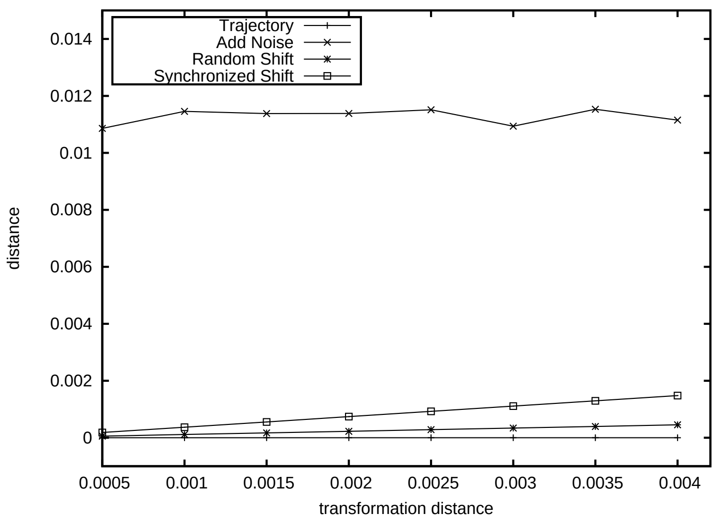


FIGURE 3.7: Result of Euclidean distance with different transformation distances

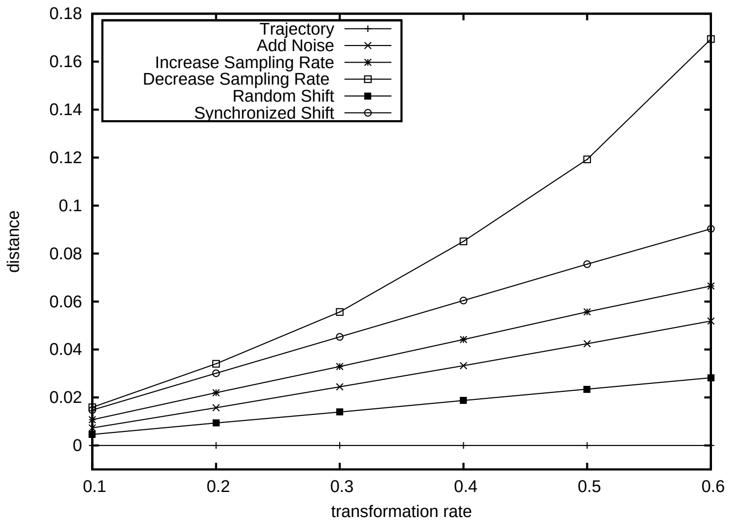


FIGURE 3.8: Result of DTW with different transformation rate

Dynamic Time Warping based Measures. The performance of DTW with a changing transformation $rate$ is shown in Fig. 3.8. It can be observed that DTW achieves a relatively good performance with a low transformation rate (i.e. $rate < 20\%$), and DTW is more robust to the random shift transformation. In addition, it is more sensitive to a decreasing sampling rate than an increasing sampling rate. Also DTW may not be a good choice when the trajectory data is contaminated by noise.

This study then evaluates the DTW distance with different transformation distances. As shown

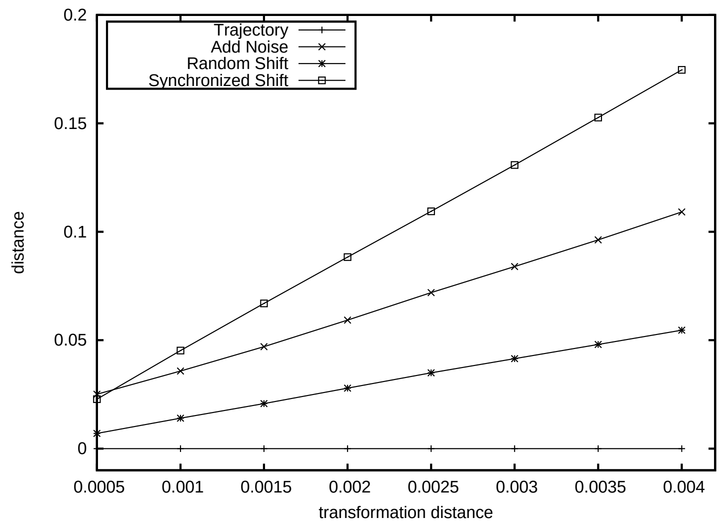


FIGURE 3.9: Result of DTW with different transformation distances

in Fig. 3.9, DTW is more sensitive to transformation distance than transformation rate as all distances increase quickly when the transformation distance grows larger. DTW may not handle dramatic sampling points shift well especially the synchronised shift.

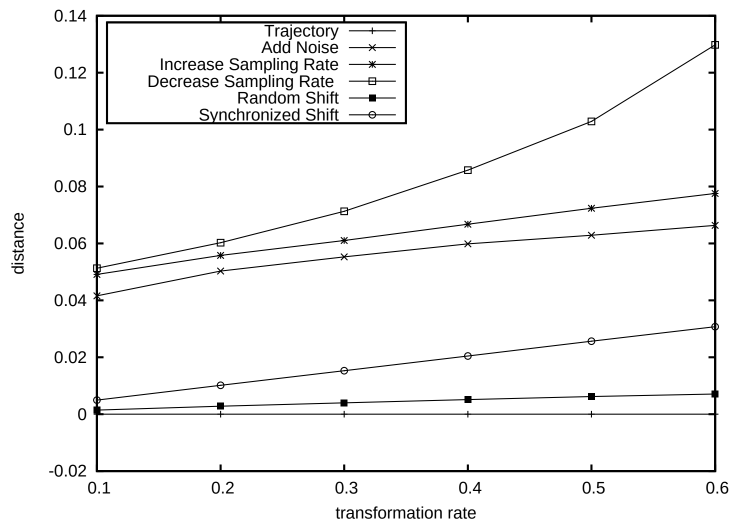


FIGURE 3.10: Result of PDTW with different transformation rate

Fig. 3.10 shows the results of PDTW with a changing transformation rate, which are similar to those of DTW, since PDTW is a variant of DTW. However, after applying the PAA method, the effectiveness of PDTW is better than DTW. With the same scale and unit as in the DTW experiments, we can see the distances reported by PDTW are less than those of DTW for all transformation rates,

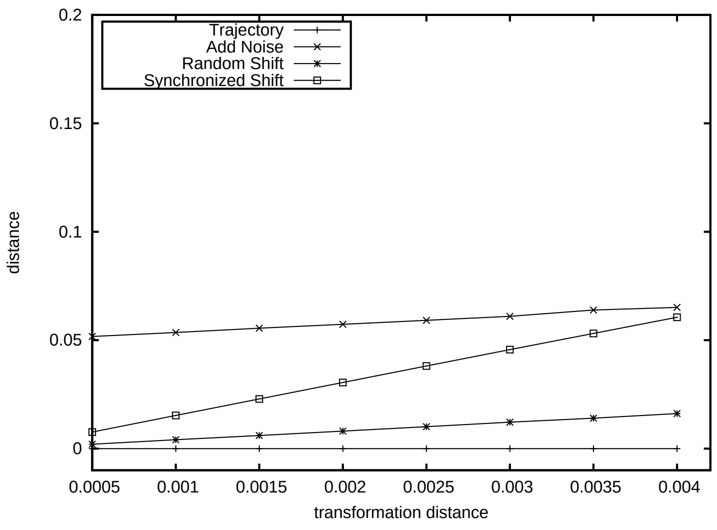


FIGURE 3.11: Result of PDTW with different transformation distances

especially for increasing sampling rate, adding noise and synchronised shifting function.

The performance of PDTW with different transformation distances is shown in Fig. 3.11. Unlike DTW, PDTW is not sensitive to length of distance of transformation. As a result, PDTW may work well in measuring the similarity of trajectories with a large number of inaccurate points (i.e points with large deviations from its true location).

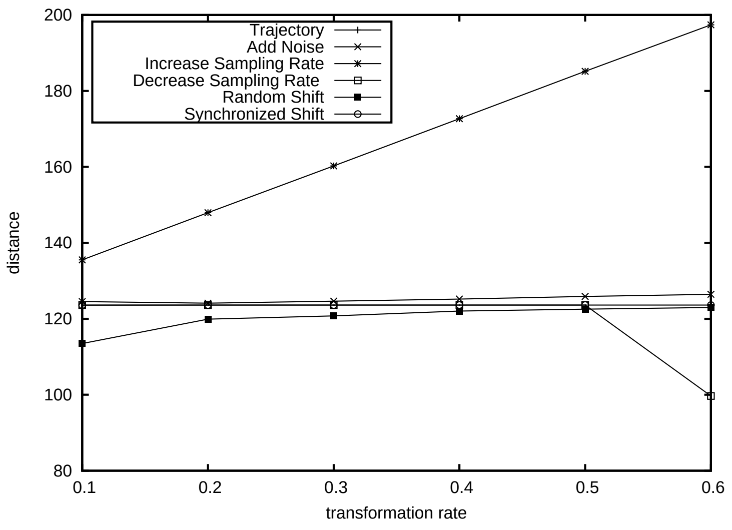


FIGURE 3.12: Result of EDR with a changing transformation rate and ε less than distance of transformation

Edit Distance based Measures. The effectiveness of EDR with a changing transformation rate

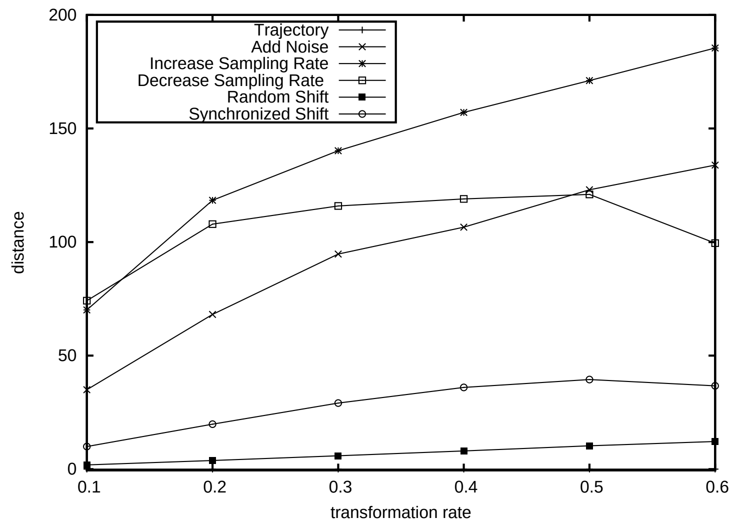


FIGURE 3.13: Result of EDR with different transformation rate and ε larger than distance of transformation

is firstly evaluated. In Fig. 3.13, the distances between the original and transformed trajectories are all large, since the transformation distance of the sample points is larger than its threshold ε. In this case, most shifted points will have no matched point in the original trajectory, hence increasing the EDR distance.

Another set of experiments is also conducted in which the transformation distance is restricted to be smaller than ε, the result of which is shown in Fig. 3.13. Based on the result, we can see that EDR is very sensitive to altering sampling rate or adding noise. However, EDR still serves as a good distance measure for handling sampling point shift.

Second, the experiment results of ERP distance with changing transformation rates are illustrated in Fig. 3.14. From the result we observe that ERP is robust to sample points shifting. Even with a very high transformation rate, ERP still achieves good performance in capturing the similarity between trajectories with sample point shifting. ERP can also handle the trajectories with a small amount of noise. However, it is sensitive to the changes in sampling rate of trajectories.

Fig. 3.15 shows the experiment result of ERP distance with different transformation distances. Based on this distance measure, the transformed trajectory with random shift is very similar to the original one, which means that ERP distance is robust to random shift. Nevertheless, ERP is quite sensitive to noisy data since adding noise to the original trajectory will result in a large distance value.

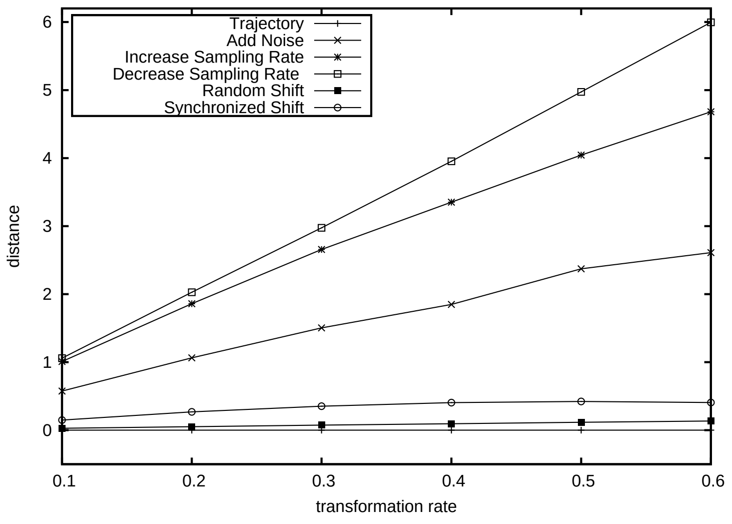


FIGURE 3.14: Result of ERP with a changing transformation rate

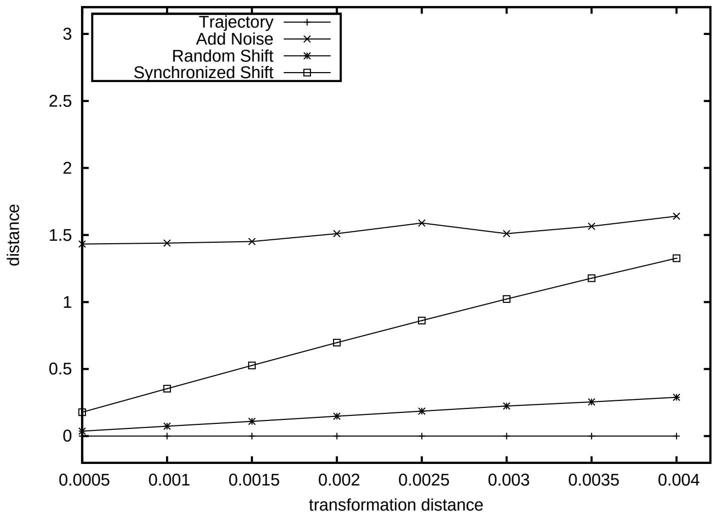


FIGURE 3.15: Result of ERP with a changing transformation distances

Longest Common Subsequence Measure. Finally, this study evaluates the distance based on LCSS for different transformations, as shown in Fig. 3.16. Due to the different definition of LCSS, it uses normalised similarity as the output, which is instead of distance. Interestingly, LCSS has perfect performance for the transformation with increasing sampling rate, but bad performance for transformation with decreasing sample rate. This is due to the fact that LCSS is calculated based on common subsequences shared between trajectories, which is not affected by increasing the sampling rate. For instance, given a trajectory sequence $Tr = [(1, 1, t_1), (2, 2, t_2), ..., (10, 10, t_{10})]$

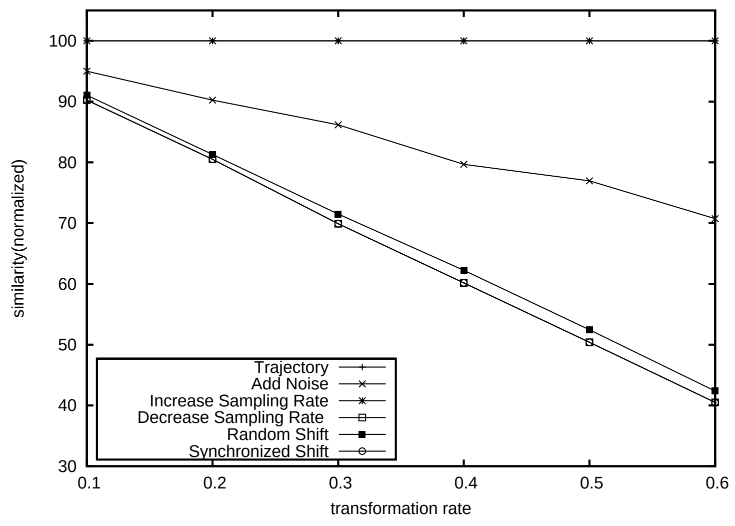


FIGURE 3.16: Result of LCSS with a changing transformation rate and ε less than distance of transformation

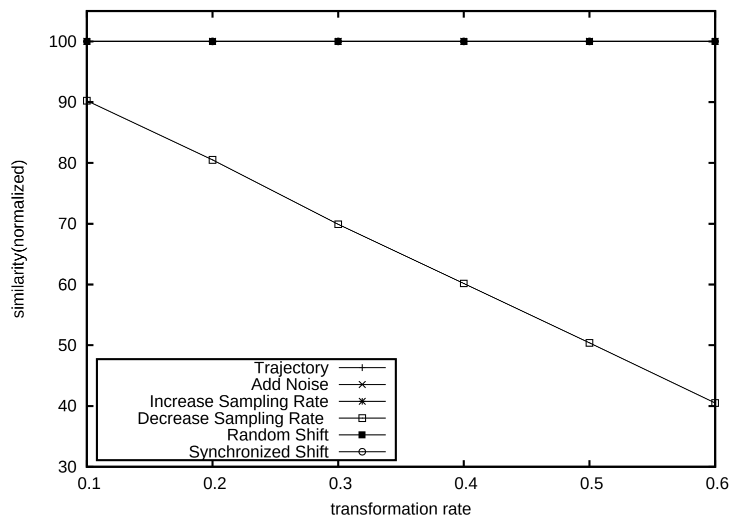


FIGURE 3.17: Result of LCSS a changing transformation rate and ε larger than distance of transformation

with length $l = 10$. By increasing the sampling rate, we have a transformed trajectory $Tr_I = [(1, 1, t_1), (1.5, 1.5, t'_1), (2, 2, t_2), (2.5, 2.5, t'_2), ..., (10, 10, t_{10})]$ with length $l = 20$; by decreasing the sampling rate, we have a transformed trajectory $Tr_D = [(1, 1, t_1), (3, 3, t_3), ..., (9, 9, t_9)]$ with length $l = 5$. Clearly, the length of common sequences that is reported by LCSS between Tr and Tr_I is larger than that between Tr and Tr_D. In addition, LCSS has a good performance to process noisy trajectory data, but might not be suitable for measuring trajectories that contain shifted sampling points.

An extra experiment for LCSS is performed by setting the threshold ε larger than the transformation distance, the result of which is shown in Fig. 3.17. As expected, the transformed trajectories are all treated as identical to the original one, except the one with the decreasing sample rate.

TABLE 3.2: Comparative Results of Trajectory Similarity Measures

	Euclidean Distance	**DTW**	**PDTW**	**EDR**	**ERP**	**LCSS**
Add noise	Sensitive	Sensitive	Fair	Sensitive	Sensitive	Robust
Increasing sampling rate	Sensitive	Fair	Fair	Sensitive	Sensitive	Robust
Decrease sampling rate	Sensitive	Sensitive	Sensitive	Fair	Fair	Sensitive
Random shift	Robust	Robust	Robust	Robust	Robust	Fair
Synchronised shift	Robust	Sensitive	Robust	Robust	Robust	Fair

In conclusion, there is no trajectory similarity measure that can beat all the others in every circumstance. Table 3.2 summarised the results for each trajectory similarity measures, which are compared based on transformation functions (first column). There are three levels, "Sensitive" , "Fair" and "Robust", which are illustrated the results. In general, Euclidean distance is a good choice when the trajectory data have similar sampling rates and high quality (small point shift), due to its simplicity of implementation and low computation complexity. PDTW is more robust to most transformations than DTW since it adopts the piece-wise aggregation to the raw trajectory before the distance computation. Edit distance based measures (EDR and ERP) achieve good effectiveness with point shift transformations, but are sensitive to altering sampling rate and outliers. In contrast, LCSS is almost immune to increasing sampling rates and noise but is sensitive to point shift. It seems that no similarity measure works well for a decreasing sampling rate, implying that processing low-sampling-rate trajectories can be a challenging problem [64][137]. Moreover, this preliminary study illustrates that the heterogeneous trajectory data can affect the performance of such fundamental trajectory queries. Hence, it is indicated that the trajectory data need to be synchronized before the query processing to avoid this issue. In the next section, a new trajectory data structure, which is called I/P frame based trajectory data structure, is proposed. This trajectory data structure can synchronize the trajectory data in nature and can improve the performance of trajectory similarity search. A comprehensive study of I/P frame based trajectory data structure is presented in the next section.

3.3 An I/P Frame based Trajectory Data Structure

This section details the three data structures as well as their encoding algorithms used to store the raw trajectory data into the in-memory column-oriented data structure.

3.3.1 Frame-based Storage

In contrast to basic operations (i.e. SELECT or DELETE), most analytic tasks such as window query and nearest neighbour query only need to touch a few trajectories to get the required answer. However, the row-oriented data structure is not good for this task, since it is hard to avoid a whole table scan during trajectory query processing, if these data are not indexed. On the other hand, the column-oriented data structure is known to have better performance in analytic task compared with a row-oriented data structure [85]. To get this advantage, this thesis proposes a novel frame based column-oriented data structure to store trajectory data in the main memory. Thus, a sequence of frames is created directly from trajectory data directly. A particular time interval is assigned to each single frame, then each trajectory sample point is allocated to the related frame based on its recorded time. Therefore, a frame contains all of the trajectory sample points in the whole dataset which are recorded at the time interval of the frame. And that time interval is called the frame rate. For instance, as Fig. 3.18 shows, if the time interval is set to one minute (i.e. the frame rate is 60 seconds), the time period from 9:01 to 9:05 is split into four frames. Hence, the sample points in the trajectory $T_1 = p'_1, p'_2, p'_3, p'_4$ and $T_2 = p_1, p_2, p_3, p_4$ will be aligned to the related frame, in this example, p'_1, p_1 and p'_2, p_2 are assigned into $Frame1$ and $Frame2$ respectively.

To keep each frame simple and tidy, each frame contains no more than one point for each trajectory. To make this frame structure continuous in temporal space, each frame must be strictly synchronised by the same time period (i.e. with the same frame rate). Normally, the frame rate will be the same as the sampling rate of the trajectory. For example, if the sampling rate of the trajectory

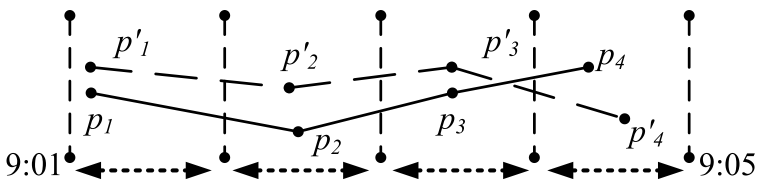


FIGURE 3.18: Trajectory snapshot

dataset is 60 seconds. Then the frame rate will be set to 60 seconds per frame as Table 3.3 shows. However, a trajectory in a raw trajectory dataset may have a different sampling rate compared with other trajectories due to an unstable GPS signal. Therefore, this situation can make the whole trajectory dataset become heterogeneous, which may affect the frame structure. To avoid this issue, if there is more than one sample point, which belongs to the same trajectory, they are aligned to the same frame. Synchronous Euclidean Distance(SED) [67] is calculated for each sample point. SED is a kind of distance measure that considers both spatial and temporal information. For instance, as Fig. 3.19 shows, the SED of P ($SED(P)$) is calculated by $D(P_{(x,y,t)}, P'_{(x',y',t)})$, where $x' = A.x + \frac{A.t - P.t}{B.t - P.t}(A.x - B.x)$, $y' = A.y + \frac{A.t - P.t}{B.t - P.t}(A.y - B.y)$. Therefore, the sample point with largest SED is kept, whereas the rest of the sample points in that frame are removed since such a point contains more information than other points that have smaller SED. In the same time, if there is no sample point of a trajectory in a frame, which is between the trajectory's start frame and end frame, it uses a line interpolation method to add a new sample point of the trajectory in that frame. A set of accuracy experiments is conducted in Chapter 5 to show the influence of heterogeneous trajectories.

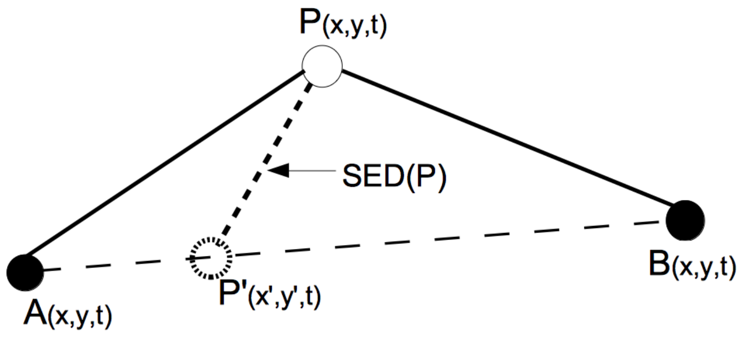


FIGURE 3.19: Example of SED

After that, converting each frame into a column-oriented data structure become straightforward.

TABLE 3.3: 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)	(2p_4)

Basically, it only needs 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 3.3 shows. In addition, each sample point (i.e. the object) in the column contains its trajectory ID, longitude and latitude. After aligning, the timestamp information of a sample point is no longer needed any more, since the temporal information can be recovered by the column ID. The advantage is that this frame data structure is a natural column-oriented data structure and synchronised by time, hence, storing this data in the column-oriented structure is simple.

3.3.2 I/P Frame-based Storage

For in-memory databases, it is good to utilise compression techniques to reduce the size of the data and improve query performance, since the compressed data can reduce the footprint of data in the memory, which can in turn reduce the searching time in memory. The approach considered to compress the trajectory data was the delta encoding technique, which keeps information for the first point and uses the δ value to record the information for the rest of the points. Therefore, the delta encoding will not change the sampling rate of trajectories and it allows the storage system to use less space to store the trajectory data. However, a problem arises when using delta encoding. For each column, the type of object (point) must be the same, which means that the objects in a single column can only be the original points or the delta encoded points, but not a mixture of both. However, the start time of trajectory and the length of trajectory is arbitrary, so delta encoding cannot be deployed directly. Therefore, a novel I/P frame based data structure is proposed to reduce the memory consumption without encountering this problem.

Building an I/P frame data structure includes a grouping process and an encoding process. First of all, the whole sequence of frames is split into small groups, called frame groups, and each

TABLE 3.4: Example of I/P-frame Structure

Frame 1	Frame 2	Frame 3	Frame 4
(1,p)	(1,Δp)	(1,Δp)	(1,Δp)
(2,p)	(2,Δp)	(3,Δp)	(4,Δp)
(3,p)	(3,Δp)	(4,Δp)	
(4,p)	(4,Δp)		

frame group contains n continuing frames. Based on the example in Table 3.3, if n is set to 4, then $Frame1$ to $Frame4$ is allocated in a group. After the encoding phase, for each group, we keep the first frame of raw information as an I-frame, and the subsequent frames, called P-frames, use delta encoding for compression. In the example is shown in Table 3.4, $Frame1$ is kept as I-frame IF_1. The rest of the frames $Frame2$ to $Frame4$ are P-frames P_1 to P_3, then the sample points are encoded in P_1 by calculating the difference between IF_1 and its related sample point in P_1 as a P-frame point $P_1.PF_1$. Then, we continue to encode the sample points in P_2 and P_3 as P-frame points $P_2.PF_1$ and $P_3.PF_1$ respectively. In general, the coordinate changes between I-Frame point and each P-Frame point are very small, therefore, we can use less bits to record such offset value for each P-Frame point. Hence, in the example shown in Table 3.4, the I-Frame point costs 16 bits (8 bits for longitude and 8 bits for latitude); and for each P-Frame point can only cost 4 bits for encoding the coordinate shifts. It is easy to see that each P-Frame point saves 12 bits compared with I-Frame point. Finally, we get one I-frame column and three encoded P-frames columns as shown in Table 3.4.

However, this I/P frame encoding algorithm still has some drawbacks. To reconstruct a segment of a trajectory that is contained in a frame group, in the current solution has to scan all of columns in that group must be scanned, which includes both I-frame and P-frames; and this process is very expensive, since it has to scan a lot of memory. Therefore, the query processing performance can be reduced; and the process does not fully utilise the power of a column-oriented data structure.

3.3.3 Cache-aware Frame-based Storage

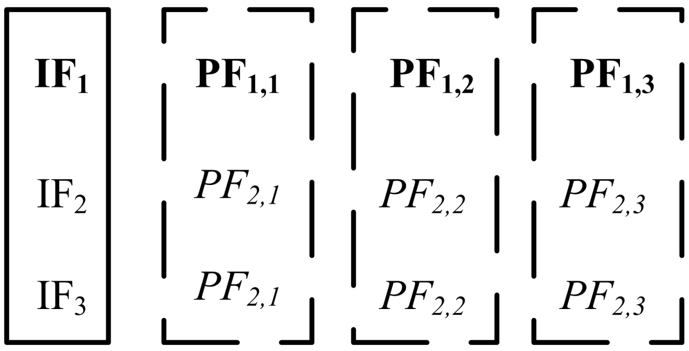
To increase the speed of trajectory reconstruction, the number of sequential scans of columns should be minimised during query processing. The idea of avoiding a sequential scan of all columns in a frame group is to reduce the number of scans in a P-frame, since the P-frames are closely related to I-frames. Therefore we fix the length of both the I-frame column and P-frame columns in a frame group, which means that the length of each P-frame column is equal to the length of the I-frame column. As a result, for the same segment of a trajectory in a frame group, each P-frame point of that trajectory shares the same index as the point in the I-frame. Meanwhile, if a trajectory segment has not fully filled a frame group such as a trajectory segment with one I-frame point and two P-frame points in a $n = 4$ frame group, we use a Nil code as a place-holder in the rest of P-frames in that group. Hence, to reconstruct a trajectory segment, we can scan the I-frame point in the I-frame only 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 the I-frame point. For example, as Table 3.5 shows, to reconstruct trajectory T_4, first the I-frame is scanned to get its I-frame point, then the related P-frame points can be accessed directly in memory by adding an offset from the beginning address of each P-frame column. This can improve the speed significantly during the trajectory reconstruction processing.

Moreover, in modern computer system architectures, accessing data from memory to CPU is via a hierarchical memory structure. Typically, a CPU has built-in cache memories, which is connected to the system main memory directly. On the other hand, the reading action from memory to CPU is parametrised by CPU cache line size (typically 64 bytes), which is the basic transferring unit. For a reading action, the CPU will read 64 bytes data from memory whether or not this data will be fully used or not. Therefore, to increase the performance, ideally the CPU cache line should be filled up 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 at the same time we cannot break the column-oriented data structure.

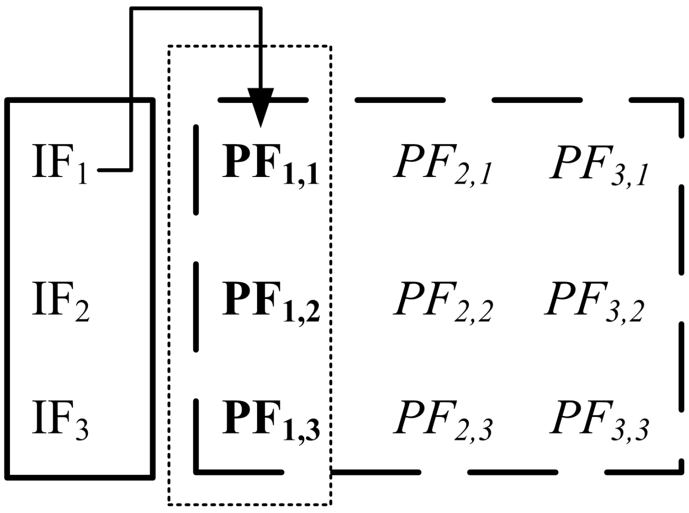
Hence, a hybrid data structure is used to store the P-frames in a frame group as they are highly

TABLE 3.5: 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,Δp)	(1,Δp)	(1,Δp)
(2,p)	(2,Δp)	Nil	Nil
(3,p)	(3,Δp)	(3,Δp)	Nil
(4,p)	(4,Δp)	(4,Δp)	(4,Δp)



(a) Non-cache Optimized Structure



(b) Cache Optimized Structure

FIGURE 3.20: Example of Cache-aware I/P-frame Structure

related. The hybrid data structure is based on a two dimensional array, which is created as an $array[x][y]$, where x is equal to the number of I-frame points in this frame group, and y is equal to the number of P-frame columns in this frame group. So, the value of x in the array is the index of the I-frame points; the value of y indicates the column number of the 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 Fig. 3.20, and 1 for first I-frame point and 2 for second P-frame column. As a result, the CPU can fetch an I-frame point with its related P-frame points and fill a cache line block via a single memory access. When the CPU tries to access the next P-frame point, it is already in the CPU cache, and therefore CPU can retrieve it directly without needing to search the memory again. Consequently, it will increase the performance of query processing. For example, as Fig. 3.20 shows, assuming a memory access reads two points vertically from the P-frame array, this will cost two memory seeking requests to decode a trajectory segment in a frame group stored in a system with a non-cache optimisation data structure, since the other points in the cache line are not related to this segment and are therefore not usable. On the other hand, only one memory access would be required to decode this trajectory segment when using a cache optimised data structure, since the related four points can be read from CPU cache directly.

3.4 Database Maintenance

To store and manage a new trajectory dataset, a new database needs to be created in SharkDB. The key issue in creating a new database is that the number of columns are unpredictable, since the trajectory data is continuously being collected from its source, which means that the total length of whole trajectory dataset is unknown. In this section, the process to create a new database in SharkDB based on the frame data structure is first introduced to address this issue. Then an appending method is presented to append new sample points to existing trajectories.

Creating the Database The frame data structure requires the sample rate of all trajectories (synchronised sample rate of trajectory) to be the same to avoid the previously noted accuracy problem, which means that the trajectory dataset needs to be synchronised before being imported into the database. However, it is impossible to guarantee this from any raw trajectory dataset due to unstable GPS signals. Actually, this problem belongs to the data cleaning area, which is not the research direction of this thesis. Hence, in this study, all trajectories are first synchronised to ensure the sampling rate of the imported trajectory data is uniform.

After the trajectory dataset is synchronised, it is time to start creating a new database. As discussed before, the trajectories will be split and encoded as frame group structure. Based on this, there are three parameters, required to set up the database. The first one is the earliest time of the whole trajectory, and the second one is the time interval of each frame, which is equal to the sampling rate of this dataset. The last parameter is the number of frames in single frame group, which is a key parameter that can affect the performance of the database significantly. More details about selecting the number of frames per frame group are discussed in the experiments chapter. After the database is created, the trajectory data can be imported into the database by the two-phase algorithm. As shown in Fig. 3.21, two components are implemented for data importing. First of all, trajectories are converted to a column based structure and aligned to the frame data structure in the aligning component. After this, in the encoding component, the frames are split into small groups and encoded to the I/P frame structure format.

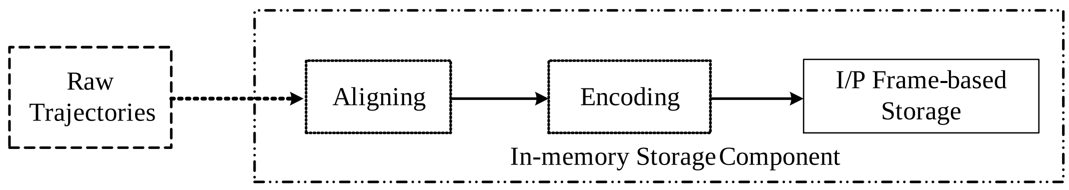


FIGURE 3.21: Creating Database

Appending the Trajectory In contrast to a transaction database, appending to an existing trajectory is more important than updating an existing trajectory as a trajectory records the historical information of moving object, which means that such data should not change once it has been generated. To expand an existing trajectory T_e by given a trajectory ID TID and a new trajectory T_{new}, the database system only needs to select the last frame group FG_{last} of T_e and decode it to get a trajectory segment. Then, we connect this trajectory segment with T_{new}. After this, T_{new} is allocated and encoded as the new frame groups with related frame group column ID. Finally, the last frame group FG_{last} is removed and these new frame groups are inserted into database.

3.5 Summary

This chapter first presents a preliminary study on heterogeneous trajectory data, which shows that heterogeneous trajectory data can harm the trajectory query processing. Then an I/P frame structure, a new in-memory column-oriented storage data structure for storing and querying trajectory data, is proposed. Finally, an I/P frame based column-oriented data structure with CPU-cache optimisation is implemented to provide an efficient storage system.

Chapter 4

Query Processing in SharkDB

4.1 Introduction

The trajectory of a moving object is typically modelled as a time-stamped sequence of consecutive locations in a multidimensional (generally two or three dimensional) space. Large amounts of trajectory data are currently generated and managed in many application domains such as environmental information systems, meteorology, wireless technology, video tracking, and video motion capture [139, 100, 136, 120, 24, 133, 23]. In general, the most basic problem of query processing in SharkDB is to support frequently used database operations, which are used to import a new trajectory into the system or download a specific trajectory from the system.

In addition, such types of data have provided unprecedented information to help us understand the behaviour of moving objects, and resulted in growing interest in the data analysis of such data. In analytical tasks, two fundamental trajectory queries are often used: the window query and the kNN query. Typical examples include collecting GPS location histories of taxicabs for safety and management purposes, tracking animals for their migration patterns, urban planning by detecting hot areas from human trajectories, and using traffic trajectories for road optimisation.

Another important problem in such analytical tasks is designing techniques for identifying trajectories that are similar. Such techniques can be used by many data analysis tasks that include trajectory clustering, classification, and mining, which have a broad range of real applications. For instance, in many sports such as football and tennis, it is very useful for sports researchers to figure out the movement patterns of top players by finding similar trajectories in the motion of objects (players, balls). By analysing similar trajectories of animals, it is possible to determine their migration patterns. In a city traffic monitoring system, it is helpful to locate popular routes by comparing similarities between the trajectories of vehicles.

In this chapter, to address the above problems, a set of queries/operations are proposed in SharkDB. The proposed queries/operations are divided into three categories that include basic operations, advanced operations and analytic operations. The details of these categories are listed below. Then, for each category, several algorithms are implemented in order to solve them efficiently.

Basic Operations This category represents classic and frequently used database operations as the basic operations. The basic operations normally contain SELECT, DELETE, INSERT and APPEND as classic trajectory operations for the transaction databases, which consider each trajectory as a single record. More specifically, the INSERT operation is used to add new trajectory data with a new trajectory ID into the database. The DELETE operation is used to delete an existing trajectory (by trajectory ID) from SharkDB. The SELECT operation is used to retrieve an existing trajectory by trajectory ID, which can be done by decoding its frame points in I/P frame structure to the original format. As both spatial and temporal information are recorded in trajectory data permanently, trajectory data should not be changed. Therefore, in SharkDB, trajectory data is considered to be a kind of historical data, which means that the UPDATE operation (i.e. update the information of an existing trajectory) is not supported. In addition, spatio-temporal information can be continuously collected from the same moving object. Thus, in SharkDB, an APPEND operation is proposed instead of the UPDATE operation. The APPEND operation is used to add new information (i.e. sample points) into an existing trajectory. Therefore, the new spatio-temporal information from same moving object can be appended to the existing trajectory by this moving object identifier (i.e. trajectory ID). The implementation details of these operations are discussed in Section 4.2.

Advanced Operations This category represents a set of advanced operations that are typically applied to trajectory data and include the window query and the kNN query. A window query will find all trajectories in the data set that are active during a given period and that pass through a given region. A kNN query will find the top-k trajectories in the trajectory data set that are close to a given point and are active during a given period of time. These queries can be extended to many real applications such as urban computing, and traffic monitoring. Thus the queries that are included in this category are the foundation of trajectory analytic tasks. Algorithms that can process these operations efficiently are proposed in Section 4.3.

Analytic Operation This category represents an important operation (i.e., trajectory similarity search) that can be applied by applications to trajectory data. Given a query trajectory, the trajectory similarity search is used to find a trajectory or a set of trajectories that are similar to the query trajectory. The similarity between two trajectories is calculated via similarity measures. In contrast to the advanced operations, the operations in this category can be used directly in many applications such as trajectory clustering and trajectory pattern mining. The algorithms for trajectory similarity searches in SharkDB are presented in Section 4.4.

4.2 Basic Operations

In this section, a naive approach for basic operations is proposed firstly in Section 4.2.1. Then a multi-thread based approach is presented to improve the performance of the basic operations.

4.2.1 Single Thread based Approach

The naive way to process SELECT and DELETE by trajectory ID in I/P frame structure is to search the auxiliary table first to get the column number of the start frame group column ID and the end frame group column ID by given trajectory ID. Then the system scans each selected frame column and finds the related frame group. For a SELECT operation, this frame group will be decoded to the original trajectory sample point, and for a DELETE operation, this frame group can be removed instantly. Since the frame groups are stored in an array structure, the rest frame groups that are followed by the deleted frame group need to be packed (i.e. to be connected with head part of array) to avoid memory fragments. Although such operation may encounter some overhead since the memory copy takes some time, the performance of DELETE operation is still acceptable. This is because the memory copy is much faster than hard disk. The performance evaluation is described in Chapter 5.

To insert a new trajectory into SharkDB, first the raw trajectory is encoded to the frame-based structure, which was proposed in the previous section. Then it splits these frame points into frame groups, and each frame group is encoded as an I-frame point and P-frame points. Finally, these frame groups are added into the relevant frame group columns.

In contrast to the INSERT operation, to expand a trajectory T_{old} by given a trajectory ID TID and a new trajectory T_{new}, APPEND operation needs to select only the last frame group FG_{last} of T_{old} and decode it to get a trajectory segment. Then, T_{new} is connected to this trajectory segment. After this, it allocates and encodes this trajectory to get the new frame groups with related frame group column ID and TID. Finally, the last frame group FG_{last} is removed and these new frame groups are inserted into the frame group columns.

4.2.2 Multi-Thread based Approach

For importing based operations (i.e. SELECT and APPEND), the performance is mainly limited by I/O speed, since the data needs to be imported from an external source (e.g. hard disk). On the other hand, there is no I/O cost when loading data from memory. Therefore, the bottleneck of loading based operations (i.e. SELECT and DELETE) is the speed of CPU, since these operations rely on sequential scanning of the main memory, which is operated by the CPU. As discussed previously, it is easy to see that the processing speed of SELECT and DELETE operations on a traditional row-oriented data structure is faster than on an I/P frame data structure because an I/P frame structure needs multiple sequential scans to re-build the original trajectory; and a traditional row-oriented data structure can get the original trajectory via just one memory scan.

A general configuration for a modern server contains multiple CPUs with multi-cores built-in into each CPU. Therefore, it has become standard that modern servers support parallel computing for query processing, which can fully release the power of hardware. The problem of parallel query processing is a divide and conquer problem. Thus the challenges are how to divide a query into sub-queries to send to different threads and how to combine the sub-query results into the final result. Based on the advantages of the I/P frame data structure, the divide and conquer process is simple, and very stable. This is because, each frame group is highly independent and "share nothing" with other frame groups. In this section, the parallel approach for basic operations is discussed, which can reduce the running time of these operations, especially for SELECT and DELETE operations.

The SELECT operation is straightforward on a single thread, but not so straightforward for parallel computing. For SELECT operation processing, the algorithm searches for a given trajectory ID in the auxiliary table to get the range frame groups that contain the information about the query trajectory. Then this algorithm creates a threads pool, and each thread is assigned to a CPU core. In the next step, selected frame groups are split into m blocks, where m equals the number of threads in the thread pool and is based on user profile or system availability; and each block is assigned one single thread for processing. After this, for each thread, the decoding process can also be done independently and in parallel without needing any other frame group's information. Finally, these decoded frame groups (i.e. simple points) can be combined and returned as the final results. The implementation of the DELETE operation is similar to the SELECT operation, the main difference for the DELETE operation is that the frame groups do not need to be decoded. Hence, once the frame group is found, it can be removed from the column directly.

For INSERT and APPEND operations, parallel computing techniques are applied to the encoding phase. Therefore, after the input trajectory has been split into frame groups, these frame groups are assigned into m blocks and each block is sent to a single thread to encode to frame points (e.g. I-frame point and P-frame points). Unfortunately, as discussed before, the performance of such operations are limited by system I/O performance and system I/O will not gain any benefits from parallel computing techniques due to hardware limitations. However, applying parallel computing in both INSERT and APPEND can still improve the overall performance since the encoding time is reduced.

4.3 Advanced Operations

There are two fundamental advanced operation query types, window query and top-k nearest neighbour (kNN) query. In this section, a simple frame based approach and a hierarchical I/P frame data structure based approach are proposed to answer these queries. A parallel version for such approaches is also discussed.

4.3.1 Simple Frame based Approach

For query processing, it is necessary to reduce the number of reconstructed 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, a two phase processing is deployed that includes pruning and refining, which are shown in Algorithm 1. In the pruning phase, first the frame group columns which are out of the given time period (Line 1) are filtered out. For the I-frame point (i.e. original trajectory sample point) in each frame group, the algorithm calculates the maximum moving distance D_{max} for its P-frame points members (Line 2–4). The D_{max} is equal to the user defined maximum possible moving speed multiple by the time duration of this frame group. Then we use D_{max} as the boundary value of this trajectory segment for pruning (Line 5) based on different query types (i.e. window query or kNN query). In the refining phase (Line 6–8) , trajectory segments are decoded and reconstructed from the I-frame point with its related P-frame points and the final answer is found, based on the given query type. Details of the differences in query processing between the window query and the kNN query are discussed below.

Window Query: Given a spatio-temporal query window $W(Area, Interval)$, where $Area$ is the spatial window and $Interval$ is the temporal window. A window query is to find all trajectory segments TS ($TS.spatial$ denotes the spatial information and $TS.timestamp$ denotes the temporal information) in a dataset, where $TS.spatial$ is contained in $Area$ and $TS.timestamp$ is within $Interval$. Therefore, for the window query, at the pruning phase (Line 5), D_{max} is used to estimate the moving area of its related frame group (i.e. trajectory segment). If the moving area overlaps the query window, the P-frame points are decoded to find the actual answer to the window query (Line 7–8).

kNN Query: For the kNN query, this algorithm calculates the distance from the I-frame point to the query point and subtracts the D_{max} as the lower bound. If the lower bound is larger than the current k^{th} closest true distance, which means this trajectory segment cannot be closer than the current best examined trajectory, it can be pruned safely (Line 5). Otherwise, this trajectory segment moves into the refining phase (Line 6–8) to find its true distance to the query point. Since it is each frame group (i.e. the trajectory segment)being processed, and not the complete trajectory, it is necessary to avoid different frame groups with same trajectory ID being pushed into result set.

Algorithm 1: Frame Search Algorithm

```
Input: Query $Q$, Frame structure $F$
Output: Results set
1  $ColumnIds$ ← TimeInterval($Q$, $F$);
2  for $i \leftarrow ColumnIds.start$ to $ColumnIds.end$ do
3      foreach frame group $FG$ in $FrameColumn.get(i)$ do
4          $D_{max}$ ← CalculateDistance($FG$, $maxSpeed$);
5          if Check($D_{max}$, $Q$) then
6              $Tr$ = Decode($FG$);
7              if CheckTrajectory($Tr$, $Q$) then
8                  Update($Tr$);
9              end
10         end
11     end
12 end
```

This situation can lead to several trajectory segments with same trajectory ID (i.e. these segments belong to one trajectory) to be consider as final results. It can cause the number of final results (i.e. the returned trajectories) to be less than k, which will make this query fail. Therefore, the priority heap is extended 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 to see whether there is an element containing the same trajectory ID as the new frame group. If it exists, the system will only update the existing element by using this new frame group rather than creating a new element in the results set.

4.3.2 Hierarchical Frame based Approach

During the query processing, the most time consuming part is the sequential searching of the I-frame columns. To reduce the searching time, this section proposes a new hierarchical frame structure that expands on the simple frame based approach.

As Fig. 4.1 shows, this structure is built from bottom to top, which is based on the sample I/P frame data structure to reduce the number of I-frame columns visited during query processing. To reduce the complexity of the whole multi-layer structure, the frame group is still used as the basic unit in the structure. First of all, this algorithm extracts the I-frame columns from the current top level to build a new I/P frame structure layer, which becomes an upper level of the current level. Then it assigns every n I-frame columns into a frame group, where n is the number of frames in a frame group. After this, it uses the same encoding methods proposed in the previous section to encode the I-frame columns in the new frame groups. That means, in each new frame group, it can keep the first I-frame column unchanged as the new I-frame column and re-encode the rest of the I-frame columns as new P-frame columns. In Fig. 4.1, an example of the process used to build a level 2 structure from a level 1 structure and build a level 3 structure from a level 2 structure, respectively. During the processing, new layers are built until the time duration of the frame group is larger than the average time duration of the trajectories at the current level. This is because if it does not stop building at this level, most trajectories will be represented as a single frame point at the upper level, which cannot assist in improving query processing.

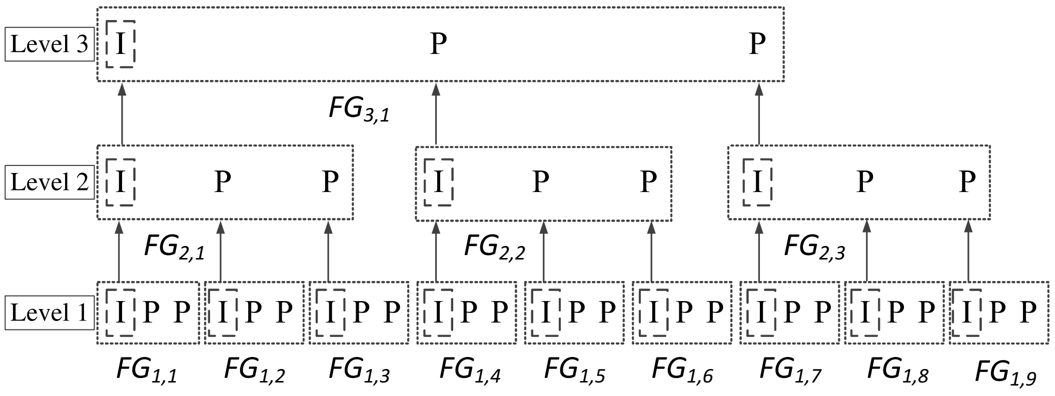


FIGURE 4.1: Hierarchical I/P-frame Data Structure

Although the encoding technology can reduce the space consumption substantially, keeping the I-frame column information in each level is still consuming space and will limit the number of layers in hierarchical structure due to memory space limitations.

When building a new layer, for each new I-frame point that needs to be copied from the lower level, it copies only the memory address from original I-frame point instead of copying the information in full. The full information for such I-frame points is kept in the bottom level frames only. Therefore, we can reduce the memory consumption for this data structure.

In addition, the same strategy is used to copy/encode new level P-frame points. Hence, when the algorithm accesses such a P-frame point, it can get the information directly without needing to decode the P-frame point, which can increase the traversing speed of the algorithm on this data structure.

At the same time, in the highest level of the hierarchical structure, the frame points for 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 points becomes longer. Hence, it is easy to see that the bound D_{max} will be too large to trigger the pruning. A large value of D_{max} will increase the searching space dramatically, which can reduce the performance of query processing or even make it the same as the exhausted search. To solve this issue, for each frame group starting at the bottom level, this algorithm calculates its MBR information and embeds it into its I-frame point instead of calculating the D_{max} during query processing. The upper level MBR information is calculated based on the MBR information of its lower level frame groups to keep the upper level MBR covering all of 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 the whole hierarchical structure is the same, this two phase processing can be maintained with fewer changes. To traverse the hierarchical structure, this algorithm uses the best first algorithm as shown in Algorithm 2. Firstly, it initiates a priority heap and pushes the first level frame groups within the query time range into it (Line 1). Then if a popped frame group is already in the bottom level, it uses an algorithm the same as Algorithm 1 to update the results set (Line 3–4). Otherwise, the algorithm travels into next level by decoding the popped frame group directly, and filtering out decoded frame groups in next level that are outside the time range of query (Line 6–7). Finally, it examines the filtered frame groups and updates the priority queue in Line 8–9. Now we detail the process for each type of query:

Window Query: For each selected frame group, the algorithm checks the MBR information in its I-frame point. If the MBR overlaps the query window, it decodes them. For example, in

Algorithm 2: Hierarchical Frame Search Algorithm

```
Input: Query $Q$, Hierarchical Frame structure $HF$
Output: Results set
1  $PQ \leftarrow$ initial($HF$, $Q.timeRange$) ;
2  while $FG \leftarrow PQ.pop()$ do
3      if $FG$ at bottom level then
4          Same as Algorithm. 1;
5      else
6          $framegroups \leftarrow$ Decode($FG$, $HF$);
7          foreach frame group $FG$ in $framegroups$ do
8              if Check($FG$, $Q$) then
9                  $PQ.push(FG)$;
10             end
11         end
12     end
13 end
```

Fig. 4.1, if frame group $FG_{3,1}$ at level 3 is popped from the candidate list, it then selects each 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 the same as for the simple frame based approach (Line 4).

kNN Query: For the kNN query, the algorithm uses the minimum distance from the MBR to the query point as the new low bound for the pruning phase. The $Check()$ function calculates its low bound and pushes it into the priority heap. As for Algorithm 1, if a frame group is already in the bottom level, it calculates the distance from the query point to the frame group. If this distance is less than the closest distance so far, the frame group with this distance is pushed into the hashed priority heap to update the results set.

4.3.3 Parallel Query Processing on Frame Structure

A general configuration of a modern server typically contains multiple CPUs with multi-cores built-in each CPU. Therefore, most standard servers will support parallel computing for query processing, which can fully release the power of parallel computing. The problem of parallel query processing is like a divide and conquer problem, for example, how to divide a query into sub-queries to send to different threads and how to combine the sub-query results into the final result. Based on the advantage of an I/P frame data structure, where each frame group is highly independent and "share nothing" with other frame groups, the divide and conquer process is simple, and very stable. Algorithm 3 shows the parallel query processing of a hierarchical structure. Firstly, the algorithm creates a thread pool to manage the process threads (Line 1). Then it assign each frame group column to a different thread to process and pushes the results to the queue (Line 2). After this, parallel decoding is used to get the next level frame groups from the popped frame group and assign them to a different thread. Each individual thread runs Algorithm 2 to check and keep traversing the hierarchy. After this, it merges the results from each individual thread. The details of approaches for the parallel window query, and the parallel kNN query are discussed below.

Algorithm 3: Parallel Search Algorithm

```
  Input: Query $Q$, Hierarchical Frame structure $HF$
  Output: Results set
1 $TP \leftarrow$ thread initial;
2 $PQ \leftarrow$ parallelInitial($HF$, $Q.timeRange$, $TP$);
3 while $FG \leftarrow PQ.pop()$ do
4     parallelDecoding($FG$, $HF$, $TP$);
5     Run Algorithm 2 in each single thread;
6     Merge results from each thread and do Update();
7 end
```

Window Query: To store the results in parallel, an $array[x][y]$ is created to organise the result trajectories; and each $array[x]$ represents a result trajectory and each $array[x][y]$ represents a sample point from a result trajectory. If a thread finds its results contain new trajectories (new

IDs), it will create new arrays in this array. When creating the new array for a new result, this thread locks this object to block the new array creation request from other threads until finished to avoid creating multiple arrays for one trajectory.

kNN Query: The challenge of parallelisation of the kNN query is that the value of the closest 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 closest 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 closest so far variable and the hashed priority queue. Then, this thread updates the closest so far variable if the distance to this candidate is less than current value; and updates the hashed priority queue. Finally, this thread releases the locks and sends a signal the to system to wait to assign the next frame group. The system continues assigning frame group(s) to free threads in the thread pool until all the frame groups in the candidate list have been investigated.

4.4 Analytic Operations

In this section, the problem definition of the trajectory similarity search, which considers both spatial and temporal information, is introduced. Then, the algorithms for processing the trajectory similarity search on the I/P frame data structure are discussed.

4.4.1 Problem Definition

The general definition of the trajectory similarity search is that given a query trajectory, the algorithm will find trajectories in trajectory database, which are similar to the query trajectory (i.e. the similarity between query trajectory and result trajectories is high). In general, the similarity is calculated based on the distance between the query trajectory and the trajectories in database, and the distance is calculated by similarity measures such as DTW and LCSS etc. However, this general definition is not suitable for 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 practice, most similarity trajectory analytic tasks focus on a particular time period, for example, finding the different patterns of traffic between the peak time and off-peak time every day. Therefore, the traditional trajectory similarity search cannot return useful results for such trajectory analytic tasks when using large scale datasets. The second issue concerns the variable length of trajectory data. As discussed previously, the length of each trajectory in the database is different. For example, if we compare the similarity between a two hour trajectory and a two month trajectory, it does not make any sense for a similarity analysis. To solve these two issues, the definition of a trajectory similarity search is extended and refined in the following, where we assume that all of the trajectories are frame encoded.

Given a time period $[HS, HE]$, a query trajectory T_q and a threshold α, a trajectory similarity query is used to find a set of trajectory segments $T_iS_j \in T_i$, where the length of T_iS_j equals T_q and T_i is the original trajectory in trajectory database. Meanwhile, for each frame point $p^j_m \in T_iS_j$ and $p^q_m \in T_q$, the distance $d(p^j_m, p^q_m) < \alpha$, where $1 \leqslant m \leqslant l$ and l is the length of T_q, then trajectory segment T_iS_j matches T_q. Moreover, the time duration of both T_iS_j and T_q must be within the given time period $[HS, HE]$.

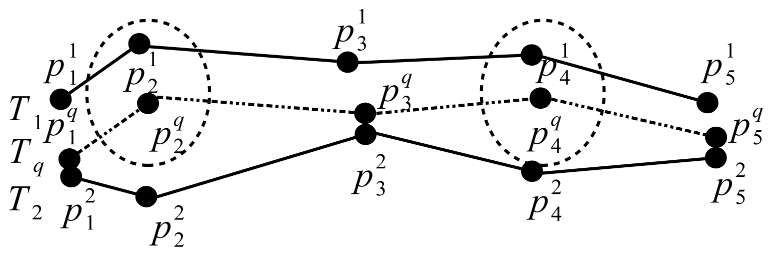


FIGURE 4.2: Similar Trajectory

Consequently, the minimum distance between two trajectories (i.e. the minimum distance between sample point pairs) is not used 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 Fig. 4.2 shows, the trajectory T_1 is similar to T_q, however, if we use the minimum distance to measure the similarity, T_2 will be considered more similar than T_1 to T_q, since the minimum distance between T_2 and T_q is less than between T_1 and T_q. Therefore, this is the reason why we compare the distance for each point pair to make sure the result trajectory segment is similar to the query trajectory in practice. 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 α and the distance from p_2^{TS} to p_2^q must be less than α and so on. Based on this measure, the trajectory T_1 is now 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 α (the α is shown as the circle). For example, in Fig. 4.3, assuming there is a trajectory T_1 and a query trajectory T_q, for the similarity query, the time period $[HS, HE]$ is set from t_1 to t_7 and the threshold is α, where $HS = t_1$ and $HE = t_7$. Then for trajectory T_1, as we can see that a sub-trajectory from p_1^1 to p_3^1, which is denoted as T_1S_1, and another sub-trajectory from p_5^1 to p_7^1, which is denoted as T_1S_2, 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 the time period is changed from t_1 to t_5, although the sub-trajectory p_5^2 to p_7^2 matches the query trajectory, the timestamp of p_6^2 and p_7^2 is out of the given time period. Hence, sub-trajectory T_1S_2 cannot be returned as a result.

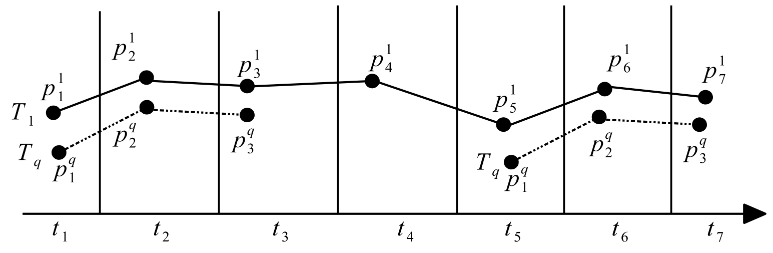


FIGURE 4.3: Example of a Similar Trajectory

4.4.2 Sliding Window based Approach

The naive way to solve this query is to decode the trajectory from the 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 as it does not utilise the advantages of the frame structure very well. Therefore, to maintain the efficiency of the trajectory similarity search on the frame structure, three challenges need to be addressed. The first challenge is to prune unnecessary the trajectories (i.e. frame groups) effectively and efficiently, which means that decoding the frame should be avoided in the pruning processing as it will add to the running time. To calculate the actual distance, the candidate frame groups need to be decoded to the original sample point. This case raises the second challenge, which is the requirement to decode the frame to the original sample points very quickly. The last challenge is to find the trajectory segments from candidate trajectory set efficiently. To solve these challenges, a novel method is proposed, which includes three-phase, pruning, decoding and searching. This three-phases method is discussed below.

In order to prune the frame groups and process the trajectory similarity search, the algorithm first synchronise the query trajectory by using trajectory calibration techniques [106], if the query trajectory does not come from the same dataset. This is because, if the trajectories are heterogeneous, the similarity measure may suffer from the accuracy problem, which is discussed in [106]. After the query trajectory is calibrated, the algorithm selects frame groups that are within a given time period, and starts the first phase of query processing, which is the pruning phase.

In the pruning phase, the algorithm uses the maximum moving distance D_{max} to prune the frame group. So, the I-frame point is tested first and if the minimum distance between the I-frame point and the query trajectory is larger than $D_{max} + \alpha$, it means that this frame group does not contain any sample point that is closer to any point in the query trajectory than α and, therefore, the frame group can be pruned safety. After all of the frame groups are scanned, the filtering processing is executed on these candidate frame groups.

The filtering process sorts the candidate frame groups by trajectory ID. After that, it can get the length of each continuing frame group (i.e. number of sample points) and compare that with the length of the query trajectory. If the length of a continuing frame group is less than the length of the query trajectory, then 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, frame groups FG_1 and FG_5 are pruned, the rest of the frame groups FG_2, FG_3 and FG_4 are put into the 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 contain). If the total length of these frame groups is less than the length of the query trajectory, we can filter out the frame groups FG_2, FG_3, FG_4 directly as they cannot contain the result trajectories.

In the decoding step, the candidate frame groups can be decoded directly to re-construct the candidate trajectories. In order to find the exact similar trajectory segments, it uses a sliding window query to check each candidate trajectory segment. The size of the sliding window is equal to the length of the query trajectory. During query processing, for each candidate trajectory, this window is moved along this trajectory, and the actual distance of each point pair is calculated to check whether there is a segment of the candidate trajectory that meets the requirement. For example, it can first test the trajectory segment, which is contained in the window, and then it slides the window on to the next point to repeat the computation task. It keeps 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 of the query trajectory, as shown in Fig. 4.4, in this case, the average distance from the query trajectory to each trajectory segment is calculated and the one with the minimum average distance is selected. In this example, the trajectory segment from p_4^1 to p_6^1 is returned as the result. Otherwise, if two trajectory segments are not connected, then both segments are returned in the final results.

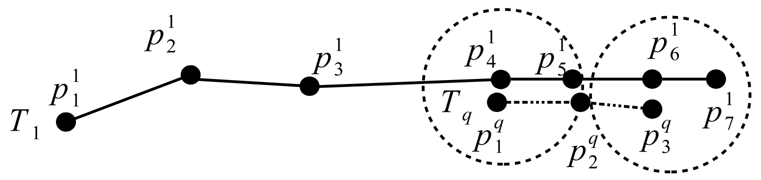


FIGURE 4.4: Example of Connected Trajectory

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 low performance of pruning phase, while the searching space is now very large. The second is that the decoding phase still needs more optimisation to reduce the processing time. Furthermore, using a sliding window approach is time-consuming. This is because, in most cases, the length of the query trajectory is much shorter than that of the candidate trajectory, which means the time complexity of search phase is $O(ab)$, where a is the length of the candidate trajectory and b is the length of the query trajectory. Therefore, this algorithm needs further optimisation to improve its performance.

4.4.3 MBR based Approach

As discussed previously, each frame group has been embedded into the MBR. Therefore, in the pruning phase, this approach calculates the MBR information of the query trajectory first. And then this algorithm uses the minimum distance between the MBR of the query trajectory and the MBR of each frame group as the new boundary value, which is also combined with α, to prune the frame groups. Using the MBR information can improve the performance significantly as it will reduce the search space considerably. Meanwhile, parallel processing can be invoked in the filtering phase, since each frame group is independent, which is the one of advantage of the frame structure. Following this, each frame group can be assigned to a different thread to check against the bound values. In addition, the filter step remains the same, since it can be done very quickly.

In the decoding step, parallel processing is invoked to further accelerate the decoding speed. Based on this, each frame group will be assigned to different threads for decoding. To continue improving the performance of the searching phase, we utilise the benefits of the frame structure. As all of the trajectories are synchronised, we can transform trajectories to string format (e.g. each sample point is considered and converted a special character). Therefore, if the distance between two sample points is less than α, these two points can be looked at as being the same character. Moreover, it allows us to perform text similarity measures to do the similarity search on the trajectory. Text similarity searching has been extensively studied, and the KMP algorithm [53] is used in the algorithm. There are two advantages of using KMP to do similarity search, the first is that the KMP algorithm is easy to extend to support the distance calculation, the second that is the time complexity of the KMP algorithm is linear $O(a + b)$, where a is the length of candidate trajectory and b is the length of the query trajectory. Hence, it is more efficient than the sliding window computation.

4.5 Summary

In this chapter, all queries/operations that are supported by SharkDB can be put into three categories. For each category, several algorithms are proposed to support efficient query processing. In the basic operation category, parallel based algorithms are introduced to accelerate transaction based operations. A hierarchical frame based approach with parallel implementation is proposed to efficiently process window queries and kNN queries. Finally, a MBR+KMP algorithm is presented to increase the performance of trajectory similarity searches.

Chapter 5

Performance Evaluation

5.1 Introduction

In the previous chapter, a set of approaches were proposed to gain significant processing efficiency for a large range of queries. Therefore, it is essential to conduct comprehensive experiments to demonstrate the effectiveness and efficiency of the proposed approaches. In addition, such experiments are still not enough to evaluate a database system. Generally, to evaluate the overall performance of a database system, the market will use workload benchmarks.

Moreover, the workload benchmarks are normally divided into two parts, online transaction processing (OLTP) workload benchmarks and online analytical processing (OLAP) workload benchmarks. OLTP workload benchmarks are designed based on a set of transaction based operations, which combine a number of read and write operations at the same time and only need to search a few rows of data such as SELECT and DELETE. On the other hand, OLAP workload benchmarks are mainly focused on analytic queries, which requires a large amount of sequential scans, but apply only to several columns.

Workload benchmark evaluations already exist for traditional database system evaluation. However, they have never been designed to evaluate trajectory based database systems. Traditional workload model is designed for transaction based databases and do not support typical trajectory queries such as the window query, the kNN query and the similarity search. Moreover, the main drawback of most workload benchmarks consider only a single or specific workload model. Such workload models do not suit real world applications because, real world applications usually contain different types of workloads and, therefore, require a mixed workload model. For example, in trajectory database system, a mixed workload model will contain all of the operations previously discussed.

Another drawback is that most traditional database systems are disk-oriented, which means the traditional workload models are mainly focused on evaluating the I/O performance because the I/O is the bottleneck of disk-oriented database systems. However, the I/O cost has been removed from main memory based database systems. Hence such workload models cannot reflect the actual performance of main memory based database system. In this chapter, a new design workload model is proposed to evaluate the SharkDB storage system as well as the traditional trajectory data structure.

To sum up, in this section, the experiments are divided into two parts that include operation level evaluation and system level evaluation. In the operation level evaluation, a set of experiments to evaluate the performance of different approaches against the traditional method are conducted. In the system level evaluation, a set of workload models under different usages are used to test the system as a whole.

5.2 Operation Level Evaluation

This section conducts extensive experiments on real trajectory datasets to study the performance of the proposed data structure and query processing methods.

5.2.1 Experimental Setup

In this evaluation, two real world trajectory datasets are used, which were collected from two big cities. The detailed statistics of the datasets are given in Table 5.1.

The experiments at this level compare the time cost of the proposed column-oriented data structures against the row-oriented data structure. The row-oriented data structure we implemented segments the trajectories and stores them in-memory, and is called a segmented trajectory database. In the segmented trajectory database, a trajectory is split into small segments, and each segment

TABLE 5.1: Trajectory Dataset

Dataset	# Sample Points	# Trajectories	Size
Dataset A	0.7 billion	243,194	14.6 GB
Dataset B	80 million	28,784	2.52 GB

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. To determine the optimal number of sample points in each segment, different values were trialled, and 50 was finally chosen as it achieved the best performance in the experiments.

Table 6.2 shows the default value and the range of each parameter. In this thesis, only the kNN query with $k = 5$ is examined, since how the length of the time interval of the query can affect the performance is attracted more interest in this evaluation. T_s is denoted as the sampling rate of the trajectory datasets, so $2T_s$ means the time interval of each frame is twice trajectory sampling rate. The default value of the time interval for a frame is T_s, to avoid the accuracy issue and to make a fair comparison with the baseline method. For the proposed hierarchical based approach, this evaluation sets the building stop condition to be equal to the time duration of each frame group when it reaches the average trajectory length. Meanwhile, this experiment uses 10 cores for the parallel model testing. For each set of experiments, 100 queries are generated and the average running time is calculated as the measurement. Each query is generated randomly with selected value of parameters. All the algorithms including the segmented trajectory database are implemented in Java and run on a sever with two Intel 8-core CPUs and 192 GB memory.

TABLE 5.2: 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$\%$–1$\%$
length of time interval (TI)	3h	1h – 11h

5.2.2 Performance Evaluation

Frame Storage Evaluation

In this section, this experiment compares the performance between the segmented trajectory database, frame storage, I/P frame storage and cache-aware I/P frame storage on real world datasets. To minimise the effect from the query processing approach, for the I/P frame storage and cache-aware I/P frame storage, only the simple frame based approach is used; for frame storage, sequential searching is applied to each frame column.

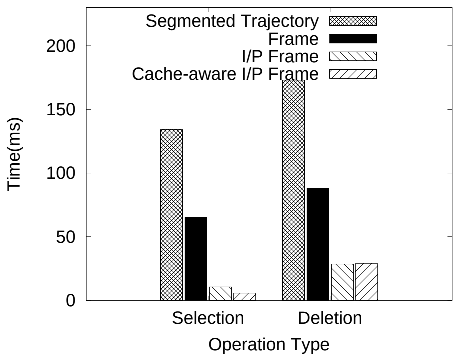
Performance of Basic Operations: Testing of the basic operations includes selecting by trajectory ID and deleting by trajectory ID. Since an insertion operation involves reading data from the hard disk into memory, the performance is limited by the read speed of the hard disks. Therefore, this experiment does not preform perform testing on insertion operations. The experiment will run following queries for testing, where id is a trajectory id chosen from the dataset randomly:

$$Selection : Select * from \quad Table \quad where \quad tid = id;$$

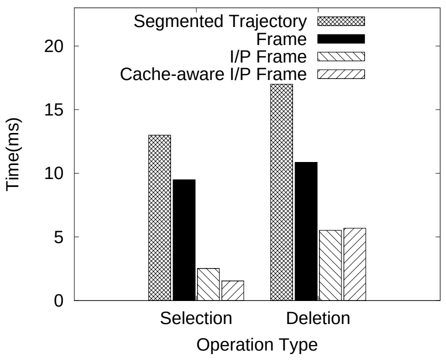
$$Deletion : Delete \quad from \quad Table \quad where \quad tid = id;$$

Fig. 5.17 shows the results of both selection and deletion operations. The segment trajectory database has the worst performance since it has to scan the segment table multiple times to reconstruct the whole trajectory. Such results also indicate that the bandwidth between CPU and memory becomes the new bottleneck of in-memory systems. Therefore, scanning the correct the segments in the segment table is time consuming if the scanning algorithm is not optimized for memory management.Moreover, the cache-aware I/P frame encoding method has the best performance due to this method having been optimised for both memory and CPU cache management. On the other hand, deletion operations are dependent on the memory speed since the data will need to be removed from memory. Therefore, the CPU cache optimisation has no effect on the efficiency of deletion operations.

Performance of Window Query: Firstly, the efficiency of these four approaches with different window sizes and a default time interval is evaluated. From the results shown in Fig. 5.2, we can see that the performance of the column-oriented data structure is very stable. This is because the main cost of query processing on a column-oriented data structure is searching the I-frame



(a) Dataset A

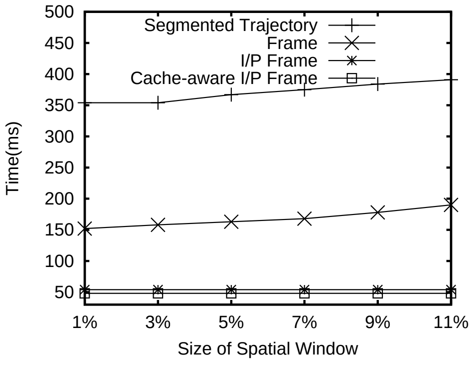


(b) Dataset B

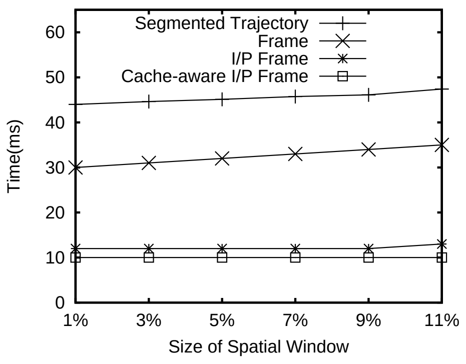
FIGURE 5.1: Results of Basic Operations

columns; and the number of I-frame columns that need to be searched is fixed since the time interval of all the queries is the same in this experiment. Hence, the performance of the I/P frame storage is better than the frame storage, since fewer I-frame columns need to be searched. Moreover, the cache-aware I/P frame storage optimises the decoding performance, so it has the best performance. Then, another experiment is conducted by changing the size of the time window of each query, the results of which are also shown in Fig. 5.2. Based on the results, as we can see that the length of the time interval is the main factor affecting query performance, especially for column-oriented data structures, since the length of the time interval decides how many frames need to be searched in I/P frame based storage.

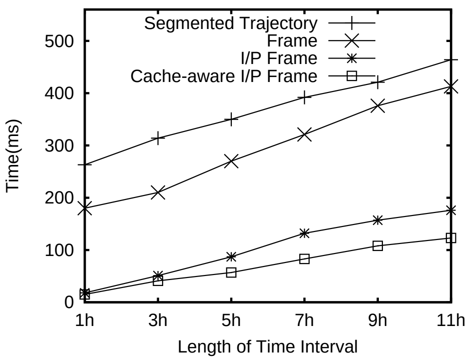
Performance of kNN Query: The query performance with regard to the length of the time interval for each query is investigated in these experiments. The results are shown in Fig. 5.3. Similar to window query, the length of the time interval for each query is also the main factor to affect query performance. But in contrast to the window query, the kNN query needs to investigate more frame points to get the final answers. Therefore, for I/P frame data structure, the cost of decoding P-frame points can also be much larger than for the window query. At the same time, the cache-aware I/P frame storage can reduce the decoding cost as it is optimised for the CPU cache to increase decoding performance. Hence, the cache-aware I/P frame data structure is much faster than regular I/P frame data structure, compared with the window query. Finally, we can see that the I/P frame data structure can increase the performance by at least 10 times as compared to a row-oriented database.



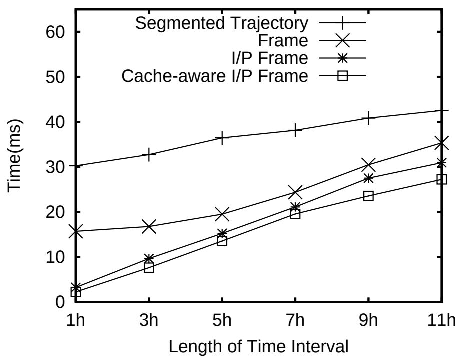
(a) AS in Dataset A



(b) AS in Dataset B

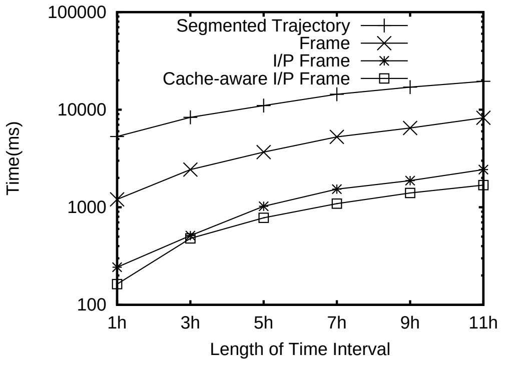


(c) TI in Dataset A

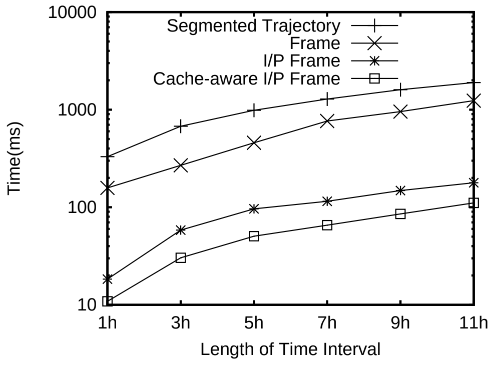


(d) TI in Dataset B

FIGURE 5.2: Performance of Window Query in Frame Storage Evaluation



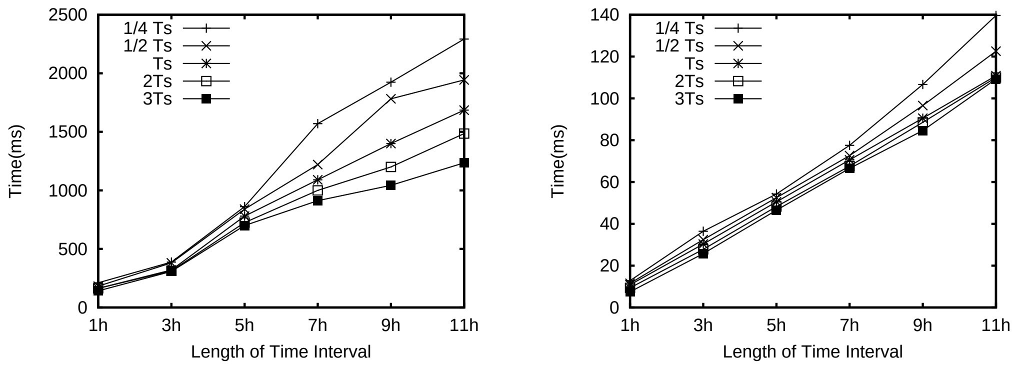
(a) Dataset A



(b) Dataset B

FIGURE 5.3: Performance of kNN Query

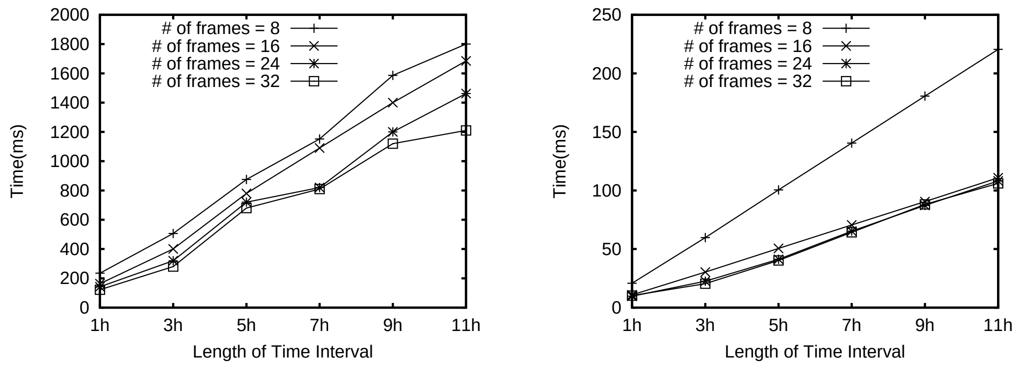
Effect of Frame Rate The next experiment studies the query performance as the frame rate varies, with the running time of the kNN query on dataset A used as the measurement. As the frame rate will cause the same effects in the three types of frame data structure, we only examine the cache aware I/P frame storage. The results are presented in Fig. 5.4. Without considering the accuracy, the experiment with $3T_s$ 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 points longer thus increasing the search area for each frame group. Therefore, it causes more candidate frame groups to be pushed into the candidate list and consequently reduce the performance in the refining phase.



(a) Dataset A (b) Dataset B

FIGURE 5.4: Effect of Frame Rate

Effect of Number of Frames in each Frame Group: Finally, the query performance with different n values is evaluated, using the same measurement method as the effect of frame rate experiments. The running time by the length of the time interval is reported in Fig. 5.5. As the basic frame data structure does not have a frame group structure; and both I/P frame data structure and cache-aware I/P frame data structure use the same frame group structure, in this experiment we examined only the cache-aware I/P frame data structure. It is easy to see that the performance increases as the value of n increases. This is because, in the query processing on the P-frames, the system can access the P-frame points directly without needing to search the P-frame columns and send to the CPU to decode, which reduces the search time and increases the performance. Meanwhile, the system can gain more benefits from the compression technique, which can reduce the consumption of memory bandwidth.



(a) Dataset A (b) Dataset B

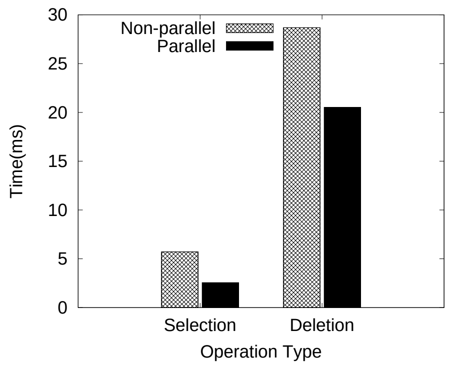
FIGURE 5.5: Effect of Number of Frames

Query Processing Evaluation

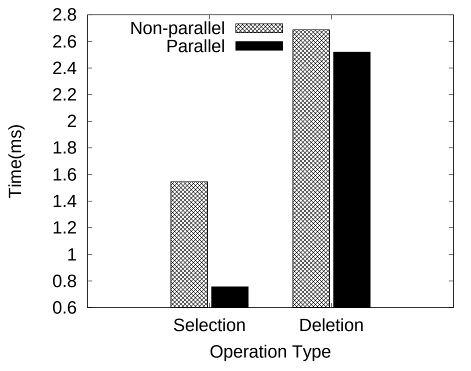
The next set of experiments focus on the performance of different query processing approaches: a simple frame based approach, a hierarchical frame based approach and a hierarchical frame based approach with parallel computing. We will test the performance of the window query and the kNN query.

Performance of Basic Operations Since both flat and hierarchical frame based approaches are designed for analytic query processing (i.e. window query, kNN query and similarity search), for basic operations this experiment involves only comparing a the parallel computing option with that of single thread processing. The results are shown in Fig. 5.6. As we can see, the performance of the operations using parallel processing significantly outperform that of the single thread. This is because the parallel computing can perform multiple frame group searching and re-construct trajectories at the same time.

Performance of Window Query Similar to the previous experiments, the results are illustrated in Fig. 5.7. It is no surprise that, the hierarchical frame based approach is better than simple frame based approach, which shows that using the hierarchical structure and MBR can reduce the search space significantly. On the other hand, the parallel computing technology can improve the performance as the time interval increases. This is because the overhead cost of parallel computing is more than that of a single thread, since it has to create a thread pool to manage the threads at beginning, which leads to a lower increase in overall performance if the investigated I-frame columns are small (i.e. short time interval). But this overhead cost is constant, so the parallel



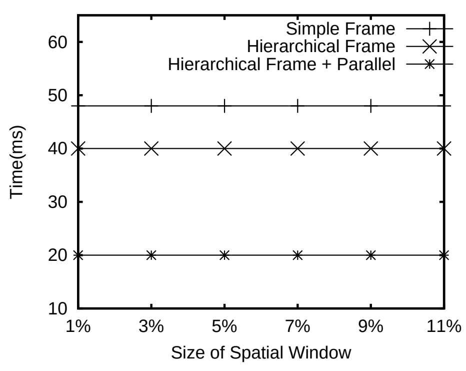
(a) Dataset A



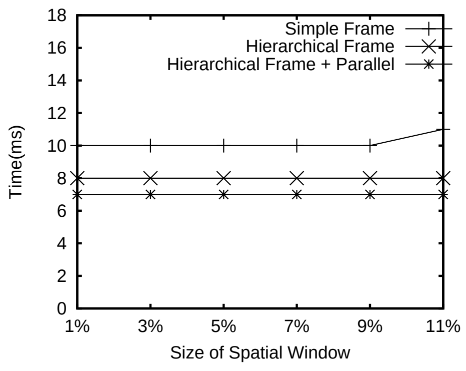
(b) Dataset B

FIGURE 5.6: Results of Basic Operations

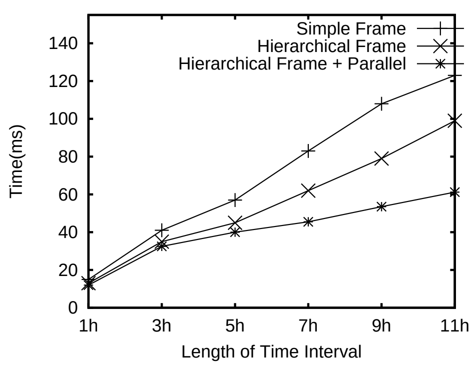
computing approach can achieve a high increase in performance when the search space is large.



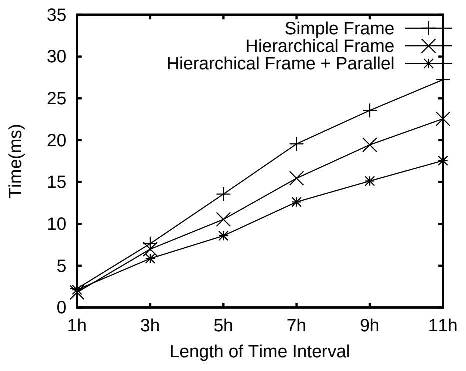
(a) AS in Dataset A



(b) AS in Dataset B



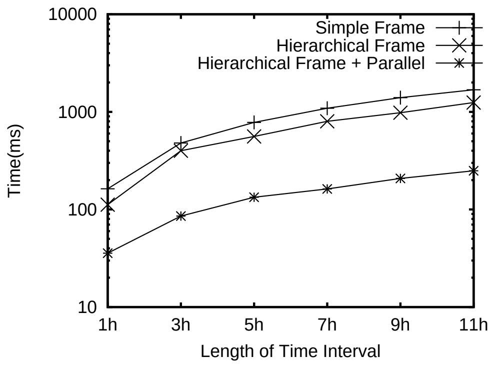
(c) TI in Dataset A



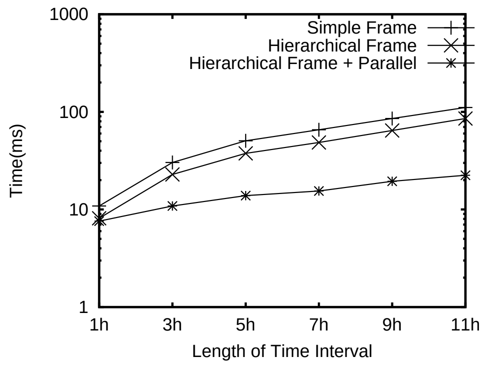
(d) TI in Dataset B

FIGURE 5.7: Performance of Window Query in Query Processing Evaluation

Performance of kNN Query Let us now evaluate the performance of kNN query with changing the length of the time interval, the results of which are shown in Fig. 5.8. As 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 times faster as compared with a single thread.

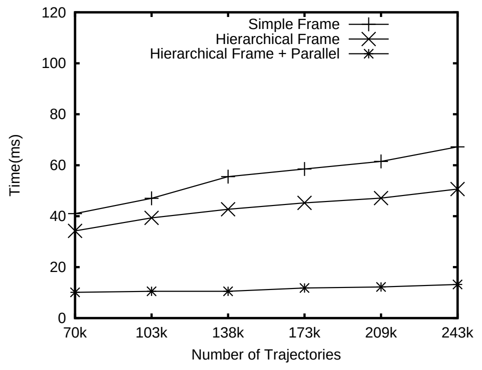


(a) Dataset A

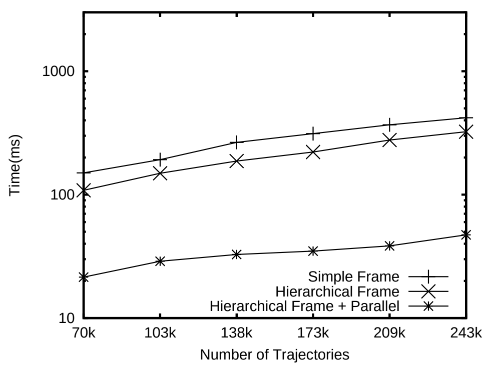


(b) Dataset B

FIGURE 5.8: Performance of kNN Query



(a) Window Query



(b) kNN Query

FIGURE 5.9: Scalability

Scalability Evaluation The scalability of all the approaches under two queries/operations is also evaluated. To achieve this, from dataset A, we randomly selected a number of trajectories with the number varying from 70 k to (approx.) 243 k. Therefore, when the number of trajectories is increased, the density of trajectories in a certain area (i.e. the area is covered by the whole trajectory dataset) will also increase. The running times are reported in Fig. 5.9, which illustrates that the time cost for each of the three data structure types increases linearly/sub-linearly with respect to the size of dataset. The hierarchical frame based approach with parallel computing achieves the best performance in this experiment.

Evaluation of Trajectory Similarity Search

In this section, four algorithms with different processing strategies are implemented. The similarity search algorithms are divided into two main parts. The first part comprises the pruning phase and decoding phase and the second part comprises the calculation phase. Details of the implemented algorithms are listed in Table 5.3, where S denotes a single thread based algorithm, P denotes a multi-thread based algorithm and RC is denoted as reconstructing algorithm (i.e. decoding algorithm). Moreover, the same real world datasets are used for this experiment, and the additional parameter settings are listed in Table 5.4.

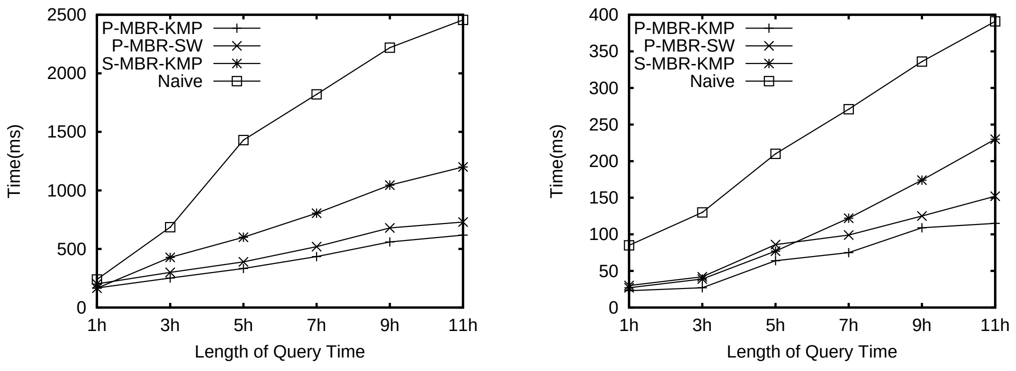
TABLE 5.3: Implemented Algorithms

Algorithm	The First Part	The Second Part
Naive algorithm	S-D_{max} + S-RC	Sliding Window
S-MBR-KMP	S-MBR + S-RC	KMP
P-MBR-SW	P-MBR + P-RC	Sliding Window
P-MBR-KMP	P-MBR + P-RC	KMP

TABLE 5.4: Parameters Setting on Similarity Tests

Parameter	Default Value	Range
# simple points of query	150	30–330
length of α (meters)	100	50–300

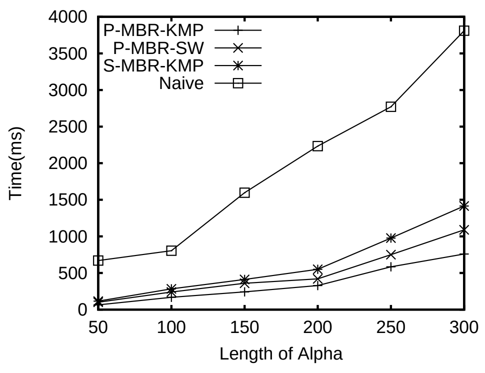
Effect of Query Time Interval The query time interval is a main factor that can affect the performance of query processing. The results of these algorithms are shown in Fig. 5.10 for both datasets. The running time of the naive algorithm is increasing very fast while the duration of the query time is increasing, since a longer query time will require more frame groups (i.e. more sample points) to be tested. The performance of the single thread MBR-KMP algorithm is better than that of the naive algorithm due to a better pruning algorithm and distance calculating algorithm, however, the trend of performance is similar to the naive algorithm. For multi-thread MBR-KMP algorithm, as we can see, its performance is same as for the single thread MBR-KMP algorithm. This is because the overhead cost of multi-thread processing is more than that for single-thread, such as thread pool management and thread allocation management. But, when the duration of the query time increases, the power of parallel processing starts to release, and as we can see, the performance of the multi-thread MBR-KMP algorithm is stable. Therefore, the performance of the multi-thread MBR-KMP algorithm dominates that of the other two algorithms.



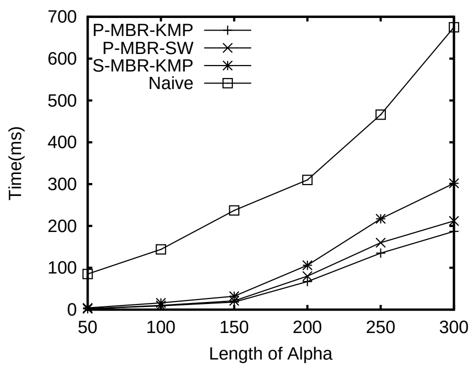
(a) Dataset A (b) Dataset B

FIGURE 5.10: Effect of Query Time Interval

Effect of α The next experiment studies the different values of α, since it is another factor that affects the performance of queries. The results of these algorithms are reported in Fig. 5.11. Comparing the multi-thread MBR-DP algorithm and the multi-thread MBR-KMP algorithm, the multi-thread MBR-DP algorithm is 20% faster than multi-thread MBR-DP algorithm, which means KMP algorithm can reduce the computational cost compare with the sliding window algorithm. However, as we can see, the running time of the multi-thread MBR-KMP algorithm is five times faster than the naive method, which shows the advantage of this pruning method on a frame structure as it can prune the unnecessary frame groups effectively, and significantly reduce the time required for the last calculating step. The performance of the naive method decreases very quickly, since a larger α will lead the pruning method of the naive method to lose efficacy and increase the calculation workload. Moreover, the performance of the multi-thread MBR-KMP algorithm is stable, and shows that the parallel processing can maintain the performance of these tasks very well.



(a) Dataset A

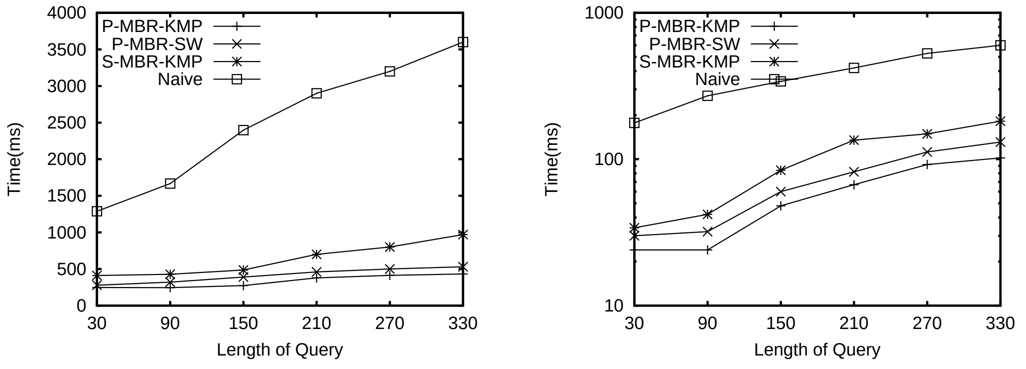


(b) Dataset B

FIGURE 5.11: Effect of α

Effect of Query Length The impact of varying query length is shown in Fig. 5.12. There is no doubt that the longer query trajectories need more running time to get the results. This is because, the longer query trajectories are likely to intersect with more trajectories in the trajectory dataset; and this will increase the number of trajectories pushed into the calculation phase. First of all, it can be seen that the performance of the multi-thread MBR-KMP algorithm is two times faster than single-thread MBR-KMP algorithm, which provides the evidence that parallel computing can gain many benefits from this novel frame structure, even if our algorithm is only partially paralleled (i.e. the pruning phase and decoding phase are paralleled, but the calculation phase is not). Moreover, it also shows that the parallel processing can significantly improve the performance of the pruning and decoding algorithms. In addition, the running time of the sliding window based algorithm can be seen to increase faster than the KMP based algorithm as a longer query length will require more calculations in the calculation phase.

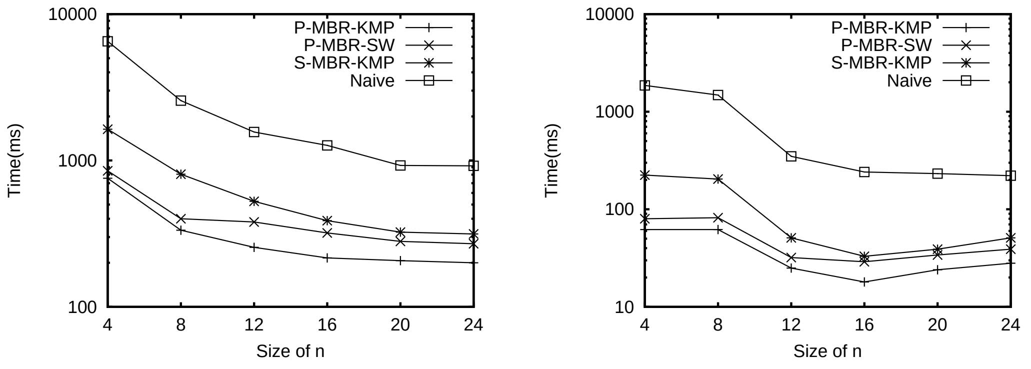
Effect of n In this test, the performance with different values of n is investigated. The results are shown in Fig. 5.13. As n ($n \leqslant 24$) increases, the performance of these algorithms also increases. The reason is that the larger the value of n is set in an I/P frame structure, the fewer frame groups will be checked in the pruning phase. Moreover, a frame group with more P-frame points will also have better performance than a frame group with fewer P-frame points, since decoding P-frame points is much faster than searching for 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 also increase considerably with a large n. Therefore, the value of n can affect



(a) Dataset A (b) Dataset B

FIGURE 5.12: Effect of Query Length

the performance of the similarity search more significantly than the window query and the kNN query, since the similarity search requires decoding more frame groups. In addition, a large value of n will also increase the size of the MBR or D_{max} of each frame group and thus reduce the efficiency of the pruning process. As we can see, when n is larger than 24, the increased decoding performance is counteracted by inefficient pruning processing.



(a) Dataset A (b) Dataset B

FIGURE 5.13: Effect of changing the valur of n

Effect of T_S The results of the similarity search are similar to the results of the window query and the kNN query, and are shown in Fig. 5.14. The lowest frame rate (i.e. $3T_s$) has the best performance. Moreover, the performance of the multi-thread MBR-KMP algorithm is much better than that of the naive algorithm, when the frame rate is set to its highest value (i.e. $1/4\ T_s$). As the highest frame rate has the largest data size in this experiment, it shows that the parallel computing is a good idea for processing of large amounts of data.

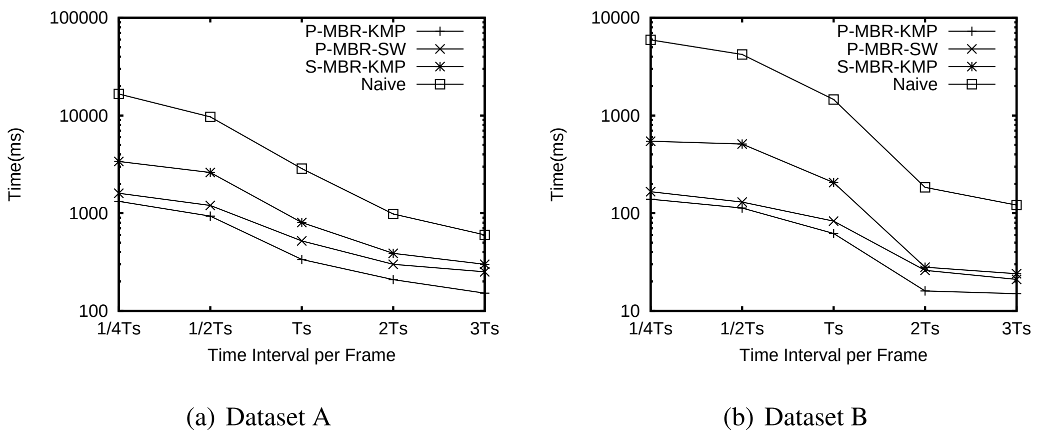


FIGURE 5.14: Effect of TS

5.2.3 Accuracy Evaluation

The accuracy testing is designed particularly for frame type of data structure, and as mentioned above, the time interval of each frame (i.e. frame rate) will affect the accuracy of the 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 the encoded data in the memory* divided by the *size of the original data in the memory*.

Effect of Frame Rate The accuracy is evaluated by changing the time interval 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 the same query processing on our approaches. After getting 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 the proposed approaches is $1, 2, 3, 4, 6$, and its recall is 80%. The results are shown in Table 5.5 with their compression ratios. As expected, increasing the time interval of a frame will result in a low accuracy rate, but the compression performance is increased in this case. On the other hand, when the time interval of each frame is less than the sampling rate of the trajectory, there is no improvement in accuracy, but the need for memory is increased.

Based on the previous experiment, it is clear that changing the frame rate in the I/P frame structure can affect the accuracy of results significantly. In this experiment, we set five different

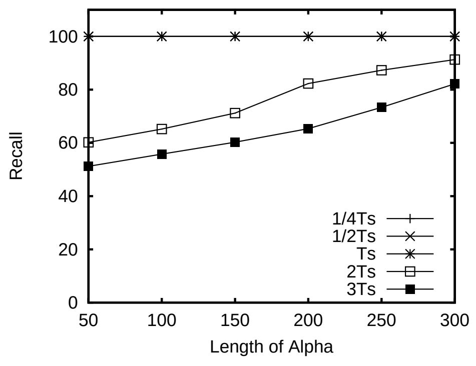
TABLE 5.5: The results of Accuracy

	Dataset A		Dataset B	
Time interval	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%
$2T_2$	82%	11.2%	86%	11.2%
$3T_3$	76%	7.4%	77%	7.4%

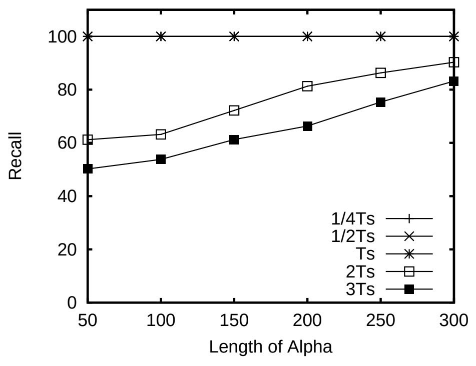
frame rates, which are same as for the previous experiment, but with different α values to test the accuracy of the similarity search. The α is set to a value from $50m$ to $300m$. The accuracy is also measured by recall, and the results are shown in Fig. 5.15. These is no doubt that the accuracy will be reduced, when we set the frame rate to be the same as the original trajectory sampling rate. As 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 α value is small, the accuracy of the results will not be 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 the window query and the kNN query, the accuracy can be affected significantly when the frame rate is lower than the original trajectory sampling rate as the shape of the trajectory has been changed due to many sample points of trajectories having been removed. Finally, as we can see, the accuracy is very low with a small value of α. 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.

5.2.4 Compression Ratio Evaluation

Then this experiment investigates the compression ratio with regard to the number of frames in a frame group. The results are shown in Table 5.6 for both datasets. We clear observe that e.g. the higher the value of n, the higher compression ratio we can achieve. This is because increasing n will reduce the total number of I-frame columns in the storage system; and the cost of retrieving P-frame points is much less than I-frame points.



(a) Dataset A



(b) Dataset B

FIGURE 5.15: Accuracy

TABLE 5.6: 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%

5.2.5 Synthetic Trajectory Data Evaluation

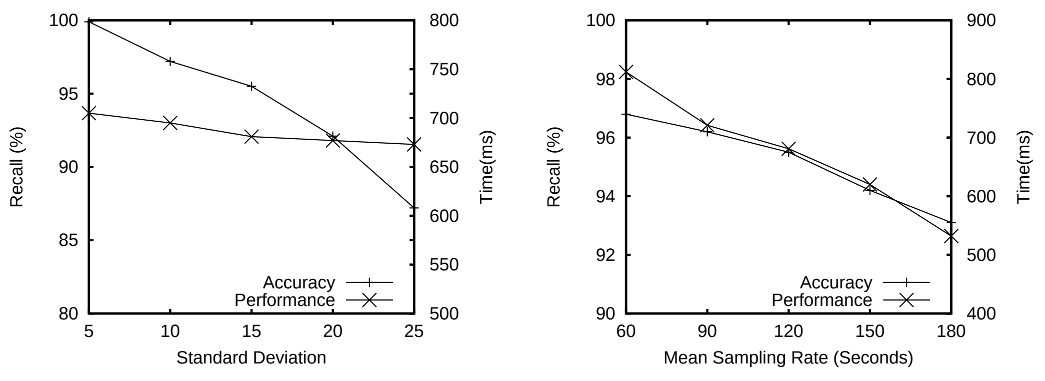
In this section, a synthetic trajectory dataset is generated with 3 billion sample points; and the sampling rate distribution is based on normal distribution with different mean and standard deviation of sampling rate. The range of means and standard deviations of the sampling rates is shown in Table 5.7. Meanwhile the sampling rate ranges from 30 seconds to 240 seconds during generation. Moreover, in this experiment, the frame rate is set to equal to the mean sampling rate, and the same settings are used (kNN query) as in the previous experiment for accuracy and performance testing.

TABLE 5.7: Synthetic Evaluation Parameters Setting

Parameter	Default Value	Range
mean sampling rate (seconds)	120	60—180
standard deviation	15	5—25

Effect of Standard Deviation Fig. 5.16(a) illustrates the effect of different standard deviations of the sampling rate. As we can see, the heterogeneous trajectory can negatively impact the accuracy of the query processing, especially when the standard derivation is high. This is because, if the sampling rate of a part of the trajectory is higher than the mean sampling rate, some of the sample points will be removed during frame encoding. Therefore, the higher the standard deviation, the more sample points will be removed, thus decreasing the accuracy rate, similar to the evaluation of the effect of frame rate. However, as we can see that this system has a good tolerance for low standard deviations. 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 Fig. 5.16(b) shows the results of different mean sampling rates. We can find that performance increases with lower mean sampling rate, since fewer frame group columns will be scanned during the query processing. This is because, the larger mean sampling rate has a longer frame rate, which reduces the number of frame group columns in a certain time period. At the same time, based on the evaluation settings, the smaller mean sampling rate can increase the accuracy. This can be explained by 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 trajectories with high sampling rates.



(a) Standard Deviation (b) Mean

FIGURE 5.16: Synthetic Evaluation

5.3 System Level Evaluation

This section first introduces the ideas of benchmarks that are designed as a workload model to test the system. Then parameters selection is discussed, and finally, there is as a focus on the datasets that are used for evaluation.

5.3.1 Workload Model Design

A key motivation to use an in-memory database system to manage trajectory data is to gain significant processing efficiency for a large range of queries. Therefore, it is essential to have a set of common trajectory queries to use for testing traditional trajectory database design approaches.

To achieve this goal, five operations are invoked and divided into three categories in this workload model. The five operations are SELECT, DELETE, Window Query, kNN Query and Similarity Search. The reason why this design does not invoke INSERT and APPEND operations is that the performance of such operations is limited by the I/O performance, since the data needs to be read from external storage such as hard disk. The I/O performance evaluation is out of scope of this thesis as it focuses on in-memory systems. Hence, INSERT and APPEND operation are not included in the workload model.

As discussed previously, the basic operations are the "lightest" operations performed in SharkDB storage system, and the analytic operation is the heaviest operation. To fully test SharkDB, three types of workloads are prepared to simulate different system usages: a typical workload, a operational workload and an analytic workload. Each workload contains a different weighted mixture of the trajectory operations discussed above. The weight of each operation is marked as a percentage, for example, if the weight of a operation is 15%, it will be tested 15 times in a workload evaluation with 100 times testing for all operations. First of all, the typical workload will focus on transaction based operations (i.e. SELECT and DELETE). Secondly, the operational workload will focus on fundamental trajectory queries (i.e. window query and kNN query). The last workload is analytic workload, which will focus on analytic queries. The details of each workload are list in Table 5.8.

TABLE 5.8: Workload Setting

	SELECT	DELETE	Window Query	kNN Query	Similarity Search
Typical Workload	35%	35%	10%	10%	10%
Operational Workload	10%	10%	35%	35%	10%
Analytic Workload	5%	5%	10%	10%	70%

5.3.2 Parameters Selection and Datasets

The parameters of SharkDB can be divided into two categories, the first are the system based parameters such as the value of n and the frame rate. The second category is query based parameters such as the temporal window of the query or the average length of selected trajectory. The objective of this workload evaluation is to evaluate the performance of SharkDB under different workload usages, therefore, we keep the system based parameters is fixed and tune the query based parameters. For each operation type, the key parameters are selected based on previous experiments, and are the key factors that affect the performance of the various queries. The details of the parameters are listed in Table 5.9.

TABLE 5.9: Parameters Setting

Parameter	Default Value	Range
Length of Trajectory	3000(sample points)	1500–9000
length of time interval	3h	1h – 11h
length of α (meters)	100	50–300

The datasets used in the benchmark evaluations are the same datasets are used as the previous chapters. Each sample point contains three attributes, longitude, latitude and a timestampe. Each attribute costs 8 bytes to store in the main memory in raw format. Meanwhile, the trajectory ID will cost 4 bytes to be stored in I-frame column in the SharkDB. For delta encoding, we use 4 bytes to encode spatial information (the error of each sample point is less 1 meter). The details of datasets are listed in Table 5.10.

TABLE 5.10: Trajectory Dataset

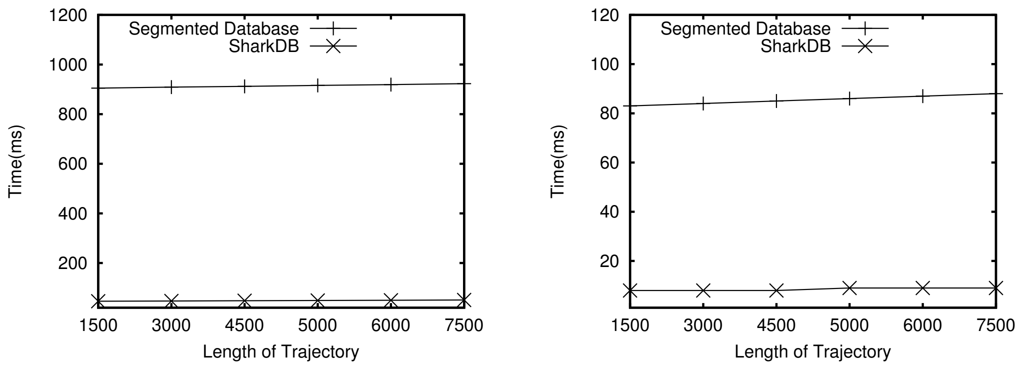
Dataset	# Sample Points	# Trajectories	Size
Dataset A	0.7 billion	243,194	14.6 GB
Dataset B	80 million	28,784	2.52 GB

In this benchmark evaluation, the segmented trajectory database is used, which is same as previous experiments. For each set of experiments, 10 CPU cores are used for SharkDB; and a total 10000 of queries are generated in each evaluation tests. And the number of each type query are based on the weighted setting in above. Then the average speed is calculated as a measure. All benchmarks are run on a sever with two Intel 8-core CPUs and 192 GB memory.

5.3.3 Typical Workload Evaluation

In this section, the segmented trajectory database and SharkDB are tested under a typical workload model.

Tune Parameters of Basic Operations First, this workload compares the performance of SharkDB with the segmented trajectory database for different trajectory length, which is shown in Fig. 5.17 and Fig. 5.18. As we can see, even the performance of the DELETE operation in the segmented trajectory database is better than in SharkDB at the large dataset. The overall workload performance of SharkDB is still better than the segmented trajectory database. This is because SharkDB significantly improves the performance of advanced operations and analytic operations.

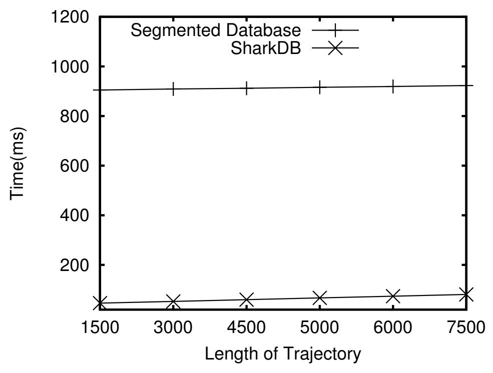


(a) Dataset A (b) Dataset B

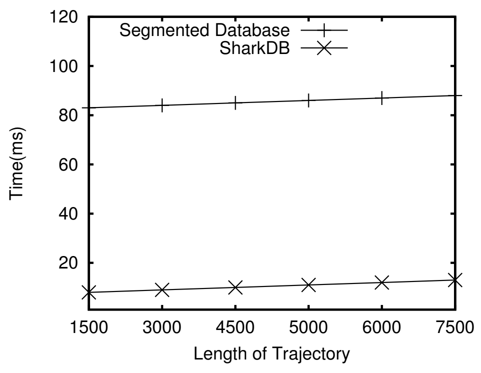
FIGURE 5.17: Typical Workload — Effect of Trajectory Length for SELECT

On the other hand, the trajectory based queries (i.e. window query, kNN query and similarity search) need more CPU time than the basic query to process, especially on large dataset. As the length of the trajectory increases, the performance difference between the segmented trajectory database and SharkDB reduces slightly, since SharkDB needs more time to search and decode frame codes. However, such cases will not affect the overall performance workload testing as a whole.

Tune Parameters of Advanced Operations



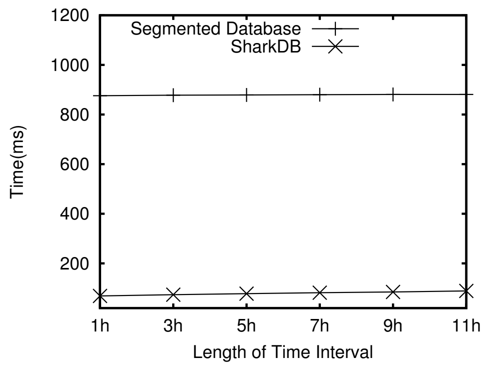
(a) Dataset A



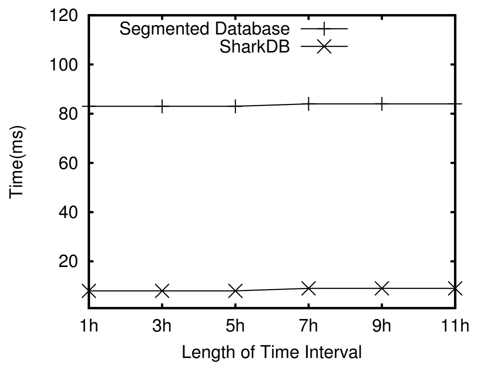
(b) Dataset B

FIGURE 5.18: Typical Workload — Effect of Trajectory Length for DELETE

This workload evaluation compares the performance of the different approaches for advanced queries. As Fig. 5.19 and Fig. 5.20 show, when the query time interval of the kNN query increases, the overall running time in the segmented trajectory database increases, since the kNN query is the most time consuming query for the segmented trajectory database. Even if the weight of the kNN query is small in this workload evaluation, it still can affect the overall performance of the segmented trajectory database. Moreover, the performance of SharkDB is still stable due to the proposed efficient kNN query algorithm. For changing query time intervals for the window query, the performance of both SharkDB and the segmented trajectory database is stable as the window query costs less CPU-time than the kNN query does.



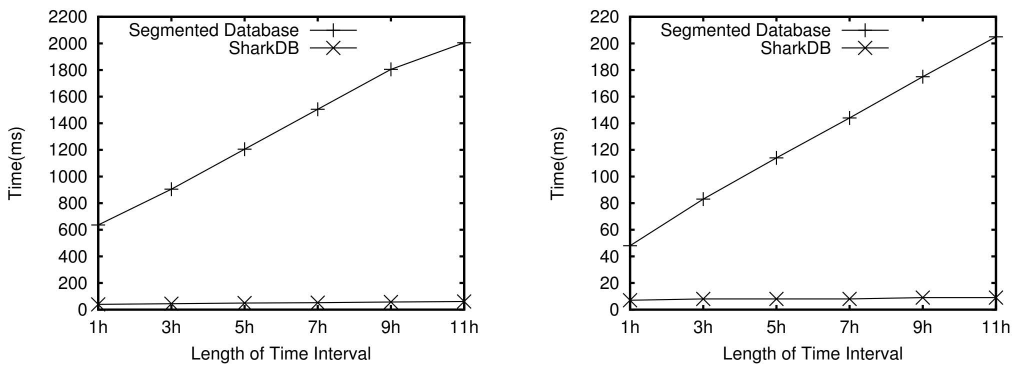
(a) Dataset A



(b) Dataset B

FIGURE 5.19: Typical Workload — Effect of Time Interval for Window Query

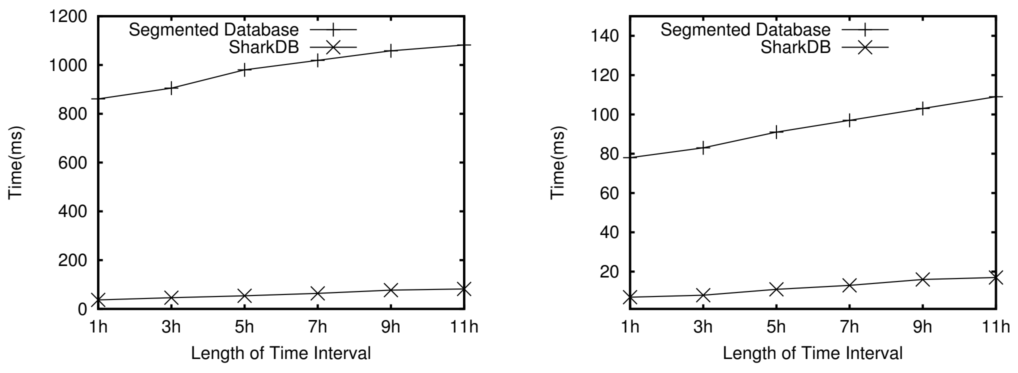
Tune Parameters of Analytic Operation



(a) Dataset A (b) Dataset B

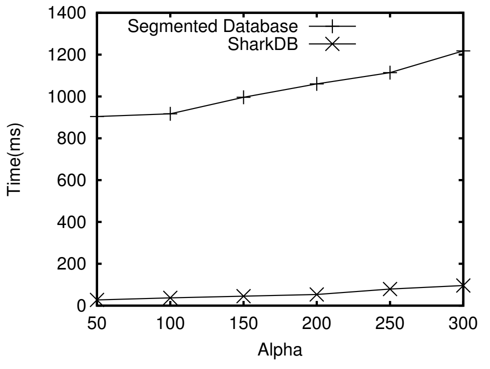
FIGURE 5.20: Typical Workload — Effect of Time Interval for kNN Query

Fig. 5.21 and Fig. 5.22 show the final results of this workload evaluation with two parameters, the length of time interval and α. As we can see, the average running time on SharkDB and the segmented trajectory database increases, when we increase either the length of time interval or the value of α. This is because the analytic query is sensitive to such parameters and the similarity search will cost more than 30 times running time compared to the basic operations. Therefore, the running time change for an analytic operation can still affect the overall performance even if the analytic operation has only 10% weight in this workload evaluation.

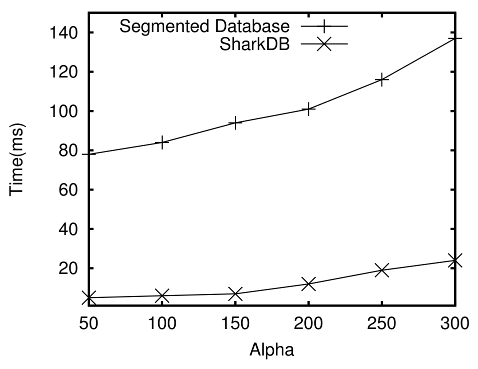


(a) Dataset A (b) Dataset B

FIGURE 5.21: Typical Workload — Effect of Time Interval for Similarity Search



(a) Dataset A



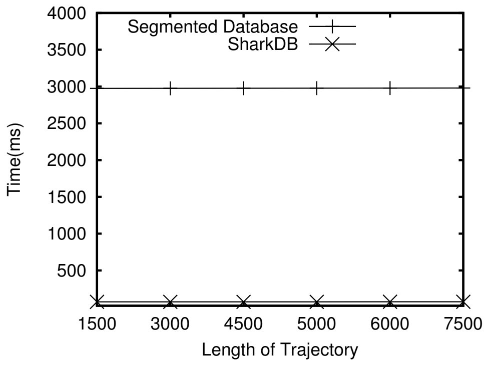
(b) Dataset B

FIGURE 5.22: Typical Workload — Effect of α for Similarity Search

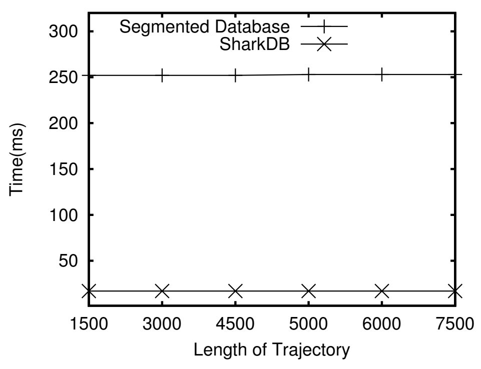
5.3.4 Operational Workload Evaluation

In this section, the segmented trajectory database and SharkDB are tested under an operational workload model.

Tune Parameters of Basic Operations The operational workload evaluation first conducts a set of experiments by using different trajectory lengths for basic operations, the results of which are shown in Fig. 5.23 and Fig. 5.24. As we can see, the overall performance of both SharkDB and the segmented trajectory database are not affected by this parameter. This is because the basic operations can be processed quickly in both systems under different trajectory lengths, which means the basic operations will no longer be a key factor to affect the performance of analytic based storage systems.



(a) Dataset A



(b) Dataset B

FIGURE 5.23: Operational Workload — Effect of Trajectory Length for SELECT

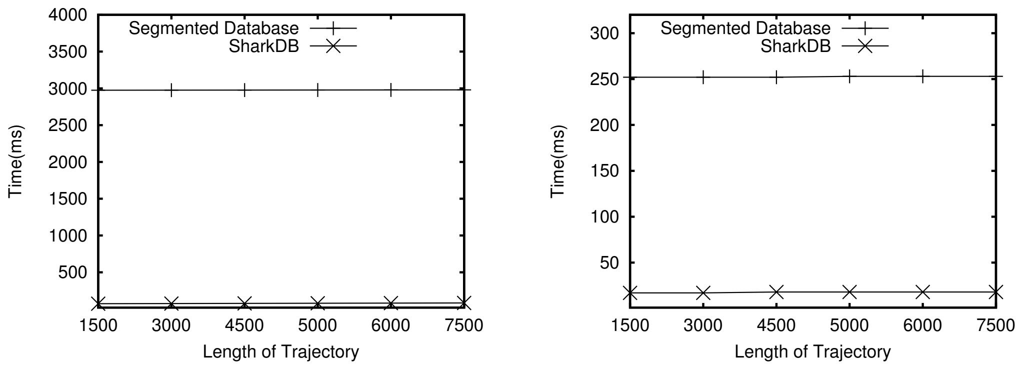


FIGURE 5.24: Operational Workload — Effect of Trajectory Length for DELETE

Tune Parameters of Advanced Operations Fig. 5.25 and Fig. 5.26 demonstrate how the length of query time interval can affect the overall performance. There is no doubt that SharkDB still achieve the best performance under all tests, since SharkDB significantly reduces the query processing time for both the window query and the kNN query. Due to the advanced operations having the most weight in this experiment, the performance trend is similar to that of previous experiments of single type of query in the previous chapter.

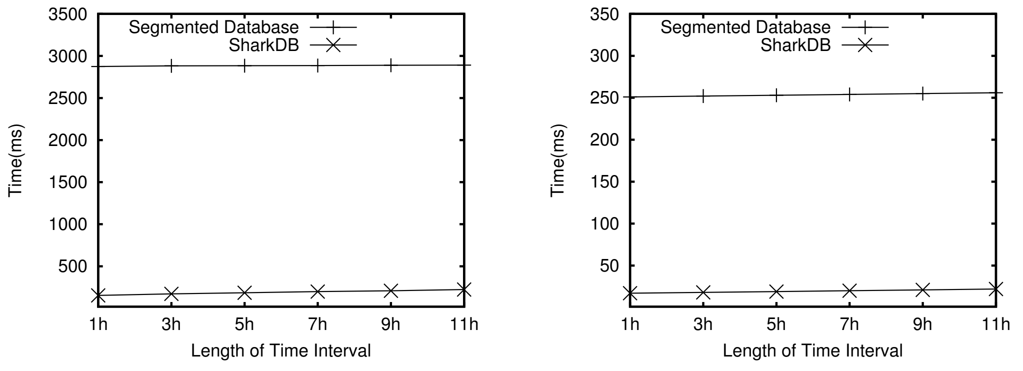
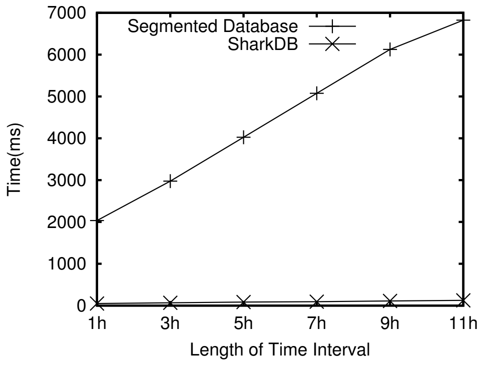
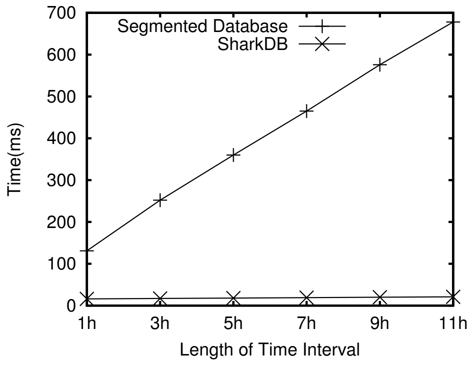


FIGURE 5.25: Operational Workload — Effect of Time Interval for Window Query

Tune Parameters of Analytic Operation The results of this evaluation are shown in Fig. 5.27 and Fig. 5.28. As we can see, the performance trend is similar to the typical workload evaluation under same circumstances, and SharkDB still outperforms the segmented trajectory database.

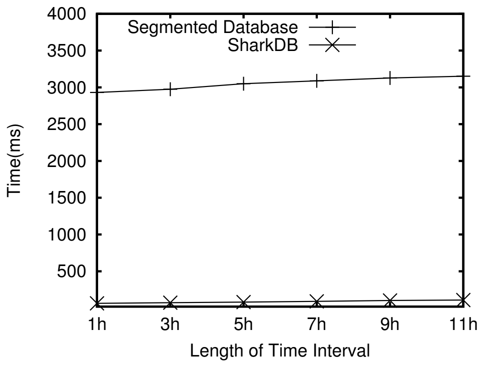


(a) Dataset A

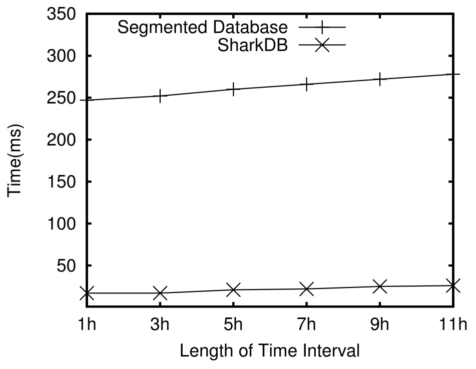


(b) Dataset B

FIGURE 5.26: Operational Workload — Effect of Time Interval for kNN Query

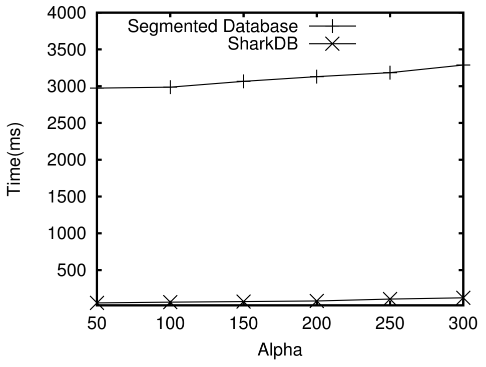


(a) Dataset A

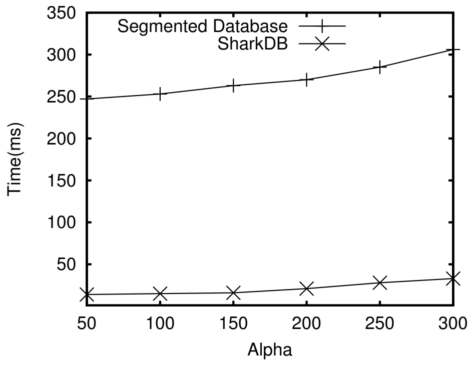


(b) Dataset B

FIGURE 5.27: Operational Workload — Effect of Time Interval for Similarity Search



(a) Dataset A



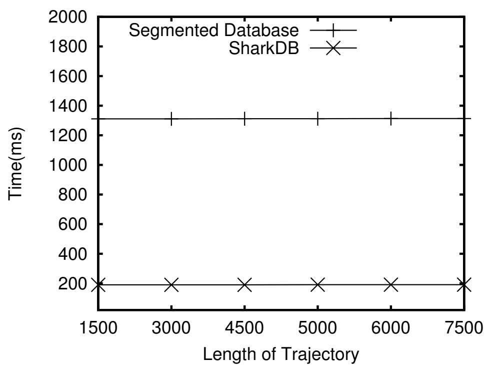
(b) Dataset B

FIGURE 5.28: Operational Workload — Effect of α for Similarity Search

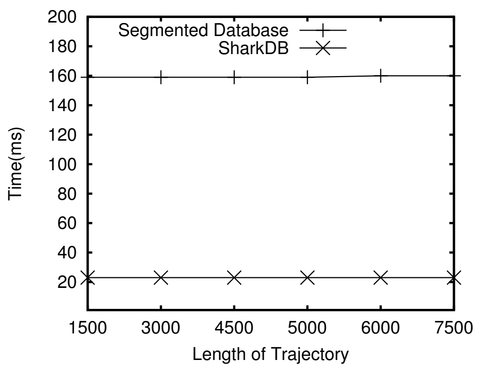
5.3.5 Analytic Workload Evaluation

In this section, the segmented trajectory database and SharkDB are tested under an analytic workload model.

Tune Parameters of Basic Operations First of all, this workload evaluates both systems by changing the length of the trajectory for basic operations. The results are illustrated in Fig. 5.29 and Fig. 5.30. As the expected, the effect on basic operations is minor and can no longer affect the overall performance. This evaluation also provides evidence that the influence of running basic operations in analytic based in-memory trajectory based systems is becoming trivial.

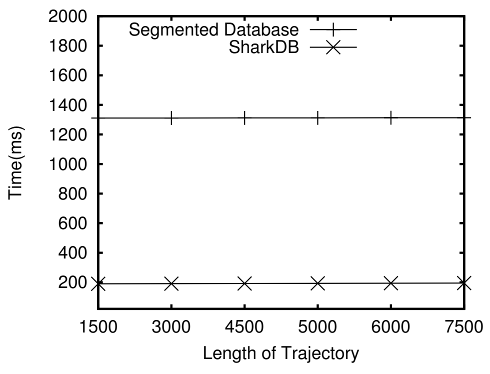


(a) Dataset A

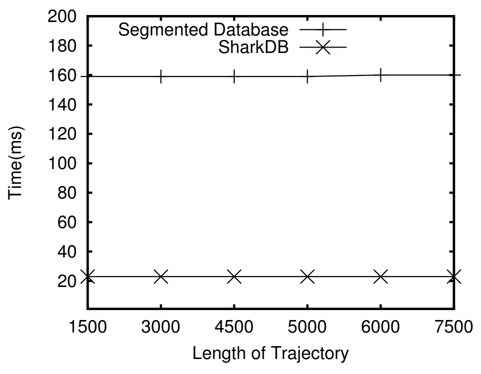


(b) Dataset B

FIGURE 5.29: Analytic Workload — Effect of Trajectory Length for SELECT



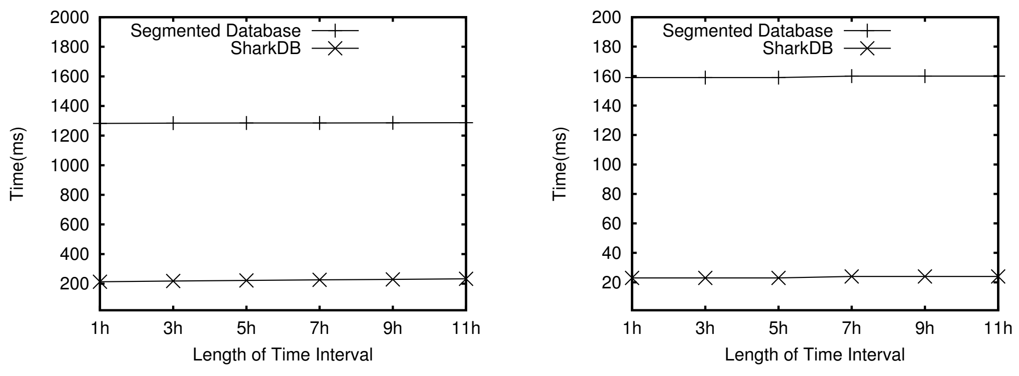
(a) Dataset A



(b) Dataset B

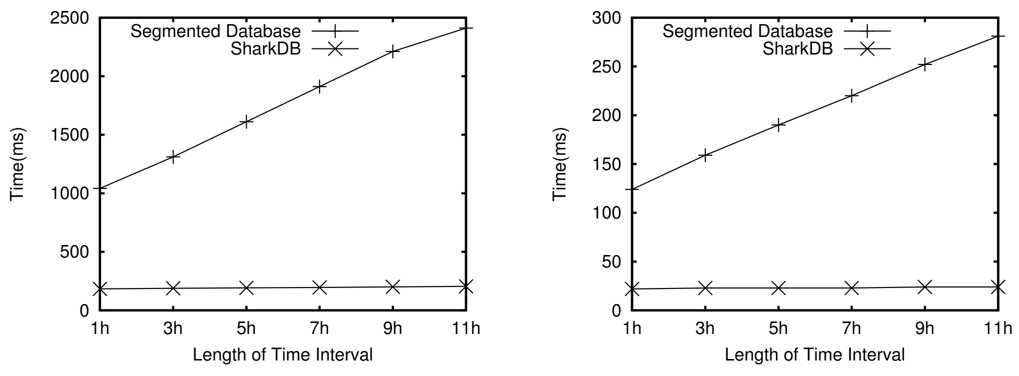
FIGURE 5.30: Analytic Workload — Effect of Trajectory Length for DELETE

Tune Parameters of Advanced Operations The results from running advanced operations on both systems with changing trajectory lengths are demonstrated in Fig. 5.31 and Fig. 5.32. As we can see, the effect on the window query is low. There are two reasons for this. The first is that the window query can be processed in a short time, which is less than one fifth of the time required to run a similarity search. The second is that the window query has only a small weight in this evaluation test. On the other hand, the performance of the kNN query can still affect the overall performance of the segmented trajectory database, since the kNN query is the one of the most time-consuming query for segmented trajectory database. Meanwhile, the SharkDB uses both novel algorithm and parallel computing to increase the performance of the kNN query. Hence the effect of the kNN query is small in this workload evaluation test.



(a) Dataset A (b) Dataset B

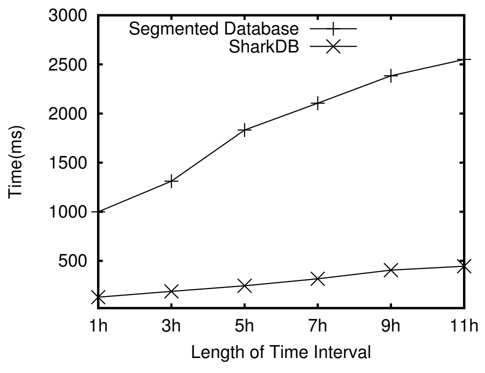
FIGURE 5.31: Analytic Workload — Effect of Time Interval for Window Query



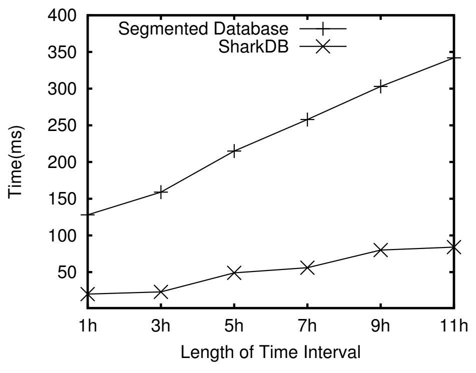
(a) Dataset A (b) Dataset B

FIGURE 5.32: Analytic Workload — Effect of Time Interval for kNN Query

Tune Parameters of Analytic Operation The last workload evaluation is to test the influence of an analytic operation. The results are shown in Fig. 5.33 and Fig. 5.34. It is easy to see that the performance of SharkDB still beats that of the segmented trajectory database. As the similarity search has 70% weight in this workload evaluation, the segmented trajectory database is more sensitive to different parameters than SharkDB. To sum up, SharkDB can significantly speed up the overall performance under a heavy workload.

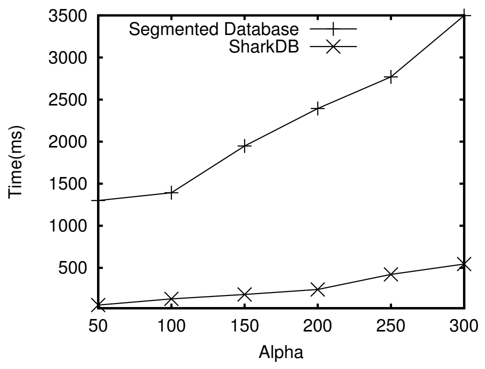


(a) Dataset A

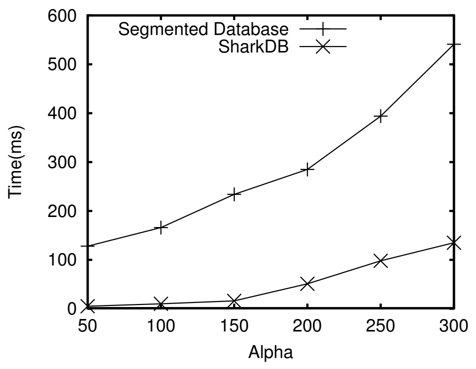


(b) Dataset B

FIGURE 5.33: Analytic Workload — Effect of Time Interval for Similarity Search



(a) Dataset A



(b) Dataset B

FIGURE 5.34: Analytic Workload — Effect of α for Similarity Search

5.4 Summary

In this chapter, extensive experimental results based on both real and synthetic datasets demonstrate that the proposed method significantly outperforms several baseline algorithms with good scalability. Moreover, a new workload benchmark is introduced to compare SharkDB with a traditional database. Test of three workload models (Typical Workload, Operational Workload and Analytic Workload) are conducted to evaluate SharkDB under different usages. In the experiment, the effect of key parameters is discussed and analysed. The comprehensive experiments verify that SharkDB has the best overall performance compared with a traditional data structure under all workload models tested.

Chapter 6

SharkDB Implementation using SAP HANA

6.1 Introduction

Huge amounts of location data that records the motion history of moving objects, known as trajectories, are being generated from different sources such as GPS-enabled devices and smartphones. Analysing trajectory data can help people understand the behavioural patterns of moving objects, and improve the quality of service in applications such as geographical information systems, location-based services, vehicle navigation systems and so on.

The main drawback of managing large scale trajectory data in traditional database management systems (i.e. disk-oriented RMDBSs), is that such database management systems do not provide a special data type or data format to support the trajectory data that are then stored in the traditional table structure. This is because the trajectory is typically modelled as a time-stamped sequence of consecutive locations in a multidimensional space, and the length of each trajectory is variable (i.e. the number of sample points of each trajectory varies from case to case). Naturally, to solve this problem, we can consider encoding each trajectory into one database object, which can be supported in most RMDBSs. Although this format can be easy applied, the efficiency of this data format is low, since the trajectory analytic queries on this data format need to scan the whole table to get the query answers.

In the last decade, the capacity of main memory has rapidly increased, which means that the dataset can be stored and processed within main memory entirely. Memory based storage is much faster than traditional disk-based storage, since the I/O bottleneck has been removed in main memory. In the previous work by [115], SharkDB, an in-memory storage system for massive trajectory data is proposed, and it has achieved promising performance in trajectory query processing. In this chapter, the computational power of SAP HANA, the in-memory column-oriented data analytics platform designed by SAP, is exploited to support efficient query processing for moving object trajectories. The new frame-based data structure design is tailored from a design developed by the previous SharkDB project and makes the trajectory data with variable lengths and sampling rates suitable for the relational databases model in SAP HANA. Extensive experiments based on large-scale real dataset have demonstrated the superior performance of the frame-based design in processing a variety of queries.

6.2 SAP HANA Overview

SAP HANA [85, 86], the newest in-memory based and column-oriented database management system, was developed and implemented by SAP. The goal of SAP HANA is to handle huge amounts of data and the real time complex query processing. Currently, the bottleneck of the traditional disk-oriented database system is the I/O cost, since the I/O performance of hard disk is very limited. To remove this bottleneck, SAP HANA moves all of the data from the hard disk into main memory, which means all of the data are accessed and maintained in the main memory directly. The hard disk in SAP HANA will only be used for data backup and to maintain the log files. Moreover, SAP HANA supports both a row-oriented data structure and a column-oriented data structure, and the column-oriented data structure is recommended by SAP, since the column-oriented data structure can provide better performance for analytic queries.

6.3 Implementation Details

In this section, a frame based data structure is proposed for SAP HANA. Meanwhile, we also implement two common traditional data structures that are used in traditional database systems, which are called key-value format and sample point format.

A key motivation to use the SAP HANA in-memory database system to manage trajectory data is to increase the performance of queries. Therefore, we design a set of common queries, which are divided into two categories, basic operations and advanced operations. The basic operations are normal database operations, which include SELECT, INSERT, DELETE and APPEND. All of the basic operations target a single trajectory by trajectory ID. The advanced operations include two analytic queries, which are window query and kNN query. The window query is used to find all trajectories passing a given region and active during a given period of time. The kNN query is used to find the top-k trajectories that are close to a given point and active during a given period of time. We will discuss the details of the implementation of these queries using three data structures in the followings sections.

6.3.1 Key-Value Format

To store the trajectory data into the traditional data structure, a naive way is to store one trajectory as a single object. Therefore, we can store the whole trajectory dataset into one table with two columns T_{id}, T_{obj}, where T_{id} is the trajectory ID and T_{obj} contains all of the simple points that belong to T_{id}. In order to speed up analytic query processing, for each trajectory object (i.e. each row), we add the auxiliary information that includes the start time, the end time and the MBR of the trajectory. Therefore, in the key-value format, there are five columns, which are $< T_{id}, T_{obj}, T_{st}, T_{et}, T_{mbr} >$.

Basic Operations

Since the key-value format is of the traditional row-based data structure, basic operations (i.e., SELECT, DELETE, INSERT and APPEND) can be done in a very straightforward manner. For INSERT operation, we first encode the trajectory sample points into one object, which is a raw type that is supported by SAP HANA. Next we use normal SQL INSERT command to insert this trajectory directly into table. The SELECT operation is a reverse operation of the INSERT operation, which selects the encoded trajectory object by a given trajectory ID and then decodes trajectory object directly. To implement APPEND operation, we combine both the SELECT and INSERT operation. Therefore, we first select the target trajectory object and encode the trajectory is to be appended as a new raw type object. After this, we add this new object to the end of the target trajectory object. Finally, we update the auxiliary information of the appended trajectory. The DELETE operation is implemented as the normal DELETE operation, which deletes the trajectory object by trajectory ID.

Advanced Operations

In this subsection, the algorithms for processing advanced operations on the key-value format are proposed. A window query will first filter out the trajectories, which are out of the query time interval, by using the trajectories' auxiliary information (i.e. the start and end time of the trajectory). After this, it scans the rest of trajectories. If the MBR of a selected trajectory overlaps the query window, then it decodes this trajectory into its original sample points and find the actual results. For processing the kNN query, MINDIST, which is the minimum distance from query point to MBR, is used as the lower bound. If the MINDIST of a trajectory is larger than the minimum distance of the current k^{th} best trajectory, this trajectory can be safely pruned out, otherwise the algorithm decodes the trajectory and calculates the actual distances from the query point to the sample points in the trajectory.

6.3.2 Sample Point Format

The major drawback of the key-value format is that it needs to decode the whole trajectory into sample points when processing advanced operations, which is very time-consuming. Meanwhile, such operations need a larger amount of memory as the buffer to store the sample points. Therefore, to solve this issue, a simple schema, which is called the sample point format, is used to store the trajectory in the table. In this format, each row stores only one trajectory sample point. The format of each row is $< TID, X, Y, T >$, where TID is the trajectory ID of this sample point, X and Y represent the coordinates of this sample point and T is the timestamp of this sample point.

There are two benefits to the sample point format. Firstly, the data in the sample point format structure does not require decoding when dealing with advanced operations, which is a significant advantage of the sample point format. Another benefit is that it can compress the data of each column in sample point format, and since SAP HANA supports compressing the data in a column-oriented storage model to save space, then the data can be compressed column by column.

Moreover, as discussed previously, the sample point format uses a simple data type, which can achieve a higher compression ratio.

Basic Operations

Both INSERT and APPEND operations on the sample point format are the same, as they only need to convert the trajectory data into sample point format and insert them directly into the table. Due to all of the sample points being stored in one table, for the APPEND operation, we only need to insert the sample points of the new trajectory into the table. The SELECT operation is different from the key-value format, as we need to first select the trajectory sample points by given trajectory ID and then order the sample points by timestamp to recover the whole trajectory. To delete a trajectory, we can directly delete the sample points for a given trajectory ID.

Advanced Operations

For dealing with the window query on the sample point format, the sample points that are located in the given spatial window and time interval are selected, then they are ordered by trajectory ID and timestamp to get the final results. To process the kNN query, the algorithm still uses the MBR information to speed up the query processing. It first tests each MBR with the k^{th} minimum distance, and if a candidate trajectory is found, the algorithm can select the sample points of this trajectory within given time interval. Then, the algorithm calculates the actual distance from the query point to each of the selected sample points to get the final answers.

6.3.3 Frame Format

As the number of columns is limited in SAP HANA, the original frame based data structure needs to be re-designed to suit the traditional table structure. Hence, the frame group column ID is extracted as a single column and combined with the trajectory ID to identify each frame group. In the table, the information for a single frame group is now contained in one row. The information from the I-frame point is stored in two columns as its x value and y value. The rest of columns are used for storing P-frame point information, and similar to the I-frame point, each P-frame point requires two columns, one for δx and another for δy.

When converting the data from the original frame structure to the table based structure, the algorithm first selects a single frame group from frame data structure, then it gets the frame group column id of this frame group and puts it into $FGCID$. Next, the algorithm extracts the trajectory ID and puts it into TID. After this, the x value and y value of the I-frame point are inserted into IFX and IFY, respectively. Then the P-frame points (x and y from each) are inserted in the rest of columns. An example of the table structure is shown in Table. 6.1.

TABLE 6.1: Example of Table based Structure

FGCID	TID	IFX	IFY	PF1PX	PF1PY	...
1	1	$p.x$	$p.y$	$\Delta p.x$	$\Delta p.y$	...
1	2	$p.x$	$p.y$	$\Delta p.x$	$\Delta p.y$	...
1	3	$p.x$	$p.y$	$\Delta p.x$	$\Delta p.y$	...
1	4	$p.x$	$p.y$	$\Delta p.x$	$\Delta p.y$	...

Basic Operations

To insert a new trajectory, the algorithm first allocates the raw trajectory to the frame-based structure which is proposed in the previous section. Then these frame points are split into frame groups and the frame column group ID is determined based on the timestamp of the raw trajectory. Next, each frame group is encoded to generate I-frame point and P-frame points. Finally, it inserts each frame group as a single raw in the table with a trajectory ID and the related frame group column ID.

The SELECT operation for this data structure is straightforward. Since the SELECT is based on trajectory ID, the frame groups are selected by given trajectory ID. In the next step, the results are ordered by frame group column ID.Finally, it decodes both I-frame point and P-frame points to the original sample point, and the timestamp can be recovered by the frame group column ID of each frame group. The DELETE operation is similar to the SELECT operation. It removes frame groups by given trajectory ID from the table directly.

In contrast to the previous format, to expand a trajectory T_{old} by given a trajectory ID TID and a new trajectory T_{new}, it only needs to select the last frame group FG_{last} of T_{old} and decode it to get a trajectory segment. Then, this trajectory segment is connected to T_{new}. After this, it allocates and encodes this trajectory to get the new frame groups with related frame group column ID and TID. Finally, it removes the last frame group FG_{last} and inserts the new frame groups into the database.

Advanced Operations

Given a window query with a rectangle query area and a time interval, this algorithm first calculates the range of frame group columns (i.e. the range of ids of frame group columns) that need to be investigated based on a given time interval. Then, it can select all the frame groups inside the given rectangle. However, the P-frame points in each row have been delta encoded and do not contain actual coordinate information. Hence, for each frame group, we pre-compute its MBR information. If an MBR intersects with a given query window, it puts the related frame group into the candidate set. After that, for each frame group in the candidate set, we decode the P-frames into sample points. Finally, it filters out the sample points that are out of the query area and re-constructs the segment of trajectory based on these sample points.

For the kNN query, we first get the range of frame groups, which is the same as for the window query. Then, for each frame group (i.e. a single row), the algorithm first calculates the MINDIST between the query point and this frame group by using its MBR information to find the candidate. The algorithm keeps updating the candidate set if there is a better candidate found. The kNN query requires a min-heap to store the candidate trajectories, hence a temporary table is created during query processing, which contains two columns: trajectory ID TID, which is the primary key , and the distance $Distance$ from this trajectory to the query point. Meanwhile, the elements in this table are ordered by distance as well. As each trajectory has been encoded as frame groups, $Distance$ records only the shortest distance to the query point from the frame groups that have been investigated with the related trajectory ID. To avoid different frame groups with the same trajectory ID being pushed into the priority queue, a table is used to simulate the min-heap because this can cause such frame groups to be considered as final results, which makes the results incorrect. For example, the distance d_1 from the query point of frame group FG_1 (trajectory ID is T_1) is 2, the distance d_2 of frame group FG_2 (trajectory ID is T_1) is 3 and the distance d_3 of frame group FG_3 (trajectory ID is T_2) is 4. Based on this approach, FG_1 and FG_2 are the two closest to the query point and are pushed into the priority queue, and if $k = 2$, then both of them will be included in the final results. However, they belong to the same trajectory, so the actual results would contain only one trajectory, which does not meet the query requirement. The correct result should return FG_1 and FG_3, since FG_2 belongs to same trajectory as FG_1. Therefore, as discussed previously, using a table and a set trajectory ID as primary key can solve this issue, and when we find a candidate frame group, we first check the trajectory ID of this frame group in the temporary table. If it exists, we update its distance, otherwise we insert this ID and its distance into the table as a new tuple. Moreover, as this table is simulating the priority queue, it will keep the size of table is equal to k rows and remove any other rows.

6.4 Experiments

6.4.1 Experiment Settings

The dataset used is a very large dataset that contains five months of motion history of $20k$ vehicles and the average sampling rate is 30 seconds per sample point. The total number of sample points in this dataset is more than 4 billion and each trajectory contains around $40k$ sample points over a one month duration. All of the data are converted into three table formats respectively, and these tables are all loaded into the SAP HANA system and stored in the main memory. The SAP HANA database system contains one TB of memory and four Intel 10-core CPUs. The algorithms are implemented using SQL/L language.

There are three formats of tables used in the experiment: key-value format, simple point format and frame format. As discussed previously, the experiments are split into two main parts. The first part is the basic operations experiments that includes four basic operations: SELECT, DELETE, INSERT and APPEND. The second part is the advanced operations experiments which includes two trajectory based operations: window query and kNN query. For the basic operations experiments, each query is run 1000 times in three tables to get the average running time. For advanced operations experiments, each query is set with different parameters. There are two parameters for window query, the spatial search window and the query time interval; and kNN query also has two parameters, k and the query time interval. The details of the parameter settings are shown in

Table 6.2. In addition, for the frame format, the number of frames per frame group n is set to 5 and the frame rate is set to 30 seconds, which is same as average sampling rate of the dataset.

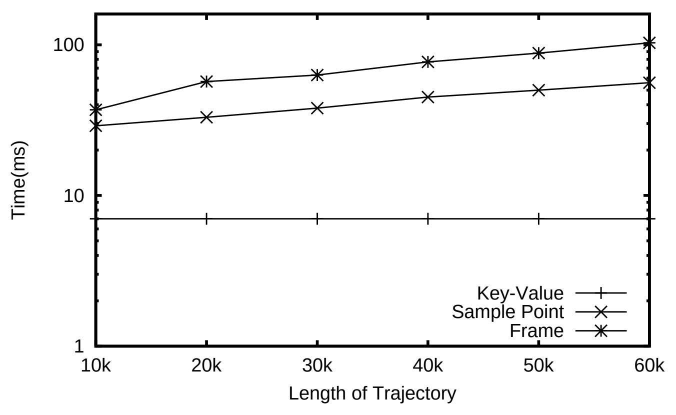
TABLE 6.2: Parameters Setting

Parameter	Default Value	Range
# results k	5	5–30
area of spatial window AS	3%	1%–11%
length of time interval TI	3h	1h – 11h

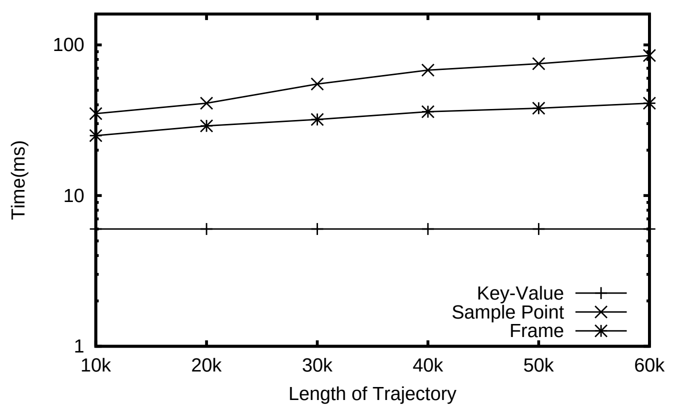
6.4.2 Basic Operations

The basic operation test is made up of two sets of experiments. The first set of experiments includes SELECT by trajectory ID, and DELETE by trajectory ID, which are all operated in main memory. Another set of experiments includes inserting a new trajectory into the database table and appending a trajectory by trajectory ID, both of which need the data to be transferred from other sources (e.g. a PC or a mobile device) to the SAP HANA database system via network. In such operations, the main factor that could affect the performance is the length of the trajectory (i.e number of sample points). Hence, this experiment sets the trajectory length from $10k$ sample points to $60k$ sample points in a trajectory.

Fig. 6.1 shows the results of both the SELECT and DELETE operations for the three data formats. As we can see, the key-value format has the best performance since such basic operations are suitable for a row-oriented store, since trajectory information can be quickly selected by ID and only one row needs to be selected for each query request. Therefore, the performance of the key-value format is very stable over different trajectory lengths. For the SELECT operation, the frame format needs to decode the data to recover the original trajectory, hence it takes longer than for the sample point format. However, there is no need to decode the data in the DELETE operation, so the frame format has better performance than the sample point format, since the number of rows that need to be modified for the frame format is less than for the sample point format. As we can see, even if the frame format structure cannot outperform the traditional row-oriented store database for SELECT and DELETE operations, but the running time is still acceptable and due to it being under one second, can still be considered as real time processing.



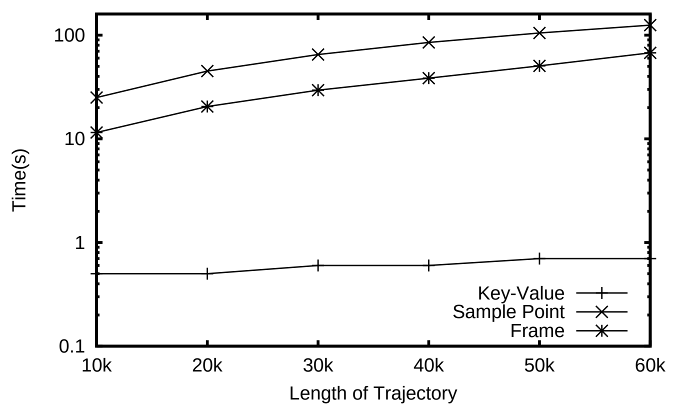
(a) Select Operation



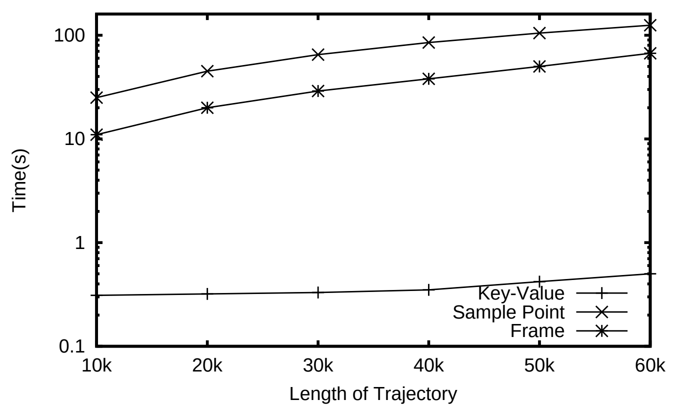
(b) Delete Operation

FIGURE 6.1: Basic Operations(1)

Fig. 6.2 shows the result of INSERT and APPEND operations. As discussed previously, the INSERT operation and the APPEND operation are similar, therefore, the performance results for the INSERT operation and APPEND operation are also similar. This is no doubt that the key-value format has the best performance, since it only needs one INSERT database operation for both INSERT and APPEND operations, which takes less than one second. Both the frame format and the sample point format need much longer running times for these operations, because these two data formats need a large amount of INSERT database operations. On the other hand, the performance of the frame format beats the performance of the sample point format, even if the frame format



(a) Append Operation



(b) Insert Operation

FIGURE 6.2: Basic Operations (2)

needs an extra encoding process when dealing with INSERT and APPEND operations. But the number of INSERT database operations and the costs of network transmission for the frame format are less than for the sample point format. Therefore, this experiment also provides evidence that the cost of network transmission of data and the INSERT database operation is larger than the cost of data encoding, which also shows the advantage of the frame format.

6.4.3 Advanced Operations

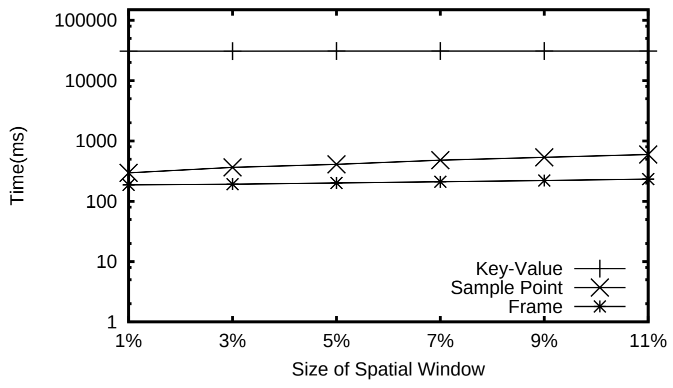
Window Query Firstly the efficiency of the three formats with different spatial window sizes and the default time interval is evaluated. In Fig. 6.3(a), the results show that the key-value format cannot handle the analytic queries, since it is mainly designed for basic operations. For each query, the algorithm can only process the query after decoding the trajectory object. In this case, decoding the whole trajectory is time consuming and will need a large amount of memory as the buffer to store the decoded trajectories.

Upon increasing the size of the searching area, the performance of the sample point format decreases very quickly, since the sample point format only need to a SELECT operation. Therefore, the larger the searching area the more data that needs to be retrieved from the sample point format. The running time for the frame format is slightly increased as the size of window is increases. Because of the larger spatial window size, more frame points need to be decoded. However, there is no doubt that the frame format outperforms the other two data formats. Given that the bandwidth between CPU and memory is now becoming the new bottleneck of in-memory databases, the fact that the frame format can reduce the amount of data that needs to be transferred to CPU to process, means that the frame format can increase the efficiency of in-memory databases.

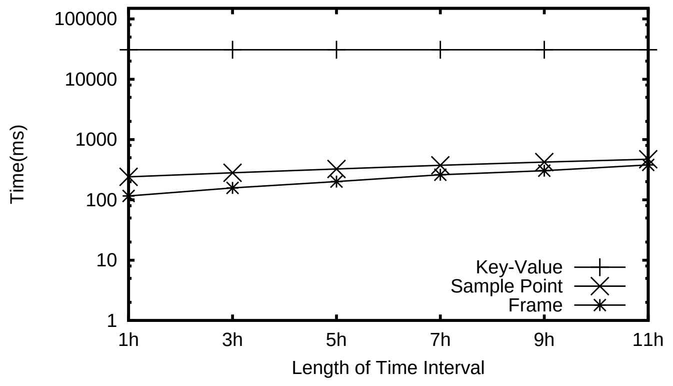
Another set of experiments is also conducted by using different length of time intervals, and the results of these experiments are shown in Fig 6.3(b). Similar to the spatial window size tests, as the length of time interval increases, the SAP HANA database system needs more time to get the answer to the query. Furthermore, we can see that the time interval has less effect than the spatial window size, since this has a high density of spatial data.

kNN Query Fig. 6.4(a) shows the experiment result of the kNN query with different query time intervals. As we can see, the key-value format cannot process the kNN query in real time as well as it can process the window query. The running time of the query is more than 600 seconds, which is not acceptable for query processing in such an advanced in-memory database system. Although, both the sample point format and the frame format can process the query within one second the frame format still outperforms the sample point format. This is because the P-frame points can be pruned by the proposed algorithm efficiently, but for the sample point format, the distance to the query point of each sample point in the selected area must be calculated, and the computational cost is very high. The results also show that the frame format is stable under the load of large trajectory analytic tasks.

The query performance with regard to the value of k for each query is also investigated. The



(a) Different Spatial Window Size



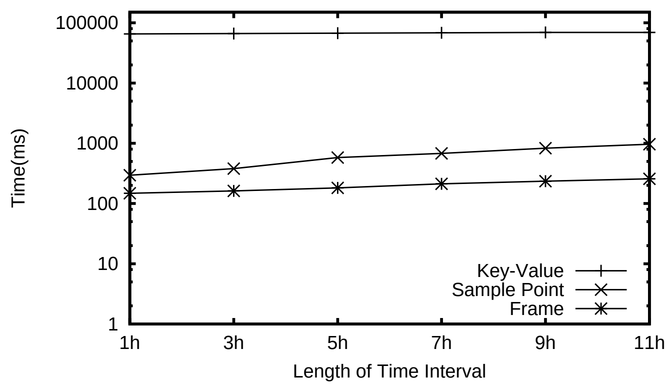
(b) Different Time Interval

FIGURE 6.3: Effect of Window Query

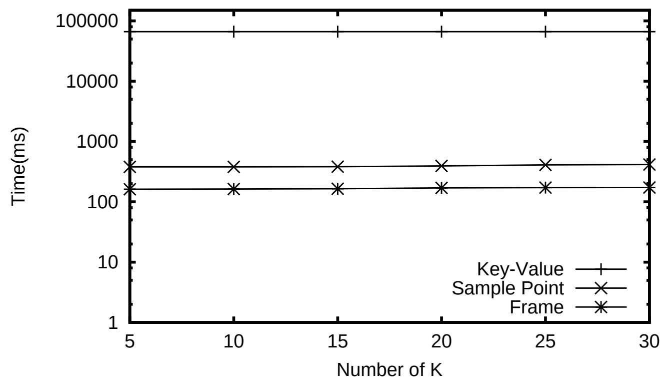
results shown in Fig. 6.4(b) are similar to those of previous experiments. But the difference is that even with a changing query time interval, the results of changing the value of k for the kNN query are quite stable. This is because, this approach uses the temporal table as a priority queue and maintains such small table in SAP HANA which can be done very efficiently.

6.4.4 Compression Ratio

In this subsection, the compression ratios of the three formats are compared. The compression ratio is calculated as the size of the data in the SAP HANA database divided by the size of the original



(a) Different Time Interval



(b) Different k

FIGURE 6.4: Effect of kNN Query

data. The compression ratio for the key-value format is 60.1%, for the sample point format is 55.2% and for the frame format is 39.9%. As we can see the compression ratio of the key-value data structure is not good, since the column of stored trajectory objects is hard to compress. As discussed previously, in the frame format, the P-frame points require less space to record a simple point and it uses an integer data type to instead of a double data type, which can increase the compression ratio. Therefore, the frame format achieves the best compression ratio.

6.5 Summary

In this chapter, the frame-based data structure for in-memory trajectory storage is migrated into SAP HANA. This new prototype takes advantages of both the power of SAP HANA as a fully functional database system and the high efficiency of a frame-based structure for trajectory query processing. The extensive experimental results show that this prototype can consistently achieve better performance than the traditional trajectory storage models.

Chapter 7

Final Remarks

7.1 Conclusions

This thesis presents SharkDB, an in-memory based column-oriented trajectory storage system. Chapter 3 introduces a novel I/P frame data structure that is used in SharkDB. The algorithms to support efficient query processing in SharkDB are presented in Chapter 4; and Chapter 5 proposes a new workload model to evaluate SharkDB under different loading usage. Chapter 6 describes the implementation details to build SharkDB on the SAP HANA in-memory database system.

Firstly, Chapter 3 investigates the drawbacks of a disk-oriented data structure for trajectory data. Then a frame based data structure is proposed to store trajectory data. This frame based data structure is then extended to an I/P frame data structure, which supports compression of trajectory data to significantly reduce the size of the data. Finally, the I/P frame data structure is improved to a cache-aware I/P frame data structure, which is optimised for CPU caching.

Chapter 4 classifies the supported query types of SharkDB into three categories: basic operations, advanced operations and analytic operations. In the basic operations category, SELECT, DELETE, UPDATE and APPEND operations are implemented, which are the traditional operations for transaction database systems. To improve the performance of these basic operations on the I/P frame data structure, a parallel computing based approach is proposed to increase the processing speed. In the category of advanced operations, to efficiently answer both the window query and the kNN query, a novel hierarchical I/P frame structure is proposed to increase the pruning efficiency during the initial query processing. Moreover, parallel processing is also deployed to deal with P-frame decoding. In the analytic operation category, the problem of the trajectory similarity search is investigated and a three-phase processing algorithm is proposed. Firstly, the MBR information from each frame group is used for pruning. In the next phase, the algorithm decodes the candidate frame groups in parallel. In final phase, the KMP algorithm is invoked to assist the trajectory similarity search. Extensive experimental results based on both real and synthetic datasets demonstrate that the proposed algorithms significantly outperform several baseline algorithms with good scalability.

In Chapter 5, extensive experiments are firstly conducted to evaluate the proposed approaches, and then a new workload model is introduced to evaluate SharkDB. A tiered set of common operations are defined for workload evaluations, with the basic operations that cover simple database operations, the advanced operations that cover complex common queries on trajectory data, and the analytic operations that cover trajectory similarity searches. In the workload model, different weights are set for different queries to simulate three types of workload usages: a typical workload, an operational workload and an analytic workload. The comprehensive experiments verify that SharkDB has the best overall performance compared with traditional data structure under all types of workload models tested.

In Chapter 6, the whole I/P frame structure is migrated and implemented into the SAP HANA database system. To achieve this goal, the I/P frame structure is redesigned to allow the whole structure to be stored in a normal table in the SAP HANA database system. The extensive experimental results show the newly designed frame formate for trajectory data can achieve better performance and better compression ratios than the traditional trajectory data formats.

7.2 Future Work

This section discusses several possible future directions for the SharkDB development.

7.2.1 An I/P frame Structure Extension for Hybrid Trajectory Datasets

In Chapter 3, a cache-aware I/P frame structure is presented for storing compressed trajectory data in the main memory. In future work, an extended I/P frame structure could be implemented to support storing hybrid trajectory datasets, which contain a different type of trajectory data. This is because the data will be collected from different sources in the age of big data. For example, trajectory data collected from vehicle contains extra information such as speed and direction. And trajectory data uploaded by users may contain extra check-in or semantic information. This additional information can be very helpful or even necessary for analytic queries. The main challenge comes from the difficulty of managing the unstructure data using an I/P frame structure and to maintain the performance and compression ratio at the same time.

7.2.2 A Distributed SharkDB Implementation on Apache Spark

In the age of big data, the volume of data being generated is increasing very rapidly. This will cause the main memory of a single machine main-memory could be filled up in a short time. However, it is not a good idea to store the historic data on the hard-disk, since many real applications may require trajectory data over a long period for analysing, such as urban planing and traffic trend analysis applications. Motivated by this, it is necessary to implement a distributed SharkDB version. There are two advantages to a distributed version, the first one is that the storage space capacity of a distributed system can be easily extended by adding a new node. Another advantage is that the computational power of a distributed system is greater than that of a single big machine. Hadoop is one of successful distributed frameworks that is based on MapReduce. Apache Spark is a Hadopp-like distributed framework with in-memory computing technology. As SharkDB is an in-memory based storage, Apache Spark is a good potential platform to which to migrate SharkDB. There are two main challenges to this future work, the first is the load-balancing of the computational nodes of whole cluster to avoid overload in one computational node which would reduce the performance. The second challenge is to minimise the communication cost between each computational node, since the network transmission of data is far slower than in memory or in hard-disk. To address these two challenges, a carefully designed algorithm is required. This algorithm will need to divide a query into sub-queries and ensure that none of the nodes are overloaded and that most of the data can be read locally.

7.2.3 Interface and APIs Design for SharkDB on SAP HANA

In Chapter 6, SharkDB was migrated to and implemented into SAP HANA. In particular, in this thesis, SharkDB is implemented as a back-end storage and query system on SAP HANA. One possible extension of SharkDB is about visualisation, which is very important for end users as it can help the user to understand the results of query. Hence, it is better to provide a web-based interface for users to input a query and get the results visually. A demo version of interface design is deployed, which supports the window query and the kNN. In future work, a schedule is planned to support more queries via this interface as well as to develop a new interface that allows users to control the parameters of SharkDB. Furthermore, a plan to pack SharkDB as a library in SAP HANA is also in the schedule, which will allow third party applications to connect to SharkDB for trajectory analysing tasks via APIs.

References

[1] D. J. Abel. siro-dbms: a database tool-kit for geographical information systems. *International Journal of Geographical Information System*, 3(2):103–116, 1989.

[2] A. V. Aho and J. E. Hopcroft. *Design & Analysis of Computer Algorithms*. Pearson Education India, 1974.

[3] J. Baulier, P. Bohannon, S. Gogate, C. Gupta, and S. Haldar. DataBlitz storage manager: main-memory database performance for critical applications. In *SIGMOD*, pages 519–520, 1999.

[4] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The r*-tree: An efficient and robust access method for points and rectangles. In *SIGMOD*, pages 322–331, 1990.

[5] R. Benetis, C. Jensen, G. Kariauskas, and S. altenis. Nearest and reverse nearest neighbor queries for moving objects. *The VLDB Journal*, 15(3):229–249, 2006.

[6] R. Benetis, C. S. Jensen, G. Karčiauskas, and S. Šaltenis. Nearest neighbor and reverse nearest neighbor queries for moving objects. In *IDEAS*, pages 44–53, 2002.

[7] S. Berchtold, B. Ertl, D. Keim, H.-P. Kriegel, T. Seidl, et al. Fast nearest neighbor search in high-dimensional space. In *ICDE*, pages 209–218, 1998.

[8] R. R. Berman and M. Stonebraker. geo-ouel: a system for the manipulation and display of geographic data. In *ACM SIGGRAPH Computer Graphics*, pages 186–191, 1977.

[9] P. Bernstein, M. Brodie, S. Ceri, D. DeWitt, M. Franklin, H. Garcia-Molina, J. Gray, J. Held, J. Hellerstein, H. V. Jagadish, M. Lesk, D. Maier, J. Naughton, H. Pirahesh, M. Stonebraker, and J. Ullman. The asilomar report on database research. *SIGMOD Rec.*, 27(4):74–80, 1988.

[10] C. Binnig, S. Hildenbrand, and F. Färber. Dictionary-based order-preserving string compression for main memory column stores. In *SIGMOD*, pages 283–296, 2009.

[11] S. Blanas, Y. Li, and J. M. Patel. Design and evaluation of main memory hash join algorithms for multi-core CPUs. In *SIGMOD*, pages 37–48, 2011.

[12] P. A. Boncz, M. Zukowski, and N. Nes. MonetDB/X100: Hyper-pipelining query execution. In *CIDR*, pages 225–237, 2005.

[13] V. Botea, D. Mallett, M. A. Nascimento, and J. Sander. PIST: an efficient and practical indexing technique for historical spatio-temporal point data. *GeoInformatica*, 12(2):143–168, 2008.

[14] S. Brakatsoulas, D. Pfoser, R. Salas, and C. Wenk. On map-matching vehicle tracking data. In *PVLDB*, pages 853–864, 2005.

[15] T. Brinkhoff, H.-P. Kriegel, and B. Seeger. Efficient processing of spatial joins using r-trees. In *SIGMOD*, pages 237–246, 1993.

[16] C. Buragohain, N. Shrivastava, and S. Suri. Space efficient streaming algorithms for the maximum error histogram. In *ICDE*, pages 1026–1035, 2007.

[17] V. P. Chakka, A. C. Everspaugh, and J. M. Patel. Indexing large trajectory data sets with SETI. In *CIDR*, 2003.

[18] N.-S. Chang and K. S. Fu. *A relational database system for images*. Springer, 1980.

[19] L. Chen and R. Ng. On the marriage of lp-norms and edit distance. In *PVLDB*, pages 792–803, 2004.

[20] L. Chen, M. Özsu, and V. Oria. Robust and fast similarity search for moving object trajectories. In *SIGMOD*, pages 491–502, 2005.

[21] M. Chen, M. Xu, and P. Franti. A fast O(N) multiresolution polygonal approximation algorithm for gps trajectory simplification. *TIP*, 21(5):2770–2785, 2012.

[22] Y. Chen, K. Jiang, Y. Zheng, C. Li, and N. Yu. Trajectory simplification method for location-based social networking services. In *LBSN*, pages 33–40, 2009.

[23] Z. Chen, H. T. Shen, and X. Zhou. Discovering popular routes from trajectories. In *ICDE*, pages 900–911, 2011.

[24] Z. Chen, H. T. Shen, X. Zhou, Y. Zheng, and X. Xie. Searching trajectories by locations: an efficiency study. In *SIGMOD*, pages 255–266, 2010.

[25] K.-L. Chung, W.-M. Yan, and W.-Y. Chen. Efficient algorithms for 3-d polygonal approximation based on LISE criterion. *Pattern Recognition*, 35(11):2539 – 2548, 2002.

[26] F. Clarke. Optimal solutions to differential inclusions. *Journal of Optimization Theory and Applications*, 19(3):469–478, 1976.

[27] D. Comer. Ubiquitous b-tree. *ACM Comput. Surv.*, 11(2):121–137, June 1979.

[28] P. Cudre-Mauroux, E. Wu, and S. Madden. Trajstore: An adaptive storage system for very large trajectory data sets. In *ICDE*, pages 109–120, 2010.

[29] D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2):112–122, Oct. 1973.

[30] C. Düntgen, T. Behr, and R. H. Güting. Berlinmod: a benchmark for moving object databases. *VLDBJ*, 18(6):1335–1368, 2009.

[31] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS*, pages 102–113, 2001.

[32] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. *SIGMOD Rec.*, 23(2):419–429, May 1994.

[33] M. Gargano, E. Nardelli, and M. Talamo. Abstract data types for the logical modeling of complex data. *Information Systems*, 16(6):565–583, 1991.

[34] L. Gruenwald and M. H. Eich. MMDB reload algorithms. In *SIGMOD*, pages 397–405, 1991.

[35] R. H. Güting. *Geo-relational algebra: A model and query language for geometric database systems*. Springer, 1988.

[36] R. H. Güting. An introduction to spatial database systems. *The VLDB Journal*, 3(4):357–399, 1994.

[37] R. H. Güting and M. Schneider. Realm-based spatial data types: the rose algebra. *VLDBJ*, 4(2):243–286, 1995.

[38] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *SIGMOD*, pages 47–57, 1984.

[39] S. Héman, M. Zukowski, N. J. Nes, L. Sidirourgos, and P. Boncz. Positional update handling in column stores. In *SIGMOD*, pages 543–554, 2010.

[40] J. Hershberger and J. Snoeyink. Cartographic line simplification and polygon csg formulae in o(n log* n) time. In *COMPUTATIONAL GEOMETRY THEORY & APPLICATIONS*, pages 93–103. Springer-Verlag, 1998.

[41] G. R. Hjaltason and H. Samet. Distance browsing in spatial databases. *TODS*, 24(2):265–318, 1999.

[42] Y.-W. Huang, N. Jing, E. Rundensteiner, et al. A cost model for estimating the performance of spatial joins using r-trees. In *SSDBM*, pages 30–38, 1997.

[43] Y.-W. Huang, N. Jing, and E. A. Rundensteiner. Spatial joins using r-trees: Breadth-first traversal with global optimizations. In *PVLDB*, volume 97, pages 25–29, 1997.

[44] T. Ichiye and M. Karplus. Collective motions in proteins: a covariance analysis of atomic fluctuations in molecular dynamics and normal mode simulations. *Proteins: Structure, Function, and Bioinformatics*, 11:205–217, 1991.

[45] S. Idreos, S. Manegold, H. Kuno, and G. Graefe. Merging what's cracked, cracking what's merged: adaptive indexing in main-memory column-stores. *Proc. VLDB Endow.*, 4(9):586–597, June 2011.

[46] M. G. Ivanova, M. L. Kersten, N. J. Nes, and R. A. Gonçalves. An architecture for recycling intermediates in a column-store. *TODS*, 35(4):24:1–24:43, 2010.

[47] R. Jonkery, G. De Leve, J. Van Der Velde, and A. Volgenant. Rounding symmetric traveling salesman problems with an asymmetric assignment problem. *Operations Research*, pages 623–627, 1980.

[48] J. Kang, M. Mokbel, S. Shekhar, T. Xia, and D. Zhang. Continuous evaluation of monochromatic and bichromatic reverse nearest neighbors. In *ICDE*, pages 806–815, 2007.

[49] J. Kearney and S. Hansen. Stream editing for animation. Technical report, DTIC Document, 1990.

[50] G. Kellaris, N. Pelekis, and Y. Theodoridis. Map-matched trajectory compression. *Journal of Systems and Software*, 86(6):1566 – 1579, 2013.

[51] E. Keogh and M. Pazzani. Scaling up dynamic time warping for datamining applications. In *SIGKDD*, pages 285–289, 2000.

[52] C. Kim, T. Kaldewey, V. W. Lee, E. Sedlar, A. D. Nguyen, N. Satish, J. Chhugani, A. Di Blas, and P. Dubey. Sort vs. hash revisited: fast join implementation on modern multi-core CPUs. *VLDBJ*, 2(2):1378–1389, 2009.

[53] D. E. Knuth, J. H. Morris, Jr, and V. R. Pratt. Fast pattern matching in strings. *SIAM journal on computing*, 6(2):323–350, 1977.

[54] G. Kollios, D. Gunopulos, and V. J. Tsotras. Nearest neighbor queries in a mobile environment. In *STDBM*, pages 119–134, 1999.

[55] C. P. Kolovson and M. Stonebraker. Segment indexes: Dynamic indexing techniques for multi-dimensional interval data. In *SIGMOD*, pages 138–147, 1991.

[56] F. Korn and S. Muthukrishnan. Influence sets based on reverse nearest neighbor queries. In *ACM SIGMOD Record*, volume 29, pages 201–212. ACM, 2000.

[57] J. Krueger, C. Kim, M. Grund, N. Satish, D. Schwalb, J. Chhugani, H. Plattner, P. Dubey, and A. Zeier. Fast updates on read-optimized databases using multi-core CPUs. *PVLDB*, 5(1):61–72, 2011.

[58] J. Kruskal. An overview of sequence comparison: Time warps, string edits, and macromolecules. *SIAM review*, pages 201–237, 1983.

[59] T. J. Lehman and M. J. Carey. A study of index structures for main memory database management systems. In *PVLDB*, volume 294, 1986.

[60] C. Lemke, K.-U. Sattler, F. Faerber, and A. Zeier. Speeding up queries in column stores. In *DaWaK*, pages 117–129, 2010.

[61] X. Lian and L. Chen. Probabilistic group nearest neighbor queries in uncertain databases. *TKDE*, 20(6):809 –824, june 2008.

[62] K.-I. Lin, M. Nolen, and C. Yang. Applying bulk insertion techniques for dynamic reverse nearest neighbor problems. In *IDEAS*, pages 290–297, 2003.

[63] D. B. Lomet and B. Salzberg. The hb-tree: a multiattribute indexing method with good guaranteed performance. *TODS*, 15(4):625–658, Dec. 1990.

[64] Y. Lou, C. Zhang, Y. Zheng, X. Xie, W. Wang, and Y. Huang. Map-matching for low-sampling-rate gps trajectories. In *SIGSPATIAL*, pages 352–361, 2009.

[65] N. Mamoulis and D. Papadias. Multiway spatial joins. *TODS*, 26(4):424–475, 2001.

[66] S. Manegold, P. Boncz, and M. Kersten. Optimizing main-memory join on modern hardware. *TKDE*, 14(4):709–730, 2002.

[67] N. Meratnia and R. By. Spatiotemporal compression techniques for moving point objects. In *EDBT*, pages 765–782, 2004.

[68] J. Muckell, J.-H. Hwang, V. Patil, C. T. Lawson, F. Ping, and S. S. Ravi. SQUISH: an online approach for gps trajectory compression. In *COM.Geo*, pages 13:1–13:8, 2011.

[69] C. Myers, L. Rabiner, and A. Rosenberg. Performance tradeoffs in dynamic time warping algorithms for isolated word recognition. *TASSP*, 28(6):623–635, 1980.

[70] M. A. Nascimento and J. R. O. Silva. Towards historical r-trees. In *SAC*, pages 235–240, 1998.

[71] J. Nievergelt, H. Hinterberger, and K. C. Sevcik. The grid file: An adaptable, symmetric multikey file structure. *TODS*, 9(1):38–71, 1984.

[72] S. Nutanong, E. Tanin, and R. Zhang. Incremental evaluation of visible nearest neighbor queries. *TKDE*, 22(5):665 –681, may 2010.

[73] M. Ostendorf and S. Roukos. A stochastic segment model for phoneme-based continuous speech recognition. *TASSP*, 37:1857–1869, 1989.

[74] B.-U. Pagel, H.-W. Six, H. Toben, and P. Widmayer. Towards an analysis of range query performance in spatial data structures. In *SIGMOD*, pages 214–221, 1993.

[75] D. Papadias, N. Mamoulis, and Y. Theodoridis. Processing and optimization of multiway spatial joins using r-trees. In *SIGMOD*, pages 44–55, 1999.

[76] D. Papadias, Q. Shen, Y. Tao, and K. Mouratidis. Group nearest neighbor queries. In *ICDE*, pages 301 – 312, 2004.

[77] D. Papadias, Y. Tao, K. Mouratidis, and C. K. Hui. Aggregate nearest neighbor queries in spatial databases. *ACM Trans. Database Syst.*, 30(2):529–576, 2005.

[78] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao. Query processing in spatial network databases. In *PVLDB*, VLDB '03, pages 802–813, 2003.

[79] D. A. Patterson, G. Gibson, and R. H. Katz. A case for redundant arrays of inexpensive disks (RAID). In *SIGMOD*, pages 109–116, 1988.

[80] J.-C. Perez and E. Vidal. Optimum polygonal approximation of digitized curves. *Pattern Recognition Letters*, 15(8):743 – 750, 1994.

[81] P. Pfeifer and S. Deutsch. A three-stage iterative procedure for space-time modeling. *Technometrics*, pages 35–47, 1980.

[82] D. Pfoser, C. S. Jensen, Y. Theodoridis, et al. Novel approaches to the indexing of moving object trajectories. In *PVLDB*, pages 395–406, 2000.

[83] T. Picton, M. Hunt, R. Mowrey, R. Rodriguez, and J. Maru. Evaluation of brain-stem auditory evoked potentials using dynamic time warping. *Electroencephalography and Clinical Neurophysiology/Evoked Potentials Section*, 71(3):212–225, 1988.

[84] A. Pikaz and I. h. Dinstein. An algorithm for polygonal approximation based on iterative point elimination. *Pattern Recognition Letters*, 16(6):557 – 563, 1995.

[85] H. Plattner. A common database approach for OLTP and OLAP using an in-memory column database. In *SIGMOD*, pages 1–2, 2009.

[86] H. Plattner. SanssouciDb: An in-memory database for processing enterprise workloads. In *BTW*, volume 20, pages 2–21, 2011.

[87] M. Potamias, K. Patroumpas, and T. Sellis. Sampling trajectory streams with spatiotemporal criteria. In *Scientific and Statistical Database Management, 2006. 18th International Conference on*, pages 275–284, 2006.

[88] M. Priestley. State-dependent models: A general approach to non-linear time series analysis. *Journal of Time Series Analysis*, 1(1):47–71, 1980.

[89] J. Rao and K. A. Ross. Making B+- trees cache conscious in main memory. In *SIGMOD*, pages 475–486, 2000.

[90] K. Raptopoulou, A. Papadopoulos, and Y. Manolopoulos. Fast nearest-neighbor query processing in moving-object databases. *GeoInformatica*, 7(2):113–137, 2003.

[91] S. Rasetic, J. Sander, J. Elding, and M. A. Nascimento. A trajectory splitting model for efficient spatio-temporal indexing. In *PVLDB*, pages 934–945, 2005.

[92] J. Richalet, A. Rault, J. Testud, and J. Papon. Model predictive heuristic control:: Applications to industrial processes. *Automatica*, 14(5):413–428, 1978.

[93] M. Robinson. The temporal development of collision cascades in the binary-collision approximation. *Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms*, 48(1-4):408–413, 1990.

[94] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *ACM sigmod record*, pages 71–79, 1995.

[95] Y. Sakurai, M. Yoshikawa, and C. Faloutsos. Ftw: fast similarity search under the time warping distance. In *SIDMOD*, pages 326–337, 2005.

[96] H. Samet, J. Sankaranarayanan, and H. Alborzi. Scalable network distance browsing in spatial databases. In *SIGMOD*, pages 43–54, 2008.

[97] H. Samet and R. E. Webber. Storing a collection of polygons using quadtrees. *ACM Trans. Graph.*, 4(3):182–222, 1985.

[98] A. Sanderson and A. Wong. Pattern trajectory analysis of nonstationary multivariate data. *TSMC*, 10:384–392, 1980.

[99] J. Sankaranarayanan, H. Alborzi, and H. Samet. Efficient query processing on spatial networks. In *Proceedings of the 13th annual ACM international workshop on Geographic information systems*, pages 200–209, 2005.

[100] S. Shang, B. Yuan, K. Deng, K. Xie, K. Zheng, and X. Zhou. PNN query processing on compressed trajectories. *GeoInformatica*, 16:467–496, 2012.

[101] C. S. J. Simonas Saltenis, S. T. Leutenegger, and M. A. Lopez. Indexing the positions of continuously moving objects. In *SIGMOD*, pages 331–342, 2000.

[102] Z. Song and N. Roussopoulos. K-nearest neighbor search for moving query point. In *SSTD*, pages 79–96, 2001.

[103] F. Soong and A. Rosenberg. On the use of instantaneous and transitional spectral information in speaker recognition. *TASSP*, 36(6):871–879, 1988.

[104] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. O'Neil, P. O'Neil, A. Rasin, N. Tran, and S. Zdonik. C-store: a column-oriented DBMS. In *VLDB*, pages 553–564, 2005.

[105] H. Su, K. Zheng, J. Huang, H. Jeung, L. Chen, and X. Zhou. Crowdplanner: A crowd-based route recommendation system. In *ICDE*, pages 1144–1155, 2014.

[106] H. Su, K. Zheng, H. Wang, J. Huang, and X. Zhou. Calibrating trajectory data for similarity-based analysis. In *SIGMOD*, pages 833–844, 2013.

[107] F. Takens. Motion under the influence of a strong constraining force. *Global theory of dynamical systems*, pages 425–445, 1980.

[108] Y. Tao and D. Papadias. Time-parameterized queries in spatio-temporal databases. In *SIGMOD*, pages 334–345, 2002.

[109] Y. Tao, D. Papadias, and X. Lian. Reverse knn search in arbitrary dimensionality. In *PVLDB*, pages 744–755, 2004.

[110] Y. Tao, D. Papadias, and Q. Shen. Continuous nearest neighbor search. In *PVLDB*, VLDB '02, pages 287–298, 2002.

[111] Y. Tao, D. Papadias, and J. Sun. The TPR*-tree: an optimized spatio-temporal access method for predictive queries. In *PVLDB*, pages 790–801, 2003.

[112] Y. Tao, C. Sheng, and J. Pei. On k-skip shortest paths. In *SIGMOD*, pages 421–432, 2011.

[113] Y. Theodoridis and T. Sellis. A model for the prediction of r-tree performance. In *SIGMOD*, pages 161–171, 1996.

[114] Y. Theodoridis, E. Stefanakis, and T. Sellis. Cost models for join queries in spatial databases. In *ICDE*, pages 476–483, 1998.

[115] H. Wang, K. Zheng, J. Xu, B. Zheng, X. Zhou, and S. Sadiq. SharkDB: An in-memory column-oriented trajectory storage. In *CIKM*, pages 1409–1418, 2014.

[116] T. C. Waugh and R. G. Healey. The geo view design a relational data base approach to geographical data handling. *International Journal of Geographical Information System*, 1(2):101–118, 1987.

[117] W. Wu, F. Yang, C.-Y. Chan, and K.-L. Tan. Finch: Evaluating reverse k-nearest-neighbor queries on location data. *PVLDB*, 1(1):1056–1067, 2008.

[118] T. Xia and D. Zhang. Continuous reverse nearest neighbor monitoring. In *ICDE*, pages 77–77, 2006.

[119] K. Xie, K. Deng, S. Shang, X. Zhou, and K. Zheng. Finding alternative shortest paths in spatial networks. *TODS*, 37(4):29, 2012.

[120] K. Xie, K. Deng, and X. Zhou. From trajectories to activities: a spatio-temporal join approach. In *Proceedings of the 2009 International Workshop on Location Based Social Networks*, pages 25–32, 2009.

[121] X. Xiong, M. F. Mokbel, and W. G. Aref. Sea-cnn: Scalable processing of continuous k-nearest neighbor queries in spatio-temporal databases. In *ICDE*, pages 643–654, 2005.

[122] H. Xu, Z. Li, Y. Lu, K. Deng, and X. Zhou. Group visible nearest neighbor queries in spatial databases. In *WAIM*, volume 6184, pages 333–344, 2010.

[123] C. Yang and K.-I. Lin. An index structure for efficient reverse nearest neighbor queries. In *ICDE*, pages 485–492, 2001.

[124] B. Yi, H. Jagadish, and C. Faloutsos. Efficient retrieval of similar time sequences under time warping. In *ICDE*, pages 201–208, 1998.

[125] M. Yiu, N. Mamoulis, and D. Papadias. Aggregate nearest neighbor queries in road networks. *TKDE*, 17(6):820 – 833, 2005.

[126] M. L. Yiu, D. Papadias, N. Mamoulis, and Y. Tao. Reverse nearest neighbors in large graphs. *TKDE*, 18(4):540–553, 2006.

[127] X. Yu, K. Pu, and N. Koudas. Monitoring k-nearest neighbor queries over moving objects. In *ICDE*, pages 631–642, 2005.

[128] N. J. Yuan, Y. Zheng, X. Xie, Y. Wang, K. Zheng, and H. Xiong. Discovering urban functional zones using latent activity trajectories. *TKDE*, 27(3):712–725, 2015.

[129] F. Zhang, N. J. Yuan, D. Wilkie, Y. Zheng, and X. Xie. Sensing the pulse of urban refueling behavior: A perspective from taxi mobility. *TIST*, 6(3):37, 2015.

[130] J. Zhang, M. Zhu, D. Papadias, Y. Tao, and D. L. Lee. Location-based spatial queries. In *SIGMOD*, pages 443–454, 2003.

[131] B. Zheng and D. L. Lee. Semantic caching in location-dependent query processing. In *SSTD*, pages 97–116, 2001.

[132] B. Zheng, N. J. Yuan, K. Zheng, X. Xie, S. Sadiq, and X. Zhou. Approximate keyword search in semantic trajectory database. In *ICDE*, pages 975–986, 2015.

[133] K. Zheng, P. C. Fung, and X. Zhou. K-nearest neighbor search for fuzzy objects. In *SIGMOD*, pages 699–710, 2010.

[134] K. Zheng, Z. Huang, A. Zhou, and X. Zhou. Discovering the most influential sites over uncertain data: A rank-based approach. *TKDE*, 24(12):2156–2169, 2012.

[135] K. Zheng, S. Shang, N. J. Yuan, and Y. Yang. Towards efficient search for activity trajectories. In *ICDE*, 2013.

[136] K. Zheng, G. Trajcevski, X. Zhou, and P. Scheuermann. Probabilistic range queries for uncertain trajectories on road networks. In *EDBT*, pages 283–294, 2011.

[137] K. Zheng, Y. Zheng, X. Xie, and X. Zhou. Reducing uncertainty of low-sampling-rate trajectories. In *ICDE*, pages 1144 –1155, 2012.

[138] K. Zheng, Y. Zheng, N. J. Yuan, and S. Shang. On discovery of gathering patterns from trajectories. In *ICDE*, pages 242–253, 2013.

[139] K. Zheng, X. Zhou, P. Fung, and K. Xie. Spatial query processing for fuzzy objects. *The VLDB Journal*, pages 1–23, 2011.

[140] Y. Zheng, L. Zhang, X. Xie, and W. Ma. Mining interesting locations and travel sequences from gps trajectories. In *WWW*, pages 791–800, 2009.