Towards an Analytical and Operational Trajectory Framework

Damião Ribeiro de Almeida[1], **Aillkeen Bezerra de Oliveira**[1], **Samuel Pereira de Vasconcelos**[1], **Fabio Gomes de Andrade**[2], **Cláudio de Souza Baptista**[1]

[1]Information Systems Laboratory (LSI) – Federal University of Campina Grande (UFCG)
Caixa Postal 10.106 – 58.109-970 – Campina Grande – PB – Brazil

[2]Federal Institute of Paraíba (IFPB) – Cajazeiras, PB – Brazil

damiao@copin.ufcg.edu.br, aillkeen.oliveira@ccc.ufcg.edu.br

samuel.vasconcelos@ccc.ufcg.edu.br, fabio@ifpb.edu.br

baptista@computacao.ufcg.edu.br

Abstract. *In recent years, many research works involving trajectories have focused on information representation, storage, semantic data enrichment, transaction processing, and analytics. Moving objects include the modeling of person, animal, vehicle, vessels, airplanes, and natural phenomenon trajectories. Trajectory OLAP systems use Data Warehousing concepts to provide diagnostic, predictive, and prescriptive analytics on moving objects. On the other hand, trajectory OLTP systems provide descriptive analytics on moving objects. Both systems deal with data streaming as object location changes through time. In order to encompass both requirements of trajectory processing and analytics systems, we propose in this paper a Trajectory Analytical and Operational framework. Our framework enables the processing of continuous queries both at operational and analytical levels, providing results in real-time.*

Resumo. *Nos últimos anos, muitos trabalhos de pesquisa envolvendo trajetórias focaram na representação de informações, armazenamento, enriquecimento de dados semânticos, processamento de transações e análise dos dados. Objetos móveis incluem a modelagem de trajetórias de pessoas, animais, veículos, navios, aviões e fenômenos naturais. Os sistemas OLAP de trajetórias usam conceitos de Data Warehousing para fornecer diagnósticos, análise preditiva e prescritiva sobre objetos móveis. Por outro lado, os sistemas OLTP de trajetórias fornecem análise descritivas sobre objetos móveis. Ambos os sistemas lidam com o fluxo de dados conforme a localização do objeto muda ao longo do tempo. Visando os requisitos de sistemas de processamento e análise de trajetórias, propomos neste artigo um framework Analítico e Operacional de Trajetórias. Nosso framework permite o processamento de consultas contínuas tanto em nível operacional quanto analítico, fornecendo resultados em tempo real.*

1. Introduction

Trajectory data management has become an important aspect of many real world applications, being applied to many domains including the modeling and analysis of

moving patterns of pedestrians, cars, vessels, airplanes, animals, and natural phenomena. Recent advances in wireless communication and sensor technologies, and cost reduction on storing and processing of big data have contributed to the development of applications that improve trajectory data management. Trajectory data can be represented as a list of time-ordered geographic points denoted by $T =< id, ((x_1, y_1, t_1, c_1), (x_2, y_2, t_2, c_2), ..., (x_n, y_n, t_n, c_n)) >$, where the id contains the moving object identifier, x_i and y_i are pairs of coordinates that represent the moving object location at the time instant t_i, where $t_1 < t_2 < ... < t_n$, and c_i represents context. Context is known as well as aspect.

Traditionally, there are two data processing methods: OLTP (Online Transaction Processing) and OLAP (Online Analytical Processing). OLTP addresses transaction processing at the operation level of the organization, whereas OLAP focuses on the strategic level to assist the decision-making process. More recently, Hybrid Transaction/Analytical Processing (HTAP) has emerged to encompass in a unique framework both OLTP and OLAP methods.

Real-time analytics applications have become ubiquitous. Risk analysis, recommendation systems, and price analysis are examples of such applications. These applications are usually deployed in distributed systems to cope with transaction processing, high throughput, data streaming, historic and windowing queries. This paper presents MobHTAP, an HTAP trajectory system that deals with data streams of moving objects at the operational and strategic levels. MobHTAP is based on a distributed architecture using horizontal scalability and load balancing and a distributed spatial database management system. The paper contributions include: a) to the best of our knowledge, MobHTAP is the first HTAP trajectory system to be proposed so far; b) MobHTAP supports both OLTP queries and OLAP queries over trajectory data streams; and c) MobHTAP supports SQL-like requests expressing both snapshot or continuous queries.

The remainder of this paper is structured as follows. Section 2 discusses related work. Section 3 presents the MobHTAP system framework, its ETL and the data storage processes. Section 4 addresses query processing. Section 5 highlights a case study. Finally, Section 6 concludes the paper and provides further research to be undertaken.

2. Related Work

Trajectory systems at the operational level deal with high throughput, without data aggregation. Queries are usually posted on each trajectory individually, as for example, current object location, moving object path, and the places visited. Trajectory data may be gathered in real-time, resulting in a spatio-temporal data streaming. [Zheng et al. 2010] infer the moving object transport type based on speed, acceleration, direction, and stop rate along the trajectory. SeMiTri [Yan et al. 2011] uses the map-matching algorithm at the geographical road map to infer user transport type.

The CRISIS system is an operational maritime navigation application that works with trajectory data streams. Data is gathered from several heterogeneous sensors and integrated into a framework that uses Semantic Web concepts to embed context, focusing on interoperability and knowledge discovery on the environment to be monitored [Dividino et al. 2018]. The Baquara conceptual framework provides an ontology and a conceptual model to accommodate the processing of semantically enriched trajectory data

[Fileto et al. 2015]. The CONSTAnT is another conceptual data model to represent semantic trajectories [Bogorny et al. 2014]. The CONSTAnT is divided into two parts. The first part models the simplest entities that contain information about the moving object, the trajectory, the sub-trajectories, semantic points, environment, places, and events. The second part models the complex objects in which data mining techniques are needed to forecast an outcome, such as the moving object's destination, transportation means, and behavior. The MASTER conceptual model uses real-world aspects to enrich trajectory data analysis. The MASTER conceptual model is a generic approach, where an aspect is an entity that contains a list of attributes [Mello et al. 2019].

Trajectory analysis at a strategic level offers information that may help decision makers on discovering patterns, insights, and foresights, such as detecting traffic jams, predicting traffic, and finding movement patterns [Alsahfi et al. 2020]. In a TDW (Trajectory Data Warehouse), the data is usually summarized either in spatial regions called cells or spatial segments, such as the city street map [Leonardi et al. 2014]. The cell-based approach presents a broader view of the spatial region under analysis, while the segment approach enables a summary analysis of the routes traveled by moving objects. The cell approach may offer an overview of a given region's traffic density, while the segment-based approach provides lower-level details. Thereby, it is possible to answer analytical queries such as: *which city regions have the highest traffic?*, *what is the average time that objects take to cross a given road?*, *what are the areas where moving objects travel at lower speeds?*

[Orlando et al. 2007] use a spatial grid to summarize trajectory data. The authors modeled a star schema to implement a TDW composed of three dimensions and one fact table. The fact table contains the number of objects that crossed the spatial grid's cell borders. SWOT [da Silva et al. 2015] is a conceptual model for TDW that represents the trajectory summarization in spatial regions. The model is composed of two layers: consensual and interpretive. The consensual layer is composed of a fact table and dimensions (space, time and trajectory). The interpretation layer represents semantic aspects of the trajectory. T-Warehouse [Leonardi et al. 2010] uses the Visual Analytics Toolkit system (VAToolkit) [Andrienko et al. 2007], which enables the analysis of multidimensional data in a raster map format. The map is divided into several cells (rectangles), giving a spatial grid, and each cell contains the measures: average speed and number of moving objects. Andrienko and Andrienko [Andrienko and Andrienko 2013] propose an analytical approach for a movement called Bird's-eye View on Movement, which consists of a generalization and aggregation strategy that enables an overall view of the spatial and temporal distribution of multiple movements. [Leonardi et al. 2014] present a conceptual TDW model that includes summarized trajectories both in cell and segment format. The Mob-Warehouse presents a TDW with a fact table containing two measures: duration and distance [Wagner et al. 2013]. [Fileto et al. 2014] present a logic model for a Data Warehouse based on the Mob-Warehouse model that includes fact tables based on both cell and segment.

Although many of the previously mentioned research above focused only on trajectory data store or TDW, they do not deal with trajectory data streams. Moreover, their solutions manage static data, and are not designed to cope with analytical queries on trajectory data streams, and do not support continuous queries. STAR is a system that

incorporates some of these requirements [Chen et al. 2020]. It consists of a DSWS (Distributed Stream Warehouse System) aiming at providing ad-hoc aggregate continuous queries over spatio-temporal data streams. Nonetheless, the analysis is made on micro-texts from georeferenced tweets and not from trajectories themselves. In addition, queries on STAR are delimited by a geographic region, for example: *which electronics are the most talked about in the Singapore region?* In our proposal, we expand this query type to enable users to discover regions of interest, as for example: *which regions comment most on the 'smartphone' electronic device?* Table 1 shows a comparison of the state-of-the-art on trajectory systems. This table shows how each application fulfills the requirements of an HTAP application for trajectories.

Table 1. Comparison of trajectory systems.

	TQ[1]	OLAP Query	CTQ [2]	Continuous OLAP
STAR [Chen et al. 2020]		X		X
CRISIS [Dividino et al. 2018]			X	
MASTER [Mello et al. 2019]	X			
Baquara [Fileto et al. 2015]	X			
CONSTAnT [Bogorny et al. 2014]	X			
MoB-Warehouse [Wagner et al. 2013]		X		
SEMITRI [Yan et al. 2011]	X			
MobHTAP	X	X	X	X

[1] Transactional Query.
[2] Continuous Transactional Query

3. The MobHTAP Framework

Big Data is an inherent characteristic of most trajectory systems. Thus, many studies have opted for a scalable and distributed infrastructure to manipulate such data [Özsu and Valduriez 1999]. Many existing technologies can be used in different modules of a given distributed architecture. MobHTAP framework is divided into five modules as presented in Figure 1: client, trajectory data stream processing, routing, processing and storage managers.

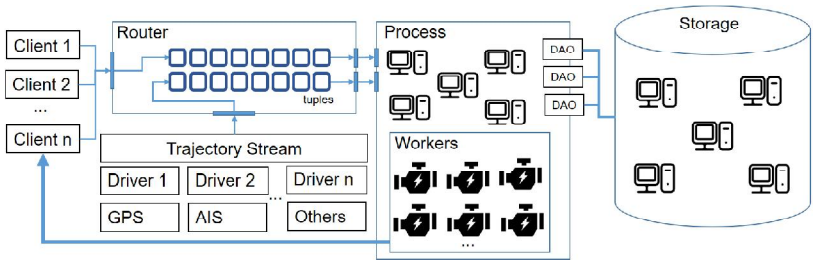

Figure 1. MobHTAP overall framework.

3.1. Trajectory Data Stream

MobHTAP has two entry points: trajectory data and user queries. The *Trajectory Stream* module is responsible for loading trajectory data into the framework. The production of

the stream and the format of the input data may vary depending on the type of sensor used. For example, trajectory data stream collected by GPS from users in an urban environment may contain information that is not present in data stream produced by AIS devices on vessels at sea and vice versa. Thus, the *Driver* module is responsible for transforming heterogeneous data sources into the data structure understandable by the framework. The input attributes of the *Trajectory Stream* interface are raw data and context data. Raw data are the common attributes for all types of moving objects (x, y and time) and the values for these fields are mandatory. Context data value varies according to the type of application. It can be physiological data, preferences, activities, means of transport, etc. In this case, the interface receives as input a list of context information in *LISTOF (context)* format, where each context is composed of a tuple <*name, type, value* >, where: *name* matches the name of the context; *type* corresponds to the type of context information, which can be number or text; and *value* corresponds to the context value. After capturing the information, the *Trajectory Stream* module transmits the data to the routing module, which will forward the message to the processing module.

3.2. Client and Router Modules

The *Client* module enables users to pose spatio-temporal queries on trajectories to the MobHTAP system. These queries are written using a non-procedural SQL like language with support for continuous queries and geographic summarization of trajectory data. The *Router* module is responsible for managing the incoming information workload and sending it forward to further processing. Messages are received in chronological order and are temporarily stored in a queue data structure.

3.3. Process Module

The *Process* module is responsible for the ETL (Extraction-Transformation-Loading) process. The first activity of the ETL process for the underlying trajectory data consists of extracting derived information from raw data. The derived information are: moving object speed, direction, duration and distance between the current location and the previous one. After that, the next activities are cleaning, compressing, and sending the trajectory data to storage in the distributed database system.

The activity of cleaning the trajectory data basically consists of removing the outliers. That is, the points that are outside the trajectory of the mobile object and occur due to some noise in the communication between the mobile device and the data gathered from sensors. There are several algorithms for removing outliers in trajectory data [Zheng 2015]. However, most of them are memory-based solutions. As MobHTAP works with data streams, it is necessary to identify whether a given point is an outlier based only on a few previous points. We accomplished this task using [Potamias et al. 2006] strategy, which consists of removing the points whose speed is higher than a predetermined threshold value. After the cleaning phase, the trajectory points are compressed to improve storage requirements. If the object remains moving in the same direction and the distance between the current point and the previous point is less than a certain threshold, then that current point can be discarded without having a major impact on the characteristics of the trajectory.

The next step is to check whether the moving object is stopped. This check is somehow complex because the moving object may be stopped, but the GPS may transmit

some noise that gives the impression that the object is moving. To check the stop points on the trajectory in real-time, we check if the speed of the point is below a certain threshold, for example, 1 meter per second. After that, the trajectory point, together with all its calculated information, is sent to the DAO (Data Access Object) that will be responsible for persisting these entities in the distributed database management system.

The *Workers* are processes that run in background and are responsible for processing continuous queries of the users. These processes are detailed in section 4.

3.4. Data Storage

In the distributed database, the data is organized into different schemas according to the trajectory stream type. The data schemas are created when the application is initialized. Each driver in *Trajectory Stream* has a configuration file containing information such as the application name, the max speed threshold for detecting outliers, the min speed threshold for stop condition, and the max bearing for detecting points with the same direction.

This strategy helps to organize the data and improves the query processing time. After performing the ETL process, the raw trajectory data is transformed and divided into three groups: raw data (location coordinates and timestamp); derived data (speed, direction, stop points, trajectory identifier); and context data (it varies according to the application domain: means of transportation, temperature, wind velocity, etc).

Raw and derived data are common attributes for all trajectory analytical applications. The database uses two methods of data storage: real-time and history. When the DAO (Data Access Object) receives the message to be saved in the database, it saves simultaneously in the real-time and in the historical databases (see Figure 2). In the real-time database, only the moving object's last location is stored. Continuous queries are carried out on a real-time database and past queries on a historical database. The system has a pre-processed script to create a new schema for each application. This script is then modified to receive the context information and prepare an environment to store the new data from the incoming stream.

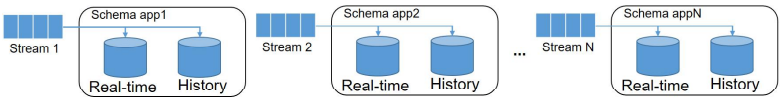

Figure 2. Distributed data tier

4. Query Processing

Spatial queries in MobHTAP are posed using the SQL language with support for the objects and functions specified by the OGC (OpenGIS Consortium)[1]. However, for the continuous aggregate queries we extended the SQL standard to perform analysis on summary trajectory data. According to Trajectory Data Warehouse research, trajectory data may be summarized in two possible ways: geographic regions and segments. The *Process* module manages the queries and reserves a *Worker* (see Figure1) that will be in the background executing the query provided by the user.

[1]https://www.ogc.org/

The aggregation of data by region may be, for example, an administrative limit (district, city, state, country, etc.) or by a grid of spatial cells of fixed size. Aggregation by segment involves summarizing data by road, river or air routes, for example. Currently, MobHTAP works only with cell summarization. During the elaboration of the query, users may specify the size of the grid and the type of relationship between the grid and the trajectory data stream. The cell configurations are specified by the GRID function that receives as parameter the geographic coordinates of the bounding box (x_{min}, y_{min}, x_{max}, y_{max}), followed by the width and height information of each cell, where the data summarization will be performed. For example, the query below calculates the number of moving objects in each cell that uses the bicycle means of transport.

```
SELECT g.id, count(loc)
FROM app.realtime loc, GRID (115.9, 39.7, 116.7, 40.3, 0.2, 0.2) g
WHERE loc.transport_mean = 'bike' AND ST_Contains(g.geom, loc.geom)
GROUP BY g.id STEP 10 min RANGE 20 min
```

To build the cell summary query, MobHTAP implements an algorithm (Algorithm 1) that dynamically creates the cell grid. The algorithm receives as input: the *rawSql*, the schema, and the SRID. The *rawSql* input stands for the user's OLAP query, the schema input stands for the database schema name, and the SRID input is the spatial reference identifier. In lines 1 to 4, a set of matchers is extracted from the GRID, and a table is created to insert the resulting GRID polygon. Then, in lines 5 to 8, for each matcher in the matchers set, the following parameters are extracted: the coordinates of the lower leftmost $bound(x_1, y_1)$, the coordinates of the upper rightmost $bound(x_1, y_1)$, the base size of each *cell(cell_base_size)*, and the height of each *cell(cell_height_size)*. Then, in lines 9 to 22, all GRID coordinates are iterated, and a polygon is created with the current coordinate and the coordinates extracted earlier. The polygon is stored in the table created in lines 3 and 4. In line 17, the parameter *rawSql* is updated, replacing the matcher occurrence by the table name created in line 3. In line 23, the algorithm returns the new *rawSql*.

When MobHTAP receives a continuous query in the processing module, a *Worker* is allocated to transform the query into the language understandable by the distributed database. The *Worker* transforms the GRID into a temporary spatial table containing a spatial column which contains cells.

In addition to the GRID function, the query has an operator called STEP. This feature is used by applications of continuous query [Chen et al. 2020, Dividino et al. 2018] to inform that the result of the query must be updated every certain period of time, which in this example is 10 min. When the client sends a query to the *Router* module, it also sends the url address and the port through which MobHTAP should send the response data stream. Thus, when a query is completely executed, a *Worker* serializes the result and send the response directly to the client. The RANGE operator informs the sliding window size. For example, if the RANGE is 20 minutes, the query will act only on the trajectory data stream received in the last 20 minutes.

Thus, when each *Worker* receives the result of the query, it serializes and sends the response directly to the client. The RANGE operator informs the sliding window size.

Algorithm 1 Translate Query Algorithm

Require: $rawSql$, $schema$, $SRID$

$Matchers \leftarrow$ call function extractGridMatcher(rawSql)
for each $matcher$ in $Matchers$ **do**
    table_name $\leftarrow$ concatenate schema, the string ".tb_grid_", and system time in miliseconds
    call function createTable($table\_name$)
    $x_1, y_1 \leftarrow$ call function getLowerBoundCoordinates($matcher$)
    $x_2, y_2 \leftarrow$ call function getUpperBoundCoordinates($matcher$)
    $cell\_base\_size \leftarrow$ call function getCellBaseSize($matcher$)
    $cell\_height\_size \leftarrow$ call function getCellHeightSize($matcher$)
    $lower\_coordinate \leftarrow x_1$
    while $lower\_coordinate$ is less than x_2 **do**
        $upper\_coordinate \leftarrow y_1$
        while $upper\_coordinate$ is less than y_2 **do**
            $param\_1 \leftarrow$ lower_coordinate + cell_base_size
            $param\_2 \leftarrow$ upper_coordinate + cell_height_size
            $polygon \leftarrow$ call function createPolygon($lower\_coordinate$, $upper\_coordinate$, $param\_1$, $param\_2$)
            call function insertPolygonIntoTable($polygon$, $table\_name$)
            $rawSql \leftarrow$ replace $matcher$ occurrence in $rawSql$ by $table\_name$
            $upper\_coordinate \leftarrow upper\_coordinate + cell\_height\_size$
        end while
        $lower\_coordinate \leftarrow lower\_coordinate + cell\_base\_size$
    end while
end for
return $rawSql$

For example, if the RANGE is 20 minutes, the query will act only on the trajectory data stream received in the last 20 minutes.

5. Case Study

The MobHTAP framework is mainly composed of a stream management system, a distributed processing system, and a distributed database. Different technologies already perform these functions, and that can be incorporated into the framework. In this case study, the MobHTAP framework is based on the following technologies: the *Client* model and *Worker* were developed in Java language version 1.8. The Apache Kafka framework[2] composes the *Router* module. Apache Storm[3] aids in distributed processing at the *Process* module. The *Data Storage* is composed by the distributed data management system CockroachDB[4]. To test our framework, we used a trajectory simulator for people located in the region of George Mason University, Virginia, USA. The simulator developed by [Kim et al. 2020] simulates human mobility with a focus on three basic needs: physiological, satisfaction, and acceptance.

[2]https://kafka.apache.org/
[3]https://storm.apache.org/
[4]https://www.cockroachlabs.com/

To run the case study experiment, we used the MobHTAP framework to simulate the behavior of 2000 people. Besides that, we used as a configuration the constant speed of approximately 4.5 km/h. We ran the framework in a cluster comprising five machines, and we used all machines for distributed processing and data storage. Two of these machines also have the role of distributing the data flow by the Router layer, as described in section 3. All computers in the cluster have the Linux Ubuntu operating system installed, and the machines have the following hardware configuration: CPU intel core i7 3.40 GHz with 8 cores, one machine with 24GB of RAM and 2TB of storage, and the others have 16GB of RAM and 1TB of storage.

In addition to some context information present in the synthetic database (such as type of activity and drowsiness), we used the OSM[5] to capture information from the PoI visited by the agents. In this synthetic base, some agents contaminated by a contagious disease, such as COVID-19, were also simulated. The objective is to show how MobHTAP can help to monitor and assess the geographic scenario in situations that require urgent decision-making.

In this first example (*Query1*), we want to know which places have agglomeration above a given threshold. To answer this question, we used the 2.0 meters radius distance per person as social distance. Just to optimize query processing, we approximated the 2.0 meters radius circle to a 4.0 meters side square, resulting in a square of 16 m^2 per person. Besides that, we used a continuous query on trajectory streams, as shown in the *Query1* below. We can observe that the query produces an output stream with an update rate of 5 minutes (STEP) and a sliding window of 15 minutes (RANGE).

Query1: Which places have agglomeration?

```
SELECT place_id, count(s.user_id) as people, o.area/16 as limitMax, o.area
FROM gmu.stream s, context.tb_osm o
WHERE st_within(s.geom, o.geom)
GROUP BY s.place_id, o.area
HAVING count(s.user_id) >o.area/16
STEP 5 min RANGE 15 min
```

The result of *Query1* is a data stream containing the PoI identifier value in the OSM, the number of people in the location, the maximum limit according to the distance rule, and the PoI area. Figure 3 shows a sequential sample of images taken from a GIS where the result of *Query1* was graphically displayed. Although the query has a 5-minute STEP, we highlight in Figure 3 images with a one-hour interval to highlight the evolution of agglomeration over four hours. The red places highlighted are the PoI that exceeded the agglomeration threshold.

In *Query2*, the user wants to know which areas have the highest concentration of sick people. The GRID function, performed in section 4, summarizes the data in spatial cells at 5-minute intervals, as specified by the STEP clause, and a 10-minute sliding window (RANGE 10 min). The result of the query is a data stream containing the cell identifier and the number of sick people inside it (sick = true attribute).

Query2: Which areas have the highest concentration of sick people?

```
SELECT g.id, count(s.user_id)
```

[5]https://www.openstreetmap.org

Figure 3. The highlight of crowded places

FROM gmu.stream s, GRID (-77.3192, 38.8235, -77.2962, 38.8365, 0.0005, 0.0005) g
WHERE s.sick = true AND st_within(s.geom, g.geom)
GROUP BY g.id
STEP 5 min RANGE 10 min

Figure 4 highlights four maps of the studied region. Each map is divided into cells as specified in *Query2*. The result of *Query2* was reproduced in a GIS and the cells changed their shade of red according to the number of sick people. Thus, the darker the cell, the greater the concentration of sick people. Figure 4 highlights only four specialized images of the *Query2* output stream. The images show a greater concentration of infected people in the eastern part of the region.

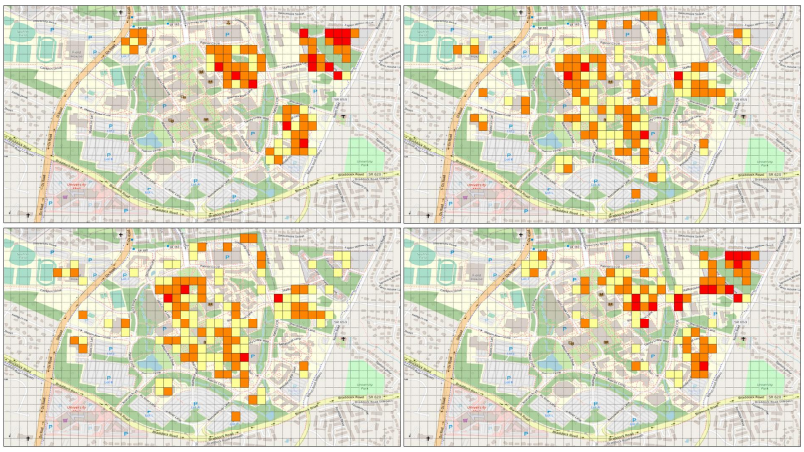

Figure 4. Cell maps highlighting the number of patients

The processing time for extracting information derived from the raw data is approximately 6 milliseconds. The average processing time for *Query1* is 3.27 seconds and

3.23 seconds for *Query2*.

6. Discussion and Conclusions

In this paper we present the MobHTAP system that enables continuous aggregate and historical queries on trajectory data streams. We adapted the SQL language to be able to express queries that use spatial summarization on real-time trajectory streams. We describe the ETL process for transforming trajectory data to be queryable using our query language, as well as describing how that data is stored.

As a future work we intend to expand the possibility of spatial summarization for cell and segment, and thus be able to summarize the trajectories through linestrings. We also intend to evaluate the framework using multi-aspect semantic trajectories, such as weather, rating, price, and transportation information. We also intend to develop a graphical interface to assist the user in building queries. Finally, performance tests are going to be implemented in order to assess MobHTAP workload capacity and evaluate how many tuples may be processed in a second and how many queries the system supports.

Acknowledgements

The last author would like to thank The Brazilian Research Council - CNPQ for partially funding this research.

References

Alsahfi, T., Almotairi, M., and Elmasri, R. (2020). A survey on trajectory data warehouse. *Spatial Information Research*, 28(1):53–66.

Andrienko, G., Andrienko, N., and Wrobel, S. (2007). Visual analytics tools for analysis of movement data. *ACM SIGKDD Explorations Newsletter*, 9(2):38–46.

Andrienko, N. V. and Andrienko, G. L. (2013). Visual analytics of movement: A rich palette of techniques to enable understanding. pages 1–27.

Bogorny, V., Renso, C., de Aquino, A. R., de Lucca Siqueira, F., and Alvares, L. O. (2014). Constant - a conceptual data model for semantic trajectories of moving objects. *Transactions in GIS*, 18(1):66–88.

Chen, Z., Cong, G., and Aref, W. G. (2020). Star: A distributed stream warehouse system for spatial data. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 2761–2764.

da Silva, M. C. T., Times, V. C., de Macêdo, J. A., and Renso, C. (2015). Swot: A conceptual data warehouse model for semantic trajectories. In *Proceedings of the ACM Eighteenth International Workshop on Data Warehousing and OLAP*, pages 11–14.

Dividino, R., Soares, A., Matwin, S., Isenor, A. W., Webb, S., and Brousseau, M. (2018). Semantic integration of real-time heterogeneous data streams for ocean-related decision making. *Big Data and Artificial Intelligence for Military Decision Making. STO. https://doi. org/10.14339/STO-MP-IST-160-S1-3-PDF*.

Fileto, R., May, C., Renso, C., Pelekis, N., Klein, D., and Theodoridis, Y. (2015). The baquara2 knowledge-based framework for semantic enrichment and analysis of movement data. *Data & Knowledge Engineering*, 98:104–122.

Fileto, R., Raffaetà, A., Roncato, A., Sacenti, J. A., May, C., and Klein, D. (2014). A semantic model for movement data warehouses. In *Proceedings of the 17th international workshop on data warehousing and OLAP*, pages 47–56. ACM.

Kim, J.-S., Jin, H., Kavak, H., Rouly, O. C., Crooks, A., Pfoser, D., Wenk, C., and Züfle, A. (2020). Location-based social network data generation based on patterns of life. In *2020 21st IEEE International Conference on Mobile Data Management (MDM)*, pages 158–167. IEEE.

Leonardi, L., Marketos, G., Frentzos, E., Giatrakos, N., Orlando, S., Pelekis, N., Raffaetà, A., Roncato, A., Silvestri, C., and Theodoridis, Y. (2010). T-warehouse: Visual olap analysis on trajectory data. In *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, pages 1141–1144. IEEE.

Leonardi, L., Orlando, S., Raffaetà, A., Roncato, A., Silvestri, C., Andrienko, G., and Andrienko, N. (2014). A general framework for trajectory data warehousing and visual olap. *GeoInformatica*, 18(2):273–312.

Mello, R. d. S., Bogorny, V., Alvares, L. O., Santana, L. H. Z., Ferrero, C. A., Frozza, A. A., Schreiner, G. A., and Renso, C. (2019). Master: A multiple aspect view on trajectories. *Transactions in GIS*, 23(4):805–822.

Orlando, S., Orsini, R., Raffaetà, A., Roncato, A., and Silvestri, C. (2007). Trajectory data warehouses: Design and implementation issues. *Journal of computing science and engineering*, 1(2):211–232.

Özsu, M. T. and Valduriez, P. (1999). *Principles of distributed database systems*, volume 2. Springer.

Potamias, M., Patroumpas, K., and Sellis, T. (2006). Sampling trajectory streams with spatiotemporal criteria. In *18th International Conference on Scientific and Statistical Database Management (SSDBM'06)*, pages 275–284. IEEE.

Wagner, R., de Macedo, J. A. F., Raffaetà, A., Renso, C., Roncato, A., and Trasarti, R. (2013). Mob-warehouse: A semantic approach for mobility analysis with a trajectory data warehouse. In *International Conference on Conceptual Modeling*, pages 127–136. Springer.

Yan, Z., Chakraborty, D., Parent, C., Spaccapietra, S., and Aberer, K. (2011). Semitri: a framework for semantic annotation of heterogeneous trajectories. In *Proceedings of the 14th International Conference on Extending Database Technology*, pages 259–270. ACM.

Zheng, Y. (2015). Trajectory data mining: an overview. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 6(3):29.

Zheng, Y., Chen, Y., Li, Q., Xie, X., and Ma, W.-Y. (2010). Understanding transportation modes based on gps data for web applications. *ACM Transactions on the Web (TWEB)*, 4(1).