A System for Trajectory Data Management and Analysis

Johann Bornholdt[1]

[1]supervised by Michael Grossniklaus,
University of Konstanz, Germany

Abstract

In a world where the volume of available trajectory data is constantly increasing, the need for efficient storage and processing of such data has motivated the development of numerous applications and analysis methods. Different analysis methods require the trajectory data to be in specific data representations. At the same time, in order to efficiently process *complex trajectory queries*, multiple representations are necessary. However, current systems focus on a single data representation for trajectories. In this paper, we present the research outline of a Ph.D. in its early stages, where we plan to propose an algebra and a corresponding set of logical and physical operators to support *complex trajectory queries* by considering different trajectory data representations. Furthermore, we propose to design a demonstrator system as proof of concept.

Keywords

trajectory database systems, mobility analysis, spatiotemporal data management

1. Introduction

Due to technological advances in the last decades that led to the ubiquity of cheap GPS sensors, there is an abundance of trajectory data available. As a result, the efficient processing and analysis of trajectory data have attracted significant interest[1, 2]. Numerous applications that require the efficient processing of trajectory data have also been developed. These applications range from optimizing a fleet of taxi cabs based on past trips [3], to studying the global migration of animals based on tracking data from space [4]. In the Centre for the Advanced Study of Collective Behaviour[1], the excellence cluster in which the presented research is situated, we are building the so-called *Imaging Hangar*, which enables us to study small animal collectives in a controlled environment using trajectory data obtained from video image analysis [5].

When processing or analyzing trajectories, the trajectory's representation can significantly impact which operations can be applied or how efficiently they can be executed. For example, geographic operations are best processed on a trajectory represented as a spatial object, while others, e.g., temporal operations, need trajectories represented as time-series. Apart from these spatial and temporal trajectory properties, some applications require processing and analysis of semantic information attributed to a (sub-sequence of a) trajectory.

While several systems have been proposed to manage trajectory data, they tend to only focus on and prioritize *one* of the representations mentioned above. Consequently, these systems are ill-suited to effectively and efficiently answer what we refer to as *complex trajectory queries*, i.e., queries that involve a combination of operations that require or benefit from trajectory data being represented in different forms (e.g., spatial and temporal).

We give a brief summary of the state of the art in trajectory data management systems in Section 2. In what follows, we outline how we intend to close this gap in the state of the art by conducting research that follows the tradition of algebraic query processing. In particular, we plan to address the following research challenges.

1. Define a data model that supports different, alternative representations for trajectory data rather than building a single general model (Section 3.2).
2. Define an operator algebra that (a) specifies operations that can be applied to trajectory data and (b) provides operators to translate between trajectory data representations (Section 3.3).
3. Design a query processor that supports complex trajectory queries (Section 3.4).

In order to demonstrate the validity of our approach, we plan to implement a proof-of-concept demonstrator (Section 4). Concluding remarks are given in Section 5.

2. State of the Art

To the best of our knowledge, no systems currently support different representations of trajectories. Polystores [6] store the same data multiple times in different formats, which makes them almost the *opposite* of what we need to work efficiently with different representations. In a recent study, Wang et al. [1] have created an overview of existing DBMS focused on trajectories. Systems such as Secondo [7] and MobilityDB [8] can store and query

Published in the Workshop Proceedings of the EDBT/ICDT 2023 Joint Conference (March 28-March 31, 2023, Ioannina, Greece)
johann.bornholdt@uni-konstanz.de (J. Bornholdt)
0000-0001-6183-1500 (J. Bornholdt)

[1]https://www.exc.uni-konstanz.de/collective-behaviour/

moving object data, but only support a relational representation of trajectories. Therefore, these systems are ill-suited to handle what we define as *complex trajectory queries*. We pose our research question: "How can we extend and apply traditional query processing techniques to complex trajectory queries?"

3. Data Model and Algebra

A *trajectory* is defined as $T = (S, D)$, where S is the spatiotemporal structure of the trajectory, defined as a sequence of (point, timestamp)-tuples $S = \langle (p_1, t_1), \ldots, (p_n, t_n) \rangle$ with $p_i = (x_i, y_i)$ and $1 \leq i \leq n$. Each component of the trajectory can have attributes with values from a domain $\mathcal{D}$, given by the function $D : I \times \mathcal{N} \to \mathcal{D}$, where I is an (open or closed) interval over S and $\mathcal{N}$ is the set of attributes. Note that for reasons of simplicity, we assume that overlapping intervals do not share common attribute names. As special cases, the interval $[1, n]$ denotes the entire trajectory, $[i, i+1]$ is a single edge, and $[i, i]$ is a single point.

3.1. Trajectory Representations

Trajectories defined by this conceptual model can be represented in different data models. Figure 2 illustrates some possible representations of the trajectories shown in Figure 1. More specifically, Figure 2a shows a mapping of our definition into the relational model, where a table is used to store the structure of the trajectory. Logical models of this nature are employed in existing trajectory management systems [7, 8].

Instead of a relational model, trajectories can be represented by a set of sequences (cf. column-based [9]), as shown in Figure 2b. When only reading the same parts of all trajectories, e.g., only the locations, a set of sequences is more efficient than a relational model.

Another option is to extract a graph representation from trajectories [10]. The extracted graph can then be stored using, e.g., the property graph data model [11], as shown in Figure 2c. In the property graph data model, a set of trajectories can be represented in labeled nodes and typed edges. In contrast, attribute values can be stored as node or edge properties.

Furthermore, it is possible to look at a trajectory as a time-series (cf. Figure 2d) and use time-series analysis either on the trajectory itself or on an aggregation, e.g., travel distance or speed. Liu et al. [12] use a time-series-based approach for trajectories to identify types of urban regions based on taxi trips.

3.2. Logical and Physical Data Model

In order to design a system that can represent trajectory data in different data models, we begin with a systematic

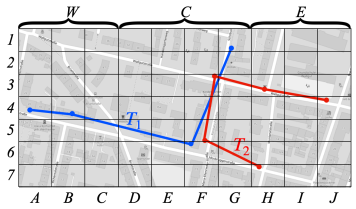


Figure 1: Sample trajectories.

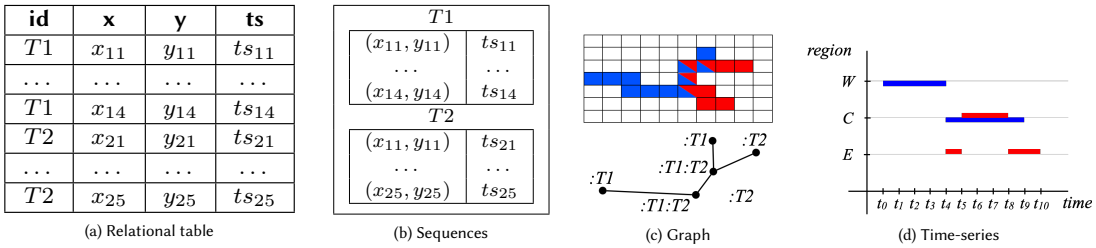
literature review of the current state of the art [1, 7, 8]. This literature review will identify the most commonly proposed trajectory representations and the essential operations. Following our approach, in which we do not want to favor any representation over all others, we will define a *logical data model* for each identified representation. We will use a consistent formal framework to enable transformation operations (cf. Section 3.3) between these representations.

There are several options for designing a *physical data model* employed by existing database systems that can be used to store trajectory data in our system, which can be beneficial for different operations. For instance, the row-based storage approach of traditional relational database systems is beneficial to query entire trajectories or filtering them based on their attribute values. A graph-based storage benefits queries that depend on regular path expression. Finally, the column-oriented storage of column stores and time-series database systems favors operations that prioritize analyzing a single aspect of trajectories. Apart from database systems that focus on a single physical model, we will also investigate polystores [6], typically used to store *different* data sets in one system in their most natural representation. In our case, however, we will study the use of polystores to store the *same* data set in multiple representations.

Finally, another well-known option from existing database systems is using index structures to enable different access paths. For example, a B+Tree index provides sorted access to a relational table. In the same way, an index could represent a relational table as a time-series, or another could represent a time-series as a spatial object. Therefore, we will also investigate how clustered, and unclustered indexes can be employed to render the need for a fixed physical data model obsolete.

3.3. Algebra

Based on the findings from the literature review in Subsection 3.2, we identify the need for handling *complex trajectory queries* on a logical level. Consider the following query Q: "find all trajectories T which pass through a region R during a time interval I." Following a filter-and-refine strategy to process Q, we can apply multiple



(a) Relational table (b) Sequences (c) Graph (d) Time-series

Figure 2: Examples of different trajectory representations.

operators: (a) a spatial operator to find all trajectories that pass through R, and (b) a temporal operator to find all trajectories that were recorded during I. While we can process Q with these two operators, we can also derive a third operator, which checks whether the timestamps of specific sample points of each trajectory lie within R during I. This spatiotemporal operator can use specialized algorithms that consider spatial and temporal aspects at the same time, similar to a join in relational databases.

The first two operators can be applied in any order and aim at reducing the search space, while the third operator verifies the correctness of the result. Following the traditional approach of database systems, we will develop an algebra with logical operators that can be combined to represent any *complex trajectory query*.

3.4. Query Processing

Complex trajectory queries combine parameters for different representations. Consider query Q from Subsection 3.3. To process Q, the spatial and the temporal parts of Q both have to be processed in their respective representation. Therefore we need to convert the logical operators of Q into physical operators.

Physical operators can be divided into two categories: *query operators* and *transformation operators*. Query operators evaluate the query. In the case of Q, they evaluate the spatial and temporal parameters. Transformation operators convert the trajectory data from one representation to another (see Figure 2). Note that both query and transformation operators can benefit from index structures.

For the physical operators, it is essential on which representation they are applied. Not all query operators work with all representations. Some combinations of operators and representations are less efficient, and some are impossible (e.g., extracting the position from a time series). For the spatial part of Q, we need to check for all trajectories if they intersect the given region. The intersection can be checked efficiently with the geometric representation using an R-Tree index. If the query operator is incompatible with the current representation of the given trajectory data, another representation should be obtained using a transformation operator. For the temporal part of Q, the data can be transformed into a time-series representation before applying the temporal query operator.

3.5. Query Optimizer

There are multiple ways to combine query and transformation operators to evaluate a given query. For instance, one could directly apply a query operator over an already materialized representation or perform a transformation *on the fly* from one representation to another and use a query operator over the new representation. As physical operators have different costs depending on the operator type, the representation method, and the cardinalities of the data, we will devise a cost model from them in order to build an optimizer that can generate execution plans. Our optimizer will be based on the Cascades Framework [13] building on Apache Calcite. Furthermore, to enable the reuse of materialized representations of trajectory data, we will investigate existing approaches for self-tuning database systems [14] and adapt them to our optimizer. The materialization is relevant if the transformation procedure is expensive and the representation is needed multiple times.

4. Proof of Concept

We propose to build the Chameleon system, a proof of concept, to demonstrate for the concepts shown in this paper. It is used to evaluate results from Section 3.3 & Section 3.4 empirically. Chameleon acts as an adapter layer between the user and a trajectory data source. Chameleon offers an API to query trajectory data based on spatial, temporal, and additional rich features. The API has different methods for each trajectory representation, meaning the user receives the data in the format that they need, e.g., as a sequence of tuples or as a relational table. It can be used with a combination of data sources for different representations. e.g., spatiotemporal DBMS (e.g., Secondo [7], MobilityDB [8]), graph DBMS

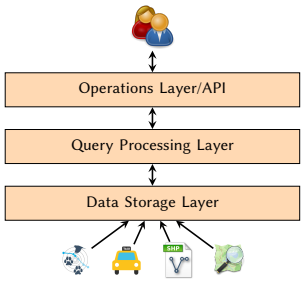


Figure 3: Overview of Chameleon's architecture.

(e.g., Neo4j), and can import data from CSV-Files or the public GPS traces from Openstreetmaps[2]. Chameleon supports *complex trajectory queries* that use properties from different representations.

The system architecture of Chameleon will consist of three layers: *operations layer*, *query processing layer*, and *data storage layer*, as shown in Figure 3.

- The *operations layer* of Chameleon offers an API that a user can interact with. The API is in the form of different methods for different kinds of operators to pose queries to the system. This layer is also responsible for returning the query result to the user.
- The *query processing layer* is responsible for the generation of execution plans. Furthermore, it determines which data representations to transform and which to materialize.
- The *data storage layer* contains the physical transformation operators and the access methods for the underlying data sources. It supports multiple data sources, e.g., CSV, shapefiles, etc.

5. Conclusion

In this paper, we have proposed a system to improve the management and processing of *complex trajectory queries*. By considering the diverse representations for trajectories, we can process all the parts of a complex query with the optimal methods. To build an efficient execution plan, we can use the extensive advancements of query optimization. Furthermore, we proposed to develop a proof of concept to evaluate the results of our system with real-world data.

Acknowledgments

This work is funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC 2117 – 422037984.

[2] https://www.openstreetmap.org/traces/

References

[1] S. Wang, Z. Bao, J. S. Culpepper, G. Cong, A survey on trajectory data management, analytics, and learning, ACM Comp. Surveys 54 (2021) 1–36.
[2] T. Chondrogiannis, J. Bornholdt, P. Bouros, M. Grossniklaus, History oblivious route recovery on road networks, in: ACM SIGSPATIAL, 2022, pp. 1–10.
[3] L. Moreira-Matias, J. Gama, M. Ferreira, J. Mendes-Moreira, L. Damas, Predicting taxi–passenger demand using streaming data, IEEE Trans. on Intelligent Transportation Systems 14 (2013) 1393–1402.
[4] J. Weppler, M. Belyaev, O. Solomina, M. Wikelski, W. Naumann, W. Pitz, Icarus-animal observation from iss, in: Proceedings of the International Astronautical Congress, IAC, 2017, pp. 5312–5322.
[5] U. Waldmann, H. Naik, N. Máté, F. Kano, I. D. Couzin, O. Deussen, B. Goldlücke, I-muppet: Interactive multi-pigeon pose estimation and tracking, in: DAGM-GCPR, 2022, pp. 513–528.
[6] R. Tan, R. Chirkova, V. Gadepally, T. G. Mattson, Enabling query processing across heterogeneous data models: A survey, in: IEEE BigData, IEEE, 2017, pp. 3211–3220.
[7] R. H. Guting, V. Almeida, D. Ansorge, T. Behr, Z. Ding, T. Hose, F. Hoffmann, M. Spiekermann, U. Telle, Secondo: An extensible dbms platform for research prototyping and teaching, in: IEEE ICDE, 2005, pp. 1115–1116.
[8] E. Zimányi, M. Sakr, A. Lesuisse, MobilityDB: A mobility database based on PostgreSQL and PostGIS, ACM TODS 45 (2020).
[9] D. J. Abadi, P. A. Boncz, S. Harizopoulos, Column-oriented database systems, Proceedings of the VLDB Endowment 2 (2009) 1664–1665.
[10] L.-Y. Wei, Y. Zheng, W.-C. Peng, Constructing popular routes from uncertain trajectories, in: ACM SIGKDD, 2012, pp. 195–203.
[11] L. Wörteler, M. Renftle, T. Chondrogiannis, M. Grossniklaus, Cardinality estimation using label probability propagation for subgraph matching in property graph databases., in: EDBT, 2022, pp. 2–285.
[12] X. Liu, Y. Tian, X. Zhang, Z. Wan, Identification of urban functional regions in chengdu based on taxi trajectory time series data, ISPRS International Journal of Geo-Information 9 (2020) 158.
[13] G. Graefe, The cascades framework for query optimization, IEEE Data Eng. Bull. 18 (1995) 19–29.
[14] S. Chaudhuri, V. Narasayya, Self-tuning database systems: a decade of progress, in: VLDB, 2007, pp. 3–14.