PG-TRAJECTORY: A PostgreSQL/PostGIS based Data Model for Spatiotemporal Trajectories

Ahmet Kucuk, Shah Muhammad Hamdi, Berkay Aydin, Michael A. Schuh, Rafal A. Angryk
Department of Computer Science
Georgia State University Atlanta, GA 30302
Email: {akucuk1, shamdi1, baydin2, mschuh, rangryk}@cs.gsu.edu

***Abstract*—Because of tremendous advances in GPS and location-based web services, highly available spatiotemporal trajectory data throws an important challenge - knowledge discovery from trajectories. Knowledge discovery tasks on Big trajectory Data such as classifying, clustering and outlier detection require a dedicated data model, that is built on the relational database, able to handle both point and region trajectories and supported by distance and similarity measure functions. This paper introduces PG-TRAJECTORY, a data model built on PostGIS, the spatial extension of the popular PostgreSQL database. PG-TRAJECTORY contains all necessary data types and supporting functions for storing and manipulating spatiotemporal trajectories. We have discussed the basic structure of the data model with the working principles of the supporting functions and shown some real life query examples. For proving the scalability and effectiveness of our model, we have shown experiments on artificial and real datasets for each of the point and region trajectories.**

I. INTRODUCTION

In recent years, outstanding growth in GPS enabled devices, satellite imagery technologies, location-based web services and social networks have triggered a massive expansion in spatiotemporal datasets [1]. Many consumer-oriented applications such as social networks (Facebook, Twitter, Swarm), mobile services with real-time navigations (Google Maps, Apple Maps) and taxi services (Uber, Lyft) generate spatiotemporal data [2]. Furthermore, there are many massive spatiotemporal data repositories deployed by scientific research organizations that monitor the moving objects. These include solar events [3], animal migrations [4], and meteorological phenomena [5].

One major portion of the spatiotemporal data is the spatiotemporal trajectory data. Described in basic terms, a spatiotemporal trajectory is the periodic/non-periodic recording of spatial locations for a moving object as it is depicted in Fig. 1. In other words, trajectories are the collections of temporally annotated spatial snapshots. The formal definitions and data models of spatiotemporal trajectories are given in [6].

Creating data models for storing, processing, and managing of *Big Spatiotemporal Data* is of great importance for both consumer-oriented and scientific Big Data applications. Recently, to answer the needs of the community, new techniques and frameworks for spatiotemporal trajectory management have been introduced. Pelekis *et al.* introduced Hermes, that is a database engine for querying trajectories [7]. Aydin *et al.* presented data models for cloud-based trajectory storage specifically designated for spatiotemporal joins [8]. Xie *et al.* presented "Elite", which is an elastic storage framework for spatiotemporal trajectories [9].

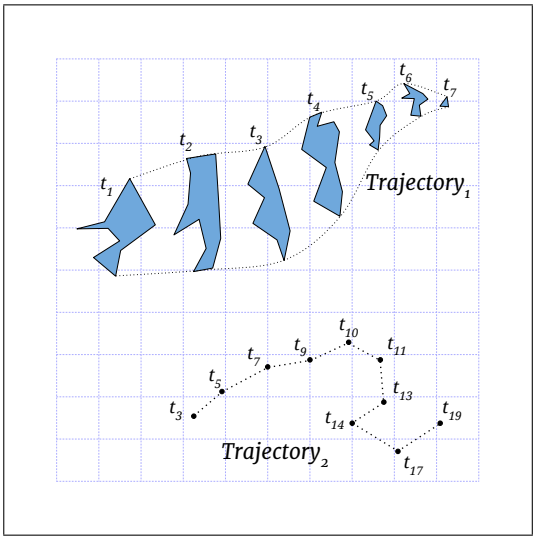


Fig. 1: Example of point and region trajectory

Following the trend in the advancements of databases, recent trajectory storage models focus on non-relational databases or elastic infrastructures to leverage the parallel computational power of underlying architectures. In this work, we address the trajectory modeling problem in relational databases to offer more effective server-side processing environment. We present a spatiotemporal trajectory data model for relational database environments. Our extension, PG-TRAJECTORY, offers a generalized and discrete trajectory model for both moving point and region objects. PG-TRAJECTORY extension provides various similarity and significance measures including Jaccard, OMAX and J*, as well as distance measures and auxiliary functions such as Edit Distance, Euclidean Distance, Duration, Area, and Overlaps. Providing the much-needed functionalities for data analysis in *Big Spatiotemporal Data* applications is our prime motivation.

The rest of this paper is organized as follows. Section II highlights the related work. Then we present trajectory data model, list of functions and some example queries in Section III, IV and V. In Section VI, we present experiments on different queries in real and artificial datasets. Lastly, in Section VII, we conclude our paper, and present future work.

II. Related Work

Because of numerous use cases of trajectory data, modeling of trajectory data occupies a rich domain in literature. The modeling of moving object datasets is rooted in 1998 when Wolfson *et al*. [10] proposed to store motion vectors to capture the current position of a moving object. Their Moving Object Spatio-Temporal (MOST) model was intended to predict the future movement of point trajectories by using Future Temporal Logic (FTL) language.

Instead of storing only current positions, Güting and his group came with a series of publications, which described a comprehensive data model for storing all trajectory segments (past and present) in relational and object-oriented databases. After raising the issues and questions in [11], they proposed an abstract data type [6], where the trajectory segments were represented by an infinite set of curves and then they proposed a discrete data type in [12], where those segments were represented by a finite number of polylines. Finally, they enriched their model by a set of trajectory related algorithms [13]. Their model, which accounts for both point and polygon trajectories, provided a solid foundation for any further research in trajectory data modeling.

For last two decades, modeling of trajectory data has been appearing in literature in many forms to solve new problems by considering new areas of application. For example, modeling of trajectory data in road constrained spatial networks [14] and modeling of semantic trajectories, which stores semantic information with different trajectory segments [15].

Though the research for modeling trajectories started in the mid-90s, this field has never lost its appeal. Rather, with the remarkable advancement of data sensing technologies such as GPS, GSM and RFID, now there are more trajectory data available than ever before. Data mining community has been using this data in trajectory classification, clustering and outlier analysis. Efficient and accurate similarity measures on trajectory are very important for these data mining tasks.

Several research works have presented a number of distance-based similarity measures for trajectories, such as Euclidean distance [16], Dynamic Time Warping (DTW) [17], Edit Distance with Real Penalty (ERP) [18], Longest Common Subsequence (LCSS) [19], Edit Distance with Real Sequences (EDR) [20] etc. In [20], Chen *et al*. have shown the comparative superiority of EDR over other measures in terms of robustness, accuracy and ability to handle noisy trajectory data. For region based trajectories, co-occurrence significance measures such as Jaccard [21], OMAX [22] and J* [23] have been used.

Taking these similarity measures in consideration, we have modeled an abstract data type for storing spatiotemporal trajectories. Our model implements basic trajectory operations described in [12], plus the aforementioned similarity and distance measures so that the model can be easily used for kNN queries of machine learning applications.

III. Modeling of Trajectory in Relational Database

Trajectory is a spatiotemporal concept. When an object moves in space (with predefined spatial frame of reference) with respect to time, the object forms a trajectory. The "object" refers to any real or virtual entity, that is capable of moving from one location to another. The domain of moving objects is huge; from microscopic particles to gigantic stellar events, from flying bird to walking human, from real players playing in field to virtual players in computer games. Trajectory data is the data collected by some sensing device that is attached with or detached from the physical body of the moving object. The data comes in the form of timestamp-geometry pairs, where each timestamp-geometry pair represents the spatial location (for region trajectory, also the shape) of the object at a particular time instant. Taking the theoretical definitions in consideration, we simplified our model using following characteristics and constraints.

- Each trajectory consists of one or more timestamp-geometry pairs.
- The difference between the start time and end time of a trajectory defines its lifespan/duration.
- Due to the limitations of the sensing devices, it is not always possible to precisely calculate the object's spatial position in a periodic manner. For this reason, we have considered the sampling interval to be constant or variable.
- The spatial components of the trajectory are represented by either points or polygons.
- Point and polygon geometries follow the standards of Open Geospatial Consortium [24].
- All geometries of a trajectory must be same type, i.e. points or polygons.

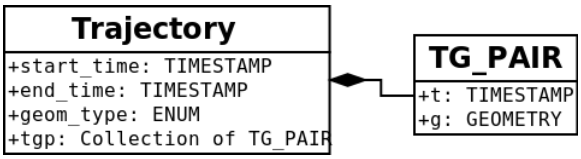


Fig. 2: Data model of two abstract data types - `TG_PAIR` and `Trajectory`

UML diagram of Fig. 2 explains the Trajectory data model. Each `TG_PAIR` object stores an individual spatial snapshot of a given timestamp. `Trajectory` is an object, which stores an ordered collection of `TG_PAIR`. Additionally, it stores some metadata such as type of geometry (enumerated as point or polygon) and temporal boundary (start time and end time).

As most trajectory data currently available use points and polygons for representing the spatial components of point and region trajectories, we have restricted our model to support only these two kinds of trajectories. Our data model allows the storage of both point and region trajectories in the same table, but does not allow one trajectory to have both point and polygon representation as the spatial component.

IV. List and Description of Functions

When the trajectories are represented in the raw format in a database table, i.e. using three columns $(traj\_id, timestamp, geometry)$, users are required to write complex queries and applications for implementing trajectory manipulating functions. Our model makes these queries easier by wrapping up widely used trajectory functions within the data model. These functions are defined in several works [11], [20], [21]. The data model and functions provide the users an easy-to-use interface so that they can store and manipulate trajectory data without being concerned of complex data structures.

In table I, we present the list of functions with their domain or type of trajectory, input parameters and return type. Some functions take one or two `Trajectory` objects as arguments while others take a collection of `TG_PAIR` objects. To distinguish between these two types, we have added prefix *"t_"* and *"tg_"* before the actual function names. The column Type of Trajectory denotes whether the function is defined for point trajectories or for region trajectories.

We have classified these functions in 6 categories - constructors, accsessors, auxiliary functions, distance measures, co-occurrence measures and modifier functions.

A. Constructor

_trajectory(TG_PAIR[]): This method is used for creating a `Trajectory` object from a collection of `TG_PAIR`. Additional metadata such as start time, end time and type of geometries are computed and set as instance variables by this function.

B. Accessor Functions

The functions of this category retrieve different characteristics given one or two `Trajectory` objects or a collection of `TG_PAIR`.

tg_start_time(TG_PAIR[]): Returns $time\_instant$ value of first tg_pair element.

tg_end_time(TG_PAIR[]): Returns $time\_instant$ value of last tg_pair element.

tg_mbr(Trajectory): It returns the minimum bounding rectangle of all the geometries of a region trajectory.

tg_type(TG_PAIR[]): It checks whether all geometries of a trajectory has consistent representation, i.e. point or polygon and returns the geometry type.

t_duration(Trajectory): Returns the difference of the end_time and $start\_time$ of a trajectory.

t_distance(Trajectory): It returns the sum distance of the consecutive points in a point trajectory.

t_area(Trajectory): It returns the sum area of the consecutive polygons in a region trajectory.

t_volume(Trajectory): It returns the spatio-temporal volume of a region trajectory. This function is only defined for the trajectories that have valid sampling interval.

$$volume = t\_area(Trajectory) * t\_duration(Trajectory)$$

TABLE I: List of Functions

Function Name	Type of Trajectory	Input Parameters	Return Type
_trajectory	Both	$TG\_PAIR[]$	$Trajectory$
tg_start_time	Both	$TG\_PAIR[]$	$Timestamp$
tg_end_time	Both	$TG\_PAIR[]$	$Timestamp$
tg_mbr	Region	$TG\_PAIR[]$	$Geometry$
tg_type	Both	$TG\_PAIR[]$	$Enum$
t_duration	Both	$Trajectory$	$Time\,interval$
t_distance	Point	$Trajectory$	$Real$
t_area	Region	$Trajectory$	$Real$
t_volume	Region	$Trajectory$	$Real$
t_geom_at	Both	$Trajectory$, $Timestamp$	$Geometry$
tg_zNormalize	Point	$TG\_PAIR[]$, $Integer$, $Integer$	$Geometry[]$
t_euclidean_distance	Both	$Trajectory$, $Trajectory$	$Real$
t_overlaps	Both	$Trajectory$, $Trajectory$	$Boolean$
t_union	Region	$Trajectory$, $Trajectory$	$Trajectory$
t_intersection	Region	$Trajectory$, $Trajectory$	$Trajectory$
t_jaccard	Region	$Trajectory$, $Trajectory$	$Real$
t_omax	Region	$Trajectory$, $Trajectory$	$Real$
t_jaccard_star	Region	$Trajectory$, $Trajectory$	$Real$
t_edit_distance	Both	$Trajectory$, $Trajectory$	$Real$
t_m_distance	Point	$Trajectory$, $Trajectory$	$Real[]$
t_add_head	Both	TG_PAIR, $Trajectory$	$Trajectory$
t_add_tail	Both	TG_PAIR, $Trajectory$	$Trajectory$
t_drop_head	Both	$Trajectory$	$Trajectory$
t_drop_tail	Both	$Trajectory$	$Trajectory$
t_update_geom_at	Both	$Timestamp$, $Geometry$, $Trajectory$	$Trajectory$

t_geom_at(Trajectory, Timestamp): This function returns the geometry of the trajectory at the given time instant. If it is point trajectory and there is no geometry record at the given time, linearly interpolated point is returned.

t_overlaps(Trajectory, Trajectory): If at any time instant, geometries of both trajectories overlap, then this function returns true, otherwise false.

C. Auxiliary Functions

The functions of this category are mainly used by distance measures and co-occurrence measures functions.

tg_zNormalize(TG_PAIR[], Integer, Integer): This function is used to z-Normalize the point geometries of a trajectory segment bounded by the $start\_index$ and end_index in second and third parameter. This function is used as an auxiliary function of Euclidean distance [25]. If X and Y are the respective sets of x and y coordinates in that trajectory segment, then X and Y are z-Normalized and an array of z-Normalized point geometries are returned.

$$Z(point_i) = (Z(x_i), Z(y_i)) = \left(\frac{x_i - \mu(X)}{\sigma(X)}, \frac{y_i - \mu(Y)}{\sigma(Y)} \right)$$

Here, μ and σ denote arithmetic mean and standard deviation.
t_union(Trajectory, Trajectory): This function returns the spatiotemporal union of two trajectories[22].
t_intersection(Trajectory, Trajectory): This function returns the spatiotemporal intersection of two trajectories [22].

D. Distance Measures

These functions can be used to calculate spatial/spatio-temporal distance, i.e. the divergence of two given trajectories.
t_euclidean_distance(Trajectory, Trajectory): This function returns the time-relaxed Euclidean distance of two trajectories. If Q and S are two input trajectories and $|Q| \leq |S|$, where $|Q|$ and $|S|$ are the respective lengths of their `TG_PAIR`, then their Euclidean distance is found by shifting Q k-times along S [25]. At first, Q is z-Normalized in full length. At each shift of Q, the corresponding segment of S of length $|Q|$ is z-Normalized and their Euclidean distance is calculated [20]. The minimal local distance found from k shifts is finally returned as the Euclidean distance.

$$Euclidean\_distance(Q, S)$$
$$= \min_{0 \leq k \leq |S|-|Q|} \sqrt{\sum_{i=1}^{|Q|} dist(Q_i, z(S_{i+k}))}$$
$$= \min_{0 \leq k \leq |S|-|Q|} \sqrt{\sum_{i=1}^{|Q|} (Q_{x,i} - z(S_{x,i+k}))^2 + (Q_{y,i} - z(S_{y,i+k}))^2}$$

For region trajectories, the centroid of each polygon is calculated to convert the region trajectory into point trajectory and then Euclidean distance is calculated.
t_edit_distance(Trajectory, Trajectory, Float): This function uses edit distance for trajectories defined by Vlachos *et al.* [20]. Since it was only defined for point trajectories, we extended it for region trajectories. We designed a measure to decide whether a region in first trajectory close enough to a region in the second trajectory. We used following formula for this decision, where e is error rate that is given in the function as the third parameter. Convex hull of a region trajectory is the envelope over all polygon geometries.

$$c_1 = Convexhull(Tr_1)$$
$$c_2 = Convexhull(Tr_2)$$
$$a_1 = Area(Union(c_1, c_2))$$
$$a_2 = Area(Convexhull(c_1, c_2)))$$
$$e < \frac{a1}{a2}$$

t_m_distance(Trajectory, Trajectory): This function is defined in [14]. It returns an array of euclidean distances between two point trajectories at all co-occurring timestamps.

E. Co-occurrence Significance Measures

Three co-occurrence measuring functions Jaccard, OMAX and J* are members of this group. These are spatiotemporal similarity measures of two region trajectories.
t_jaccard(Trajectory, Trajectory): This function returns the ratio of intersection volume and union volume of two region trajectories [22].

$$Jaccard = \frac{t\_volume(t\_intersection(Tr_1, Tr_2))}{t\_volume(t\_union(Tr_1, Tr_2))}$$

t_omax(Trajectory, Trajectory): This function returns the ratio of intersection volume and maximum individual volume of two region trajectories. As union is not used, this function has less computational overhead than Jaccard.

$$OMAX = \frac{t\_volume(t\_intersection(Tr_1, Tr_2))}{max(t\_volume(Tr_1), t\_volume(Tr_2))}$$

t_jaccard_star(Trajectory, Trajectory): This function returns another significance measure J* of two region trajectories [23]. J* is calculated by finding the temporal coexistence of Tr_1 and Tr_2. Later, within the temporal co-existing interval, the time intervals where spatial co-occurrence appears are found. For these intervals, intersection and union volumes are found for both trajectories. Finally, the ratio between the intersection and union volume is returned.

F. Modifier Functions

The functions of these categories are used in UPDATE statements to change the recorded trajectory data. New `TG_PAIR` object can be added/removed and existing `TG_PAIR` object can be modified using these functions.
t_add_head(TG_PAIR, Trajectory): This function inserts a new `TG_PAIR` element at the beginning of the `TG_PAIR` collection of a `Trajectory`. It returns the `Trajectory` with updated `TG_PAIR` collection.
t_add_tail(TG_PAIR, Trajectory): This function adds a new `TG_PAIR` element at the end of the `TG_PAIR` collection of the given `Trajectory` and returns the `Trajectory` with updated `TG_PAIR` array.
t_drop_head(Trajectory): It returns the updated `Trajectory` after removing its first `TG_PAIR` element.
t_drop_tail(Trajectory): It removes the last `TG_PAIR` element of the given `Trajectory` and returns the updated `Trajectory`.
t_update_geom_at(Timestamp, Geometry, Trajectory): This function sets the geometry at the given time instant. It returns the updated `Trajectory` with the modified `TG_PAIR` array.

V. Some Example Queries

In this section, we present some queries that demonstrate real life use cases of PG-Trajectory in storing and manipulating point and region trajectories.

Let's consider the following relational schema for a table $taxi\_ride$ that stores the point trajectories found from the GPS data of the taxis in the city of Atlanta. The table also

stores the source and destination of the taxi rides.

taxi_ride($\underline{id}$: Integer, $from$: Text, to: Text, $traj$: Trajectory)

```
CREATE TABLE taxi_ride
(id INTEGER PRIMARY KEY, from TEXT,
to TEXT, traj Trajectory);
```

Find all the taxi rides which reached at the airport within 30 minutes after starting the journey at Downtown.

```
SELECT id
FROM taxi_ride
WHERE from = 'Downtown' AND to = 'Airport'
AND t_duration(traj) <=
INTERVAL '30 minutes';
```

Find the source and destination of 5 longest (in distance) taxi rides.

```
SELECT id, from, to
FROM taxi_ride
ORDER BY t_distance(traj) DESC
LIMIT 5;
```

Find the number of taxi rides from Decatur to Lenox in last 24 hours.

```
SELECT COUNT(*)
FROM taxi_ride WHERE
from = 'Decatur' AND to = 'Lenox'
AND traj.start_time > CURRENT_TIMESTAMP -
INTERVAL '1 day' AND
traj.end_time < CURRENT_TIMESTAMP;
```

Find the taxi rides from Doraville to Chamblee that follows exactly the same route.

```
SELECT DISTINCT t1.id
FROM (SELECT * FROM taxi_ride t
WHERE t.from = 'Doraville' AND
t.to = 'Chamblee') AS t1,
(SELECT * FROM taxi_ride t
WHERE t.from = 'Doraville' AND
t.to = 'Chamblee') AS t2
WHERE t1.id <> t2.id AND
t_euclidean_distance(t1.traj, t2.traj) = 0;
```

Now, let's consider a table with region trajectories, which stores the solar events.

solar_events($\underline{id}$: Integer, $traj$: Trajectory)

```
CREATE TABLE solar_events
(id INTEGER PRIMARY KEY, traj Trajectory);
```

Find the pairwise co-occurrence measure (Jaccard) of the trajectories stored in the table $solar\_events$.

As OMAX is computationally less expensive than Jaccard, an efficient method for this query is the Filter-and-Refine approach. First, less co-occurring trajectories having OMAX value less than co-occurrence threshold(in this example, 0.5) are filtered. Then Jaccard of this reduced set of trajectories is calculated.

```
SELECT (R.A).id, (R.B).id,
t_jaccard((R.A).traj, (R.B).traj)
FROM (
SELECT t1 as A, t2 as B,
t_omax(t1.traj, t2.traj) as omax
FROM solar_events t1, solar_events t2
) AS R
WHERE R.omax > 0.5;
```

Find 10 closest trajectories of the sigmoid trajectory with id = 3. This is an example of k nearest neighbor query. We use spatiotemporal edit distance for this query.

```
SELECT * FROM (
SELECT t2.id
FROM solar_events t1, solar_events t2
WHERE t1.id = 3 AND t2.id <> t1.id
ORDER BY t_edit_distance(t1.traj, t2.traj)
) AS R
LIMIT 10;
```

VI. Experiments

We implemented PG-Trajectory as an extension of the PostgreSQL database that is a popular open source relational database with the spatial extension, PostGIS, which is used for storing and processing spatial data. PG-Trajectory is implemented in PL/pgSQL, a procedural language for PostgreSQL. `TG_PAIR` and `Trajectory` of Fig. 2 are implemented as custom PostgreSQL data types and all functions are implemented as PostgreSQL subroutines. Using PL/pgSQL subroutines gives some significant advantages such as easy integration to PostgreSQL and inheritance of high-level PostGIS functions [26].

PG-Trajectory provides different functions for both point and region trajectories. The objective of our experiments is to provide a general understanding of how the data model works. Therefore, we focussed on the effectiveness of our extension in storing and manipulating real life trajectory data. In this section, we present the runtimes of the distance and co-occurrence functions recorded on both artificial and real-life datasets.

PG-Trajectory requires PostgreSQL database with PostGIS installed. We tested the model in a dedicated server of 128 CPUs and 1 TB main memory. PostgreSQL 9.5 and PostGIS 2.2 are used in these experiments. More details about version compatibility can be found in the project page. [1]

A. Datasets

Table II lists the datasets used in the experiments. Artificial datasets consist of two parts: Point and Region. Artificial region dataset is generated by Evolving Region Moving Object Dataset Generator [27]. This software is designed to generate co-occurring region trajectories. While using this software

[1] http://pg-trajectory.dmlab.cs.gsu.edu/

TABLE II: Datasets

Source	Type	Length	Tag
Artificial	Point,Region	25	AR25
Artificial	Point,Region	50	AR50
Artificial	Point,Region	100	AR100
Artificial	Point,Region	200	AR200
Solar Data	Region	16-2093	Solar
GeoLife	Point	3-144	GeoLife

for generating artificial region trajectory dataset, we specified some characteristics such as lifespan, region complexity and intersection percentage. Since the complexity of distance and co-occurrence measures in PG-TRAJECTORY mostly depends on the length of the trajectory, we used different length values in data generation. The length of a trajectory is the number of elements in the `TG_PAIR` collection. For region trajectory, the number of points in the region shape (polygon) is an important factor of complexity. As this polygon representation complexity is more related to PostGIS than our PG-TRAJECTORY extension, we preferred to keep them in a constant range between 20 to 30 points for each polygon. Using this software, we generated four different trajectory datasets with lengths of 25, 50, 100 and 200. Each dataset contains 1,000 trajectories. To have similar characteristics, we produced point trajectory datasets using the centroid of each region occurred in region trajectory datasets. That gives us another four datasets with same length property, but instead of region geometries, we had point geometries in these datasets.

To see the real-life performance of PG-TRAJECTORY, we experimented with one real point trajectory dataset and one real region trajectory dataset. Region trajectory dataset is the tracked solar event dataset. This dataset contains 4 different solar event types with thousands of tracked instances over one month period of time (collected by the SDO mission of NASA in June 2012). This dataset has already been used for spatiotemporal co-occurrence pattern mining in [8]. In order to have the same size between artificial and real-life datasets, we randomly picked 250 trajectories from each event type, and thus ended up with 1,000 trajectories in total to have the same number of trajectories with the artificial datasets. The range of length of the solar dataset is 16 to 2,093 and lengths are randomly distributed. For the real life point trajectory data, we used Microsoft's GeoLife GPS dataset, which is one of the largest publicly available GPS dataset [28], [29], [30]. GeoLife dataset is collected from 182 users in a period of over 5 years (April 2007 to August 2012). The dataset contains 17,621 trajectories with more than 48,000 hours of total duration and 1.2 million kilometers of total trajectory length. We extracted 1000 random trajectory samples from the GeoLife dataset. In our random choice, we picked trajectories having lengths less than 150 to make it comparable with our artificial dataset. Finally, we got 1,000 point trajectories with the length range of 3 to 144.

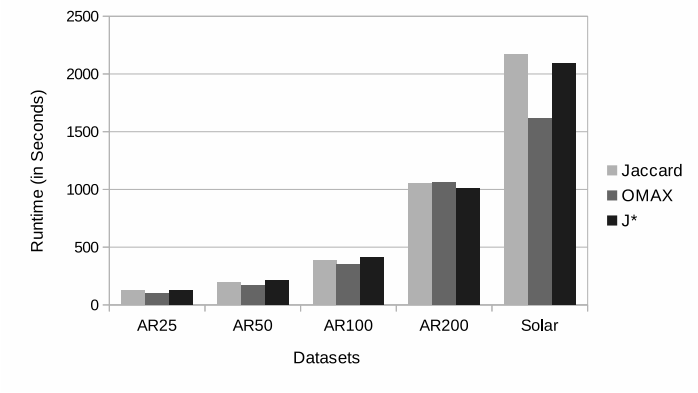


Fig. 3: Jaccard, OMAX and J* runtimes on region trajectories of varying lengths

B. Similarity Measures

We have recorded runtimes of three similarity measures: Jaccard, OMAX and J*. This experiment joins two tables, each having a column of Trajectory data type. We keep two copies of the same table to join them. For example, in AR25 table, we have 1000 trajectory entries. We join this table with an identical copy of this table without using any joining conditions. So it gives us 1000*1000 results for each query. As our focus is the runtimes of the measures, we have not included any joining condition or any application-specific optimizations.

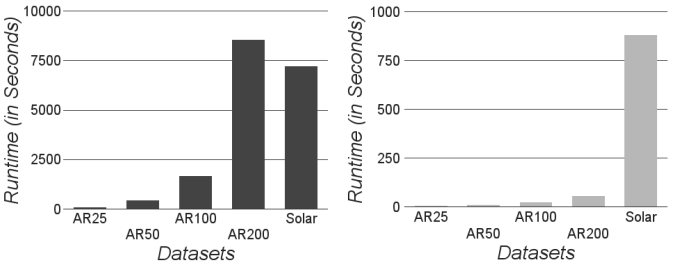
We sent separate queries for each measure on each data set. In theory, if the number of points in each polygon geometry is constant, all three measures have $O(n)$ complexity. Fig. 3 shows the runtime of the measures that justify our theoretical complexity.

Another observation from Fig. 3 is that solar event dataset gives the longest runtime. It can be explained by two reasons. The first reason is the length of the solar data, which is much longer than the artificial dataset. Secondly, polygon geometries of solar data contain more points, which increase the runtime of union and intersection operations calculated by PostGIS.

C. Distance Measures

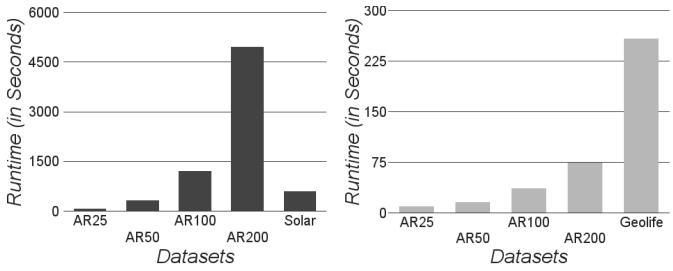
Euclidean and Edit Distance are provided in PG-TRAJECTORY as distance measures. Both methods can be used for region and point trajectories. The complexity of Euclidean Distance is $O(n)$ whereas complexity of Edit Distance is $O(mn)$, where m and n are lengths of input trajectories [31] and $n > m$, when we consider other variables as constant. The formula of edit distance require exponential complexity, but we implemented it using dynamic programming, so $O(mn)$ complexity stems from the size of dynamic programming table.

In our queries, to get the runtimes of distance measures, we followed the same steps we used for co-occurrence measures. But this time, we had 100 point trajectories in each table because of the greater complexity of edit distance. Fig. 4



(a) Edit Distance (b) Euclidean Distance

Fig. 4: Distance measures runtime on region datasets



(a) Edit Distance (b) Euclidean Distance

Fig. 5: Distance measures runtime on point datasets

shows runtimes for both distance measures in different region datasets, while Fig. 5 shows runtime on point datasets.

D. Additional Functionalities

Constructor is one of the most important functions of our extension, as it provides the most convenient way to create an object of the `Trajectory` data type. In this function, we compute some useful properties of a trajectory such as start time, end time and the minimum bounding rectangle (MBR) of spatial area (for region trajectories only). This calculation creates an overhead in the construction of trajectory object, but it can give significant speed up in some use-cases. For example, precalculated MBRs of trajectories makes spatial indexing easy. In Fig. 6, we show insertion times of 1000 trajectories for artificial and real datasets. Insertion time is also linearly dependent on the length of trajectory. We again can observe the effect of length difference in real-life datasets.

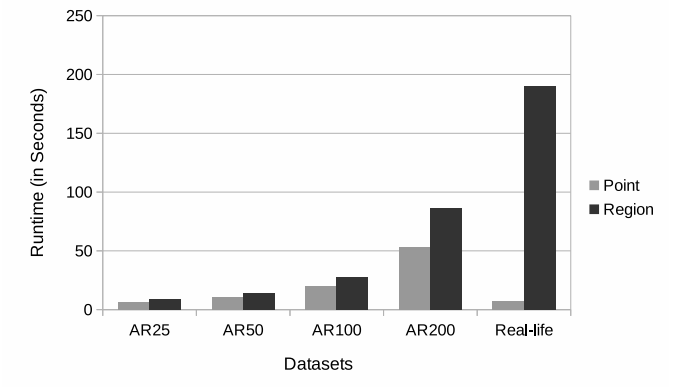


Fig. 6: Insertion times of 1000 trajectories in point and region datasets

VII. CONCLUSION

This paper introduces PG-TRAJECTORY, a PostGIS/PostgreSQL based data model for storing and manipulating spatiotemporal trajectory data in relational database. As this model covers both point and region trajectories, it can be used by various fields of spatiotemporal data analysis such as traffic flow analysis, animal migration analysis, solar physics and astronomy etc. Similarity and distance measures functions that use this model can be used by data mining community in writing simpler queries for complex tasks such as k-nearest neighbor trajectory search. In the future, we want to add more functions to existing data model and we are planning to extend this model to support semantic trajectory, where the trajectory is segmented into a list of ”stops” and ”moves” and additional information is included in each segment.

ACKNOWLEDGMENT

This work was supported in part by two NASA Grant Awards (No. NNX11AM13A, and No. NNX15AF39G), and one NSF Grant Award (No. AC1443061). The NSF Grant Award has been supported by funding from the Division of Advanced Cyberinfrastructure within the Directorate for Computer and Information Science and Engineering, the Division of Astronomical Sciences within the Directorate for Mathematical and Physical Sciences, and the Division of Atmospheric and Geospace Sciences within the Directorate for Geosciences.

REFERENCES

[1] M. A. Nascimento, D. Pfoser, and Y. Theodoridis, “Synthetic and real spatiotemporal datasets,” *IEEE Data Eng. Bull.*, vol. 26, no. 2, pp. 26–32, 2003.

[2] D. Quercia, N. Lathia, F. Calabrese, G. Di Lorenzo, and J. Crowcroft, “Recommending social events from mobile phone location data,” in *Data Mining (ICDM), 2010 IEEE 10th International Conference on*. IEEE, 2010, pp. 971–976.

[3] M. A. Schuh, R. A. Angryk, K. G. Pillai, J. M. Banda, and P. C. Martens, “A large-scale solar image dataset with labeled event regions.” in *ICIP*, 2013, pp. 4349–4353.

[4] M. Buchin, S. Dodge, and B. Speckmann, “Context-aware similarity of trajectories,” in *Geographic information science*. Springer, 2012, pp. 43–56.

[5] J. Wang, K. Young, T. Hock, D. Lauritsen, D. Behringer, M. Black, P. G. Black, J. Franklin, J. Halverson, J. Molinari *et al.*, “A long-term, high-quality, high-vertical-resolution gps dropsonde dataset for hurricane and other studies,” *Bulletin of the American Meteorological Society*, vol. 96, no. 6, pp. 961–973, 2015.

[6] R. H. Güting, M. H. Böhlen, M. Erwig, C. S. Jensen, N. A. Lorentzos, M. Schneider, and M. Vazirgiannis, “A foundation for representing and querying moving objects,” *ACM Transactions on Database Systems (TODS)*, vol. 25, no. 1, pp. 1–42, 2000.

[7] N. Pelekis, E. Frentzos, N. Giatrakos, and Y. Theodoridis, “Hermes: A trajectory db engine for mobility-centric applications,” *International Journal of Knowledge-Based Organizations (IJKBO)*, vol. 5, no. 2, pp. 19–41, 2015.

[8] B. Aydin, V. Akkineni, and R. A. Angryk, "Modeling and indexing spatiotemporal trajectory data in non-relational databases," *Managing Big Data in Cloud Computing Environments*, p. 133, 2016.

[9] X. Xie, B. Mei, J. Chen, X. Du, and C. S. Jensen, "Elite: an elastic infrastructure for big spatiotemporal trajectories," *The VLDB Journal*, pp. 1–21, 2016.

[10] O. Wolfson, B. Xu, S. Chamberlain, and L. Jiang, "Moving objects databases: Issues and solutions," in *Scientific and Statistical Database Management, 1998. Proceedings. Tenth International Conference on*. IEEE, 1998, pp. 111–122.

[11] M. Erwig, R. H. Gu, M. Schneider, M. Vazirgiannis *et al.*, "Spatio-temporal data types: An approach to modeling and querying moving objects in databases," *GeoInformatica*, vol. 3, no. 3, pp. 269–296, 1999.

[12] L. Forlizzi, R. H. Güting, E. Nardelli, and M. Schneider, *A data model and data structures for moving objects databases*. ACM, 2000, vol. 29, no. 2.

[13] J. A. C. Lema, L. Forlizzi, R. H. Güting, E. Nardelli, and M. Schneider, "Algorithms for moving objects databases," *The Computer Journal*, vol. 46, no. 6, pp. 680–712, 2003.

[14] R. H. Güting, T. de Almeida, and Z. Ding, "Modeling and querying moving objects in networks," *The VLDB JournalThe International Journal on Very Large Data Bases*, vol. 15, no. 2, pp. 165–190, 2006.

[15] S. Spaccapietra, C. Parent, M. L. Damiani, J. A. de Macedo, F. Porto, and C. Vangenot, "A conceptual view on trajectories," *Data & knowledge engineering*, vol. 65, no. 1, pp. 126–146, 2008.

[16] S.-L. Lee, S.-J. Chun, D.-H. Kim, J.-H. Lee, and C.-W. Chung, "Similarity search for multidimensional data sequences," in *Data Engineering, 2000. Proceedings. 16th International Conference on*. IEEE, 2000, pp. 599–608.

[17] M. Vlachos, D. Gunopulos, and G. Das, "Rotation invariant distance measures for trajectories," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2004, pp. 707–712.

[18] L. Chen and R. Ng, "On the marriage of lp-norms and edit distance," in *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*. VLDB Endowment, 2004, pp. 792–803.

[19] M. Vlachos, G. Kollios, and D. Gunopulos, "Discovering similar multidimensional trajectories," in *Data Engineering, 2002. Proceedings. 18th International Conference on*. IEEE, 2002, pp. 673–684.

[20] L. Chen, M. T. Özsu, and V. Oria, "Robust and fast similarity search for moving object trajectories," in *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*. ACM, 2005, pp. 491–502.

[21] K. G. Pillai, R. A. Angryk, J. M. Banda, M. A. Schuh, and T. Wylie, "Spatio-temporal co-occurrence pattern mining in data sets with evolving regions," in *Data Mining Workshops (ICDMW), 2012 IEEE 12th International Conference on*. IEEE, 2012, pp. 805–812.

[22] K. G. Pillai, R. A. Angryk, and B. Aydin, "A filter-and-refine approach to mine spatiotemporal co-occurrences," in *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 2013, pp. 104–113.

[23] B. Aydin, V. Akkineni, and R. A. Angryk, "Time-efficient significance measure for discovering spatiotemporal co-occurrences from data with unbalanced characteristics," in *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems, Bellevue, WA, USA, November 3-6, 2015*, 2015, pp. 80:1–80:4. [Online]. Available: http://doi.acm.org/10.1145/2820783.2820871

[24] OGC, *OpenGIS Standards*, 2010-04-27. [Online]. Available: http://www.opengeospatial.org/standards

[25] C. Hundt, B. Schmidt, and E. Schomer, "Cuda-accelerated alignment of subsequences in streamed time series data," in *Parallel Processing (ICPP), 2014 43rd International Conference on*. IEEE, 2014, pp. 10–19.

[26] P. Ramsey *et al.*, "Postgis manual," *Refractions Research Inc*, 2005.

[27] B. Aydin, R. A. Angryk, and K. G. Pillai, "Ermo-dg: Evolving region moving object dataset generator." in *FLAIRS Conference*, 2014.

[28] Y. Zheng, L. Zhang, X. Xie, and W.-Y. Ma, "Mining interesting locations and travel sequences from gps trajectories," in *Proceedings of the 18th international conference on World wide web*. ACM, 2009, pp. 791–800.

[29] Y. Zheng, Q. Li, Y. Chen, X. Xie, and W.-Y. Ma, "Understanding mobility based on gps data," in *Proceedings of the 10th international conference on Ubiquitous computing*. ACM, 2008, pp. 312–321.

[30] Y. Zheng, X. Xie, and W.-Y. Ma, "Geolife: A collaborative social networking service among user, location and trajectory." *IEEE Data Eng. Bull.*, vol. 33, no. 2, pp. 32–39, 2010.

[31] A. Marzal and E. Vidal, "Computation of normalized edit distance and applications," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 15, no. 9, pp. 926–932, 1993.