SPATIO - TEMPORAL SYNTHETIC DATASET GENERATION AND QUERYING

Ramiz MAMMADLİ, Elif Yağmur DURAN
Bilgisayar Mühendisliği Bölümü
Yıldız Teknik Üniversitesi, 34220 Istanbul, Türkiye
{l1118903, 1118071}@yildiz.edu.tr

***Özetçe* —Bu çalışmada ham İstanbul yol haritası üzerinde rasgele olarak belirli sayıda sokak seçilmiş ve ayrı bir katman olarak kaydedilmiştir. Bu sokaklar üzerinde her 40 metreden bir olmak üzere noktalar üretilmiş ve bu noktalara zaman etiketi atanmıştır. Bu noktalar, sokak üzerinde ilerileyen araçların hareket yörüngesini temsil etmektedir. Üretilmiş noktalara, aracın hızına göre, sokakların keskin dönüşleri de göz önüne alınarak zaman etiketi atanmıştır. Üretilen bu sentetik veri seti 3 boyutlu zaman-uzamsal hale getirilmiş ve 3 farklı yöntem ile indekslenmesi gerçekleştirilmiştir. Bu indeksleme yöntemlerinin performansları kıyaslanmış ve indekslenmemiş veriye göre faydaları gözlemlenmiştir.**

Anahtar Kelimeler—Sentetik veri seti üretimi, Zaman etiketi, 3 boyutlu zaman-uzamsal veri, İndeksleme

***Abstract*—In this study, a certain number of streets were randomly selected on the raw Istanbul road map and recorded as a separate layer. On these streets, points were produced every 40 meters and time stamps were assigned to these points. The points represent the trajectory of the vehicles advancing on the street. Time stamps are assigned to the generated points, taking into account the speed of the vehicle and the sharp turns of the streets. This synthetic dataset generated was made 3-dimensional time-spatial and indexed with 3 different methods. The query performance of these indexing methods was compared and the benefits were observed compared to non-indexed data.**

Keywords—Synthetic dataset generation, Time tag, 3-dimensional spatio-temporal dataset, Indexing.

I. INTRODUCTION

Istanbul is known to be one of the most famous and crowded metropolitan cities in the world. Our city, which has been a home to many civilizations throughout its 2500-year history, has managed to withstand housing many people of different backgrounds throughout the years. With every new group of people passing through the city, new roads, streets and neighborhoods were built, and the reconstruction never stopped. For this reason in particular, the map has been evolving for the past 2000 years. Surely, the analysis of such a historically complex map would be just as difficult. In this project, we aimed do exactly that. With the help of our mentor, Associate Professor Utku Kalay, we generated a spatio-temporal database on QGIS and managed to upload that into PostgreSQL environment, with the aid of PostGIS tool. Then we applied certain optimization procedures to our data, in order to make things more efficient. With the scripts we have written here, we managed to add a time factor to the generated routes, considering the certain conditions of the streets. Lastly, we pulled our synthetic data back to PostgreSQL environment to make it 3-dimensional spatio-temporal data, apply various indexing methods and performance analysis. With the analysis we performed, we observed on many different factors, such as,the differences and the pros and cons of different indexing techniques, how they place themselves into the memory, and how PostgreSQL operates on using the indexes, then we made visual representations for our findings.

II. DATASETS AND THEIR RETRIEVAL

The map used in the project was originally planned to be retrieved from 'OpenStreetMap' official website. However, since the size of the map data we will capture is larger than the upper limit of 'OpenStreetMap' determined by its website, this method was abandoned and the OSM-based BBBike API, which is used as an alternative for such scenarios, was used. It should also be noted that both of the above-mentioned resources provide free services. There are 8 different tables in .shp format, namely 'buildings', 'landuse', 'natural', 'places', 'points', 'railways', 'waterways' and 'roads'. The operations were done on the 'roads' table and other Shapefiles were used to increase the visualization in the QGIS environment.

III. CREATING NECESSARY ENVIRONMENT

A. Preparing the PostGIS environment in PgAdmin 4

In the first step, in the PgAdmin4 environment, an empty database was opened using PostgreSQL and the PostGIS extension was connected to this database. It automatically creates a "spatial_ref_sys" table after connecting to the PostGIS database. Next, a "geometries" table was created to represent the geometry variants. Thanks to this table, the environment was prepared for the functions that can be used in the later stages of the project.

B. Preparing the QGIS environment

A connection has been established between PostGIS and QGIS so that the data can be processed. For this, a new project was opened in QGIS and a connection was made to PostGIS via our own username. At first, pre-existing raw data will be added to the QGIS environment and synthetic data will be linked to the PostGIS environment after generating.

In the raw Istanbul data, the data is kept as a 'vector layer'. Therefore, it is necessary to use the 'Vector Bundle'

to add these Shapefiles to the QGIS environment. The data set is generated on the roads shapefile layer, but other layers of the data retrieved from OSM have also been added to the project for visualization.

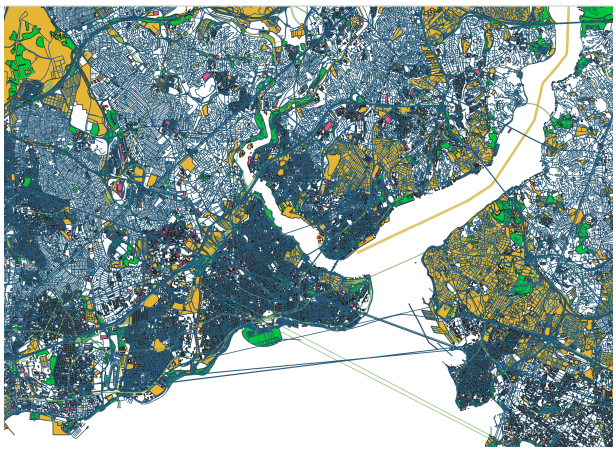


Figure 1 Primary visualization of the data

At first, when examining the 'roads' layer from the downloaded Shapefile files, it can be seen that the CRS format is fixed to WGS 84. QGIS is set to automatically open in WGS 84 mode with 'Global Projection Specification' in every loaded project [1]. If the project was handled on a continental or larger map, it was possible to ignore the CRS difference.

However, since the scope of this project is to work on a smaller area such as the Istanbul map, it is essential to consider the projection difference. In addition, if the project is run in WGS 84 CRS format, the units of measure on the layers will be degrees, which can create certain obstacles in the later steps of the project. In this context, the CRS format of the project was changed to 'TUREF / 3-degree Gauss-Kruger zone 10'. In addition, each previously added layer must also be brought to the specified CRS format. Eventually, both the unit of measurement has been fixed as meters and attention has been paid to set the CRS format to the portion of the map which includes the location of Istanbul as well.

IV. GENERATING AND PROCESSING SYNTHETIC DATA

A. Generating random streets layer out of roads map and Filtering

After the format of the project is set, data generation processes are started. As the route production scenario, it was decided to first select 2500 of the more than 88 thousand streets in Istanbul and save them as a separate layer. To implement this, the 'Random Extract' algorithm in the 'Processing Toolbox' in QGIS was used. This algorithm takes a vector layer and creates a new layer containing only a subset of the features in the input layer. The subset is randomly identified using a percentage or number value to define the total number of features in the subset [2]. Based on these rules, 2500 streets were randomly selected and defined as a new layer, as stated earlier.

Although the streets were randomly selected out of the roads layer, the filtering process must be applied. So that, first, the NULL and unnecessary columns must be deleted from the newly generated layer. In addition, as can be expected, there are also very short (under 80 meters) streets in Istanbul. Including them to this and next steps of the project will not serve the purpose of the study. Therefore, all streets with a length of less than 80 meters were cleared from the generated dataset.

B. Generating Points

Suppose a vehicle is ready for action on each selected street. If a point is generated every 40 meters from the beginning to the end of all routes, these points may represent the trajectory traveled by the vehicle moving on that street. We used the 'Points along geometry' algorithm in Processing Toolbox to generate these points. This algorithm creates a point layer with points distributed along the lines of an input vector layer. The distance between points along the line is defined as a parameter. As might be expected, the points were created every 40 meters on pre-generated routes.

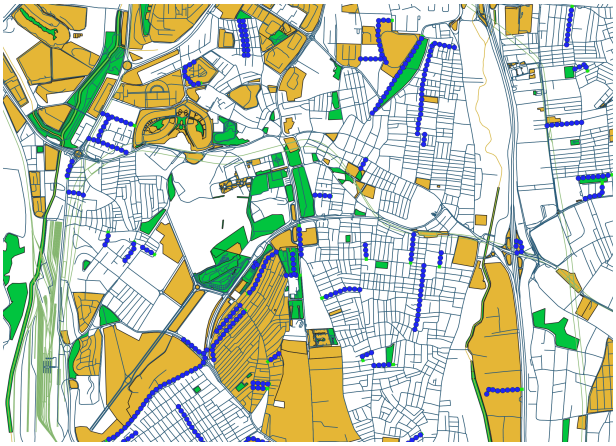


Figure 2 Points generated every 40 meters on pre-randomly selected streets

C. Assigning Timestamps to the Points

In addition to all these operations, it can be assumed that all vehicles are traveling at a certain speed. If we assume that their speed is 20 m/s (72 km/h), the vehicles will arrive from one point to the next in two seconds. Based on this logic, a time stamp showing how many seconds the vehicle arrived at that point can be assigned to each point.

It can be created with the 'Field Calculator' in the 'Attributes Table' of pre-generated points to add a timestamp under the specified conditions. 'Field Calculator' is one of the QGIS features in which code can be written, most useful for generating synthetic data, in the table showing the data of each layer. Algorithms can be developed in the Expression field by using SQL-like but unique functions and syntax. Taking advantage of this feature, the code was developed and the time stamp was assigned to the points.

Additionally, the assigned timestamps travel at a certain speed at each particular interval, regardless of the road conditions. In other words, vehicles do not always move at a constant speed on roads. For example, if there is a certain level of turn on the road, the speed of the vehicle must decrease.

In this context, we also reduced the speed of the vehicle to a certain extent on the roads with curves in our project. In order to understand whether the path with curves, the generated points keep the current azimuth (relative to the North direction of the line) angle of the linestring they are on in the Attributes Table (Figure 3).

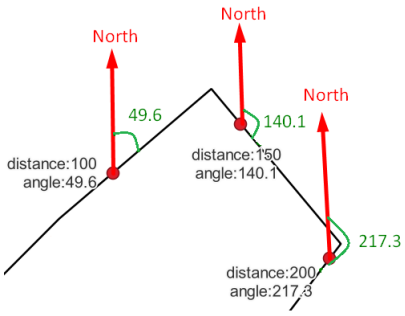
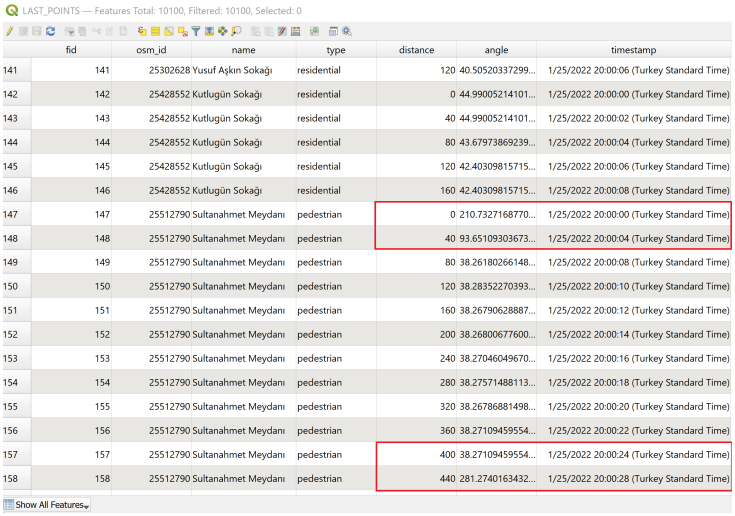


Figure 3 Finding the Azimuth angle of the points based on the line they are on

Based on this information, since each point holds this angle value, if we find the difference between the angles of two consecutive points on a route, we can estimate how curved the road is. That is, if the angle difference between the two points is 40 or more, the speed of the vehicle must be reduced drastically (by half). If the speed is decreasing at a fixed distance, the time taken to travel that distance increases. Therefore, if there is an angle difference above the specified limit, the predefined timestamp on the vehicle's current location will be updated. The algorithm for this case is calculated using the 'Field Calculator' in the 'Attributes Table' as before. As a result, thanks to the code we developed, the return status was checked and the time stamp was updated. In the figure 4, it can be observed that the angle difference between the two points indicated by the red squares is over 40, thus traveling the distance between those two points in a longer time.

D. Adding Time Dimension to Geometry Values

In this part, after the processing of the data in QGIS was completed, a transfer to PostGIS was made for the final adjustments to be made on the data. First of all, two new data sets derived from the 'roads' table on the QGIS side and processed there were added to PostGIS. From these two datasets, the points table was the concern. Since 3D indexing will be done to the data here, we aimed to combine geom and timestamp columns. We used PostGIS functions for this process. The ST_MakePointM function is used to create a new point by accepting X, Y and M coordinates as arguments. Since the aforementioned function accepts all of the arguments as numeric, first of all, the necessary values are brought into the correct formats.


LAST_POINTS — Features Total: 10100, Filtered: 10100, Selected: 0

	fid	osm_id	name	type	distance	angle	timestamp
141	141	25302628	Yusuf Aşkın Sokağı	residential	120	40.50520337299...	1/25/2022 20:00:06 (Turkey Standard Time)
142	142	25428552	Kutlugün Sokağı	residential	0	44.99005214101...	1/25/2022 20:00:00 (Turkey Standard Time)
143	143	25428552	Kutlugün Sokağı	residential	40	44.99005214101...	1/25/2022 20:00:02 (Turkey Standard Time)
144	144	25428552	Kutlugün Sokağı	residential	80	43.67973869239...	1/25/2022 20:00:04 (Turkey Standard Time)
145	145	25428552	Kutlugün Sokağı	residential	120	42.40309815715...	1/25/2022 20:00:06 (Turkey Standard Time)
146	146	25428552	Kutlugün Sokağı	residential	160	42.40309815715...	1/25/2022 20:00:08 (Turkey Standard Time)
147	147	25512790	Sultanahmet Meydanı	pedestrian	0	210.7327168770...	1/25/2022 20:00:00 (Turkey Standard Time)
148	148	25512790	Sultanahmet Meydanı	pedestrian	40	93.65109303673...	1/25/2022 20:00:04 (Turkey Standard Time)
149	149	25512790	Sultanahmet Meydanı	pedestrian	80	38.26180266148...	1/25/2022 20:00:08 (Turkey Standard Time)
150	150	25512790	Sultanahmet Meydanı	pedestrian	120	38.28352270393...	1/25/2022 20:00:10 (Turkey Standard Time)
151	151	25512790	Sultanahmet Meydanı	pedestrian	160	38.26790628887...	1/25/2022 20:00:12 (Turkey Standard Time)
152	152	25512790	Sultanahmet Meydanı	pedestrian	200	38.26800677600...	1/25/2022 20:00:14 (Turkey Standard Time)
153	153	25512790	Sultanahmet Meydanı	pedestrian	240	38.27046049670...	1/25/2022 20:00:16 (Turkey Standard Time)
154	154	25512790	Sultanahmet Meydanı	pedestrian	280	38.27571488113...	1/25/2022 20:00:18 (Turkey Standard Time)
155	155	25512790	Sultanahmet Meydanı	pedestrian	320	38.26786881498...	1/25/2022 20:00:20 (Turkey Standard Time)
156	156	25512790	Sultanahmet Meydanı	pedestrian	360	38.27109459554...	1/25/2022 20:00:22 (Turkey Standard Time)
157	157	25512790	Sultanahmet Meydanı	pedestrian	400	38.27109459554...	1/25/2022 20:00:24 (Turkey Standard Time)
158	158	25512790	Sultanahmet Meydanı	pedestrian	440	281.2740163432...	1/25/2022 20:00:28 (Turkey Standard Time)

Show All Features

Figure 4 Attributes Table of points with timestamps

X and Y values can be translated thanks to ST_X(geom) and ST_Y(geom). Then, thanks to the SELECT EXTRACT function, timestamp values were converted to double precision. Then all the obtained values were combined in the ST_MakePointM function. The query in question is: select ST_MakePointM(ST_X(geom), ST_Y(geom), (SELECT EXTRACT (EPOCH FROM timestamp))) from "LAST_POINTS" To protect the dataset we generated, we created a new table, with every column carrying the same attributes as the one before. Except for the geometry column, which we first dropped from the table, to add a new one of POINTM type (as opposed to the previous POINT type). Then every record from the query in question was inserted here. We observed that we were successful when the geom values of the table and the geom values of the query were compared. After that, the query result was inserted into a new table and the results were indexed.

V. INDEXING

The process of indexing data provides a lot of benefits, especially in larger databases. Without indexing, the data would have to be searched with a linear method in each query. Indexing operations in normal databases are done by placing the values of the selected column in a tree data structure. However, it is not possible to make such an application in spatial databases. Since geometric features are not indexable values, they are usually replaced by bounding boxes, and then these bounding boxes are placed in data structures suitable for spatial data structures. In order to adapt to different needs, different data structures and index types that place their data have been produced. There are 10 different types of indexes that adapt to the geometry type of PostGIS and are still supported by applications today. The first two of these, 'btree' and 'hash' indexes, are prepared for indexing with special cases in geometric columns. The 'btree' index is specific to types of geometry that are sortable, and the 'hash' index is specific to geometry types where each row is known to be distinct. In addition, adding and removing these two index types is not among

the permissions that PostgreSQL gives to the end user. Other indexes can basically be grouped into 3 groups: these are GIST, BRIN and SPGIST. According to the size of the data they index, they are also divided into subgroups such as 2-dimensional GIST indexes and 3-dimensional GIST indexes. The GIST method is an index type that implements the r-tree data structure. The letter G stands for 'generalized', and the GIST method is designed to meet the most general needs, so it is not known to be efficient enough for customized tables. Therefore, it can be said that it is most suitable for uniformly distributed data types. The SPGIST method was designed based on the GIST method, and it is an index type with generalizing features as well. However, unlike the GIST method, it focuses on solving the confusion between overlapping bounding boxes. It uses a quad-tree data structure. The BRIN method is completely different from the other two. It is a method that increases efficiency as tables grow. It is particularly effective in regularly increasing and sequentially placed columns.

VI. EXPERIMENTAL RESULTS PERFORMANCE ANALYSIS

The analysis' were made over the points_3dgeom table, which was the last step result of all the operations done so far. There are about 10 thousand rows in the table. The reason for choosing this size of data is a limitation due to the CPU power of the computers used in the project. Therefore, it should be noted that small changes in the results obtained mean big differences. Also, in some types of indexing, algorithms are designed to minimize the intersection between the bounding boxes created. This may be the reason why a certain type of index is preferred in some program types. However, these factors were not taken into account in this study, as it is known that the table we indexed in our project consists of only points and these points are set so that there is no intersection when creating them. A query that allows spatial indexes to work was chosen for the analysis'. The ST_DWithin function is used because the table consists of points. This function allows to write a query that can return all points at the desired distance to a given point. In the study, firstly, the query was run without indexing on this table for reference. Later, other indexes were tried one by one. And the results were printed on this graphic 5.

It should be reiterated that these results were obtained under certain conditions. First of all, the table size has been tried to be kept small within the constraints of the CPU power of the computers used. In addition, PostgreSQL calculates that it can process faster in certain situations and prefers not to use indexing beyond the control of the user. Sometimes, factors such as the query result being in the cache or the more useful indexes that the user cannot access can affect this selection. Since our study is about examining the features of index types that interest us and in which situations they can be useful, we tried to keep these factors apart as much as possible while obtaining the results. When these other mentioned factors are ignored, the results of the performance analyzes come out as expected. Our observations were:

- It is natural to waste time when querying non-indexed data.

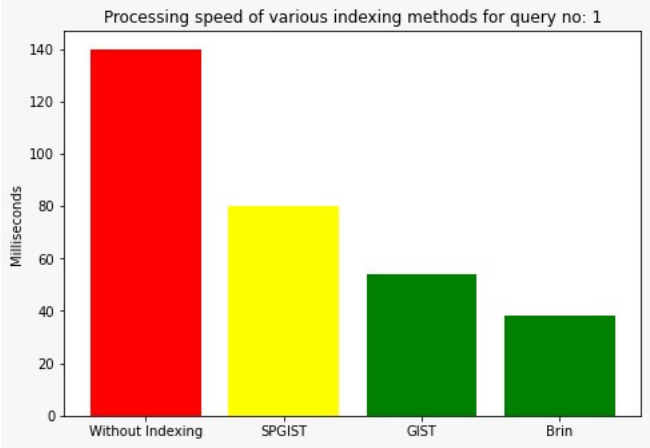


Figure 5 Graphic on query times of different indexes)

- The SPGIST method is the least efficient for our table, because the properties for which the method is customized are not available in our table. It is known that all points have different geom values.
- The GIST method was more successful than SPGIST because it is known that the data is uniformly distributed.
- The BRIN method is the most efficient indexing method for our project, it fits well with our sequential and regularly increasing data.

VII. CONCLUSION

Our project essentialy consisted of 2 parts. First part, various layers of the Istanbul map were taken from the open source in the first half of our project and added to the QGIS environment. So that the necessary environment for synthetic dataset production was prepared. The CRS format and unit of measure of the roadmap were changed to match, 2500 streets were randomly selected and those below a certain length were filtered out. Subsequently, points representing the trajectory of vehicles moving on the route were generated every 40 meters on these routes, and time stamps were assigned according to a certain speed, taking into account the sharp turns of the streets. In the second part, the data processed in QGIS was moved to PostGIS and the process of adding a time dimension to the previously 2D geometry column has been completed. Analysis' were made on the last table obtained thanks to the 'spatial indexing' methods we learned. During these analysis, queries created with PostGIS functions were used. As a result, indexing in large and 3D data proved to be useful again and it was observed that the most appropriate index type of data in the project was the BRIN method. The results were also visualized with the help of Python's Matplotlib library.

REFERENCES

[1] "QGIS Documentation: Working with Projections," https://docs.qgis.org/3.16/en/docs/user_manual/working_with_projections/working_with_projections.html, 2020, [Online; accessed 21-Jan-2022].

[2] “QGIS Documentation: Random Extract,” https://docs.qgis.org/3.16/en/docs/user_manual/processing_algs/qgis/vectorselection.html#random-extract, 2020, [Online; accessed 21-Jan-2022].