Analyzing Index-based Queries for Spatiotemporal Taxi Trajectory Data

Anant Joshi

INTRODUCTION

The dataset used in this project is known as *'Taxi Trajectory Data'* and it contains spatiotemporal data. It offers an insight of the dynamics of the taxi transportation for the city of Porto in Portugal for a duration of one year i.e. from July 2013 to June 2014. Each trip is unique and has a 'TRIP_ID'. There are three types of trips:
Type A - Trips that are requested from the taxi central.
Type B - These trips are directly requested by passengers to a specific taxi driver from a specific stand.
Type C – these trips are requested by passengers to a taxi driver on a random street.

The trajectory for each of these trips is also given in a column named 'polyline'. For a trip, each point for where the vehicle is, at an interval of 15 sec, is given within the polyline column. A column by the name 'TIMESTAMP' contains the time of the start of the trip. All these columns provide us with a robust dataset to extract meaningful information out of it.

As this dataset contains historical data of taxi rides, it can have multiple practical applications, such as:

- The data can be used to predict the time that will be taken by a future trip based on the past trips.
- Or what will be the best route to take, based on the past trips and the routes those rides had to take to reach a particular destination.
- If a tourist lists the places they want to visit, a taxi driver can know beforehand how much fuel they would need based on the distances travelled by the previous driver.
- Looking at the historical data, the government can determine if a new crossing must be made, or a new roundabout must be created for the traffic to be regulated.
- Using the data for rides on a holiday, the rates of the taxis can be regulated based on the number of rides for that day.
- Looking at the data for a taxi being called on a random street. New taxi stands can be created for that area, if there are many such taxis being requested.

The dataset also has a lot of potential to contribute to scientific research, as it can act as potential training data for training of machine learning algorithms. It can be used for testing out pathfinding algorithms or algorithms to calculate the ETA of an object.

In this project the queries that I have built using this dataset are:

- A query to display the trips that have used a particular roundabout just below the famous Casa Da Música in a particular month.
- A query to optimize the total distance/total time and the number of trips that a taxi has done in a month.
- A query to select the top 10 most similar trip, to the first trip of any particular month.

For these tasks efficient index-based queries are made so that retrieval in real life applications can be done easily and efficiently. An index on a column of data that is being used for any query makes the computation time of that query really fast. This helps in completing multiple tasks faster and real-life tasks, such as tasks, that are dependent on the result of that query, can get completed a lot faster.

These efficient index-based queries can be built by using *'PostgreSQL'*. This is done by leveraging the database's indexing functionality on columns that have values that can be indexed, such as:

- For columns that contain spatial data, to retrieve this data again and again it takes a lot of computation to search through all the data again and again, so an index like GIST on spatial data can make that retrieval really fast.
- For other columns such as timestamps B-TREE index can be used.

Using such indexes really helps. Also, if there are many such indexes, PostgreSQL chooses the best combination out of all so that the most efficient combination of multiple indexes on multiple columns can be used.

METHODOLOGY

The dataset is in the form of a csv file. The 'POLYLINE' column contains string values, and each string value contains a list of GPS coordinates for each of the points in that trajectory. Each point is in the form of [LONGITUDE, LATITUDE]. Each point is at a time gap of 15 seconds from the previous point.

As all this is in the form of a string, this had to be converted in the form 'LINESTRINGM(longitude1 latitude1 timestamp1, longitude2 latitude2 timestamp2…)'. This is because PostgreSQL takes in data in this form to convert it into geometry. The third dimension 'M' is for a measure. Here the timestamp for each point is stored as that measure thus making it spatiotemporal data.

There were many other changes done to the input file as well (All these changes were made using python), such as:

- There is a column named 'MISSING_DATA', this contained Boolean values and was true if there was any missing data present. Such rows were dropped.

- Remove rows with POLYLINE containing only 1 point or no points.
- There were some trip ids that were duplicate, so those were removed.

There is a famous concert hall by the name ‘Casa Da Música’. Just below it is a roundabout which is quite big with the GPS coordinates as ‘[-8.6294468, 41.1581764]’. The first query presents 10 trips that use that roundabout in a particular month.

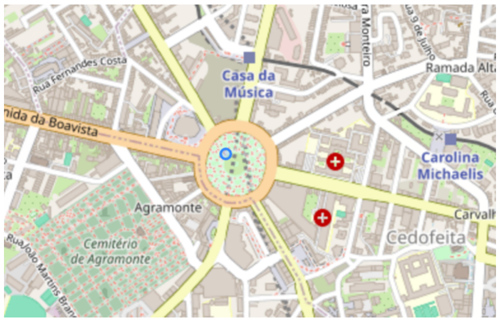

Fig. 1. A picture of the roundabout below Casa Da Música

This is done by the following steps:

- A buffer is created around the roundabout using ‘ST_Buffer(), of 150m, this is done so that any trajectory that intercepts with that buffer in that specified month can be selected. This is creating a box around the point, thus using the range query.
- Then all the trajectories are traversed, using an index (SPGIST, GIST or BRIN), if the trajectory intersects with the buffer at any point, and the point is within that specified month, that trajectory is selected.
- The distance between the roundabout and the closest point of the trajectory is calculated for the ‘closest_distance’.

This query can prove to be really useful, as by viewing the data for rides new improvements to the roundabout, may it be connectivity or road quality that can be changed accordingly.

The second query optimizes the efficiency of a driver, which is the total distance covered divided by the total time for that distance covered in a particular month, and the total trips that a driver has done in a month. This is done by using the *skyline algorithm* [4].

The query is as follows:

- Month_trips CTE: calculates the aggregate statistics for each driver for a specific month. For e.g. it selects a particular taxi_id and calculates the total distance travelled by adding all the trajectory lengths and then calculates the total time by using the third dimension M, of the end and start point of all the trajectories and then adding the difference. It also checks if the start point is not NULL so that it can use that start point.
- Skyline_trips CTE: it calculates the efficiency of a driver by dividing the total distance by the total time. It then filters the data based on the skyline algorithm, i.e the taxi_ids that are dominated are removed. This is done by not selecting a trajectory which is dominated by any point.
- The SELECT statement selects the TRIP_IDs, the efficiency and the TRIP_COUNTS to print them. It also orders them by the skyline rank.

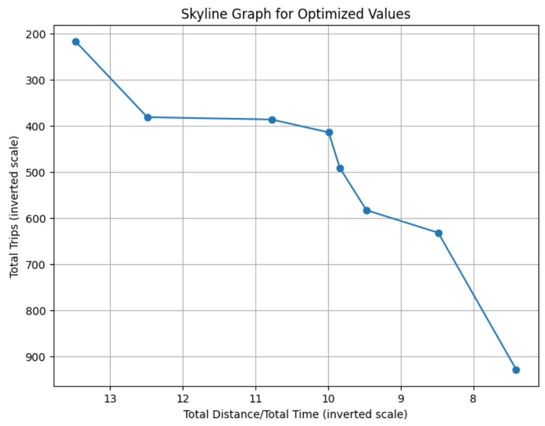

Fig. 2. A graph of the skyline algorithm being used for March

This query can be very useful with respect to practical applications, as it can give the taxi service owners an overview of which taxi is performing the best so as to provide incentives. This can also be used to make the drivers feel more motivated about their performances.

The third query presents the top ten most similar rides to the first ride of a particular month. The similarity of two rides is derived from the calculation of the *Hausdorff Distance.*

$$h(A,B) = \max_{a \in A} \{ \min_{b \in B} \{ d(a,b) \} \}$$

Fig. 3. Formula for *Hausdorff Distance* [5]

Hausdorff Distance is the maximum distance of a set to the nearest point in the other set [5]. This can be used to calculate the similarity between two trajectories. The query is built as follows:

- Month_trajectory CTE: this is used to select the first trip for a particular month, this is done by ordering all the trips for a particular month by the timestamp of the starting point.
- Then the trajectory column is scanned to calculate the Hasusdorff distance between the selected trajectory and all the other trajectories. Such that the difference between the starting points is less than 10m. This is done by using the ST_DWithin() function.
- These selected trajectories are then ordered by their similarity and the top 10 are presented.

Figure 4 show two similar trajectories for the month of march.

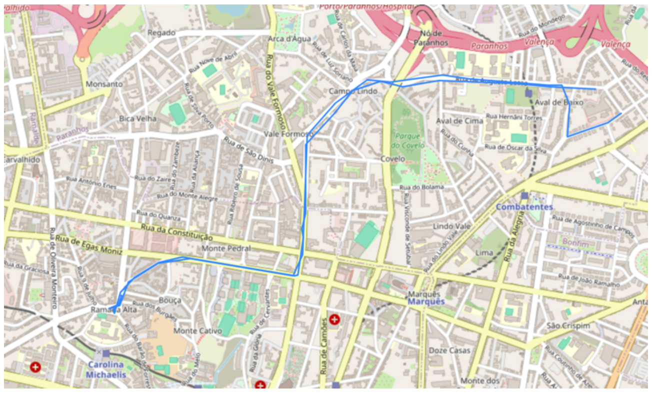
Fig. 4. Two similar trajectories for the month of March

This query has many practical applications too, as we can know about the trips that have started at nearly the same point (due to the condition for starting point) we can know about any frauds taking place due to the detection of difference in time or taking a longer route somewhere in the trajectory.

For each of these queries, there are three test cases that are tested for this project. The three testcases for each of these queries are:

- For the first query, is the ride using that roundabout in the month of January, March or May. So, the month is changed for each of the query.
- For the second query, the skyline optimization for the rides is for the month of January, February, or March. For this query the month is changed for the calculation of skyline optimization.
- For the third query the first ride is chosen for the month of January, February or April. Different test cases select different month for the first ride for selecting the similarity.

For each of these test cases, in each of the query, three different indices are used. These are:

- BRIN: BRIN [3]or Block Range Index was used for the TRAJECTORY column for the first query, and for the START_POINT index for the second and the third query. Both columns contain spatiotemporal columns.

- GiST: GiST [2] or Generalized Search Tree was used for the TRAJECTORY column for the first query, and for the START_POINT index for the second and the third query. Both columns contain spatiotemporal columns.

- SPGiST: SPGiST [1] or Space-Partitioning Generalized Search Tree was used for the TRAJECTORY column for the first query, and for the START_POINT index for the second and the third query. Both columns contain spatiotemporal columns.

EXPERIMENTAL RESULTS AND ANALYSIS

Query Number	BRIN	SPGiST	GiST
Query 1	54.41	65.01	49.01
Query 2	126.54	50,372.34	46,780.34
Query 3	54.17	2,762.84	2,762.84

Table 1. Cost of Index for each query

From table 1 it can be seen that, for the first query GiST has the lowest cost as compared to SPGiST and BRIN. This could be because GiST has a generalized structure which leads to less overhead during indexing. GiST also works really well with spatial data, and as the index was applied to the TRAJECTORY column, which contains LinestringM geometry type.

For the second and third query there is a clear difference between the costs for BRIN as compared with SPGiST and GiST. This can be due to the fact that BRIN has a simpler index structure, that does not store the individual data point but stores the summarized information about the ranges of values.

Query	Testcase	SPGiST	GiST	BRIN
Query 1	1	137ms	280ms	85ms
	2	70ms	144ms	29ms
	3	66ms	90ms	29ms
Query 2	1	3s 90ms	4s 412ms	2s 900ms
	2	3s 29ms	2s 546ms	2s 958ms
	3	3s 305ms	2s 955ms	3s 214ms
Query 3	1	42s 526ms	44s 192ms	40s 9ms
	2	1m 12s 291ms	1m 14s 345ms	1m 8s 294ms
	3	3m 40s 231ms	3m 38s 428ms	3m 40s 449ms

Table 2. Execution time for every testcase using different index

From table 2 and from figure 5, for the first query BRIN index performs the best when executing. This is because BRIN works really well when there are large ranges of contiguous values in the column that is indexed. Since the first query involves filtering trajectories based on how far they are to the roundabout, BRIN performs really well, as the trajectories that pass near the roundabout, form contiguous ranges.

Compared to SPGiST and GiST, which are more suitable for geometric data, BRIN still is better them in this scenario just because of its efficiency in handling range-based queries.

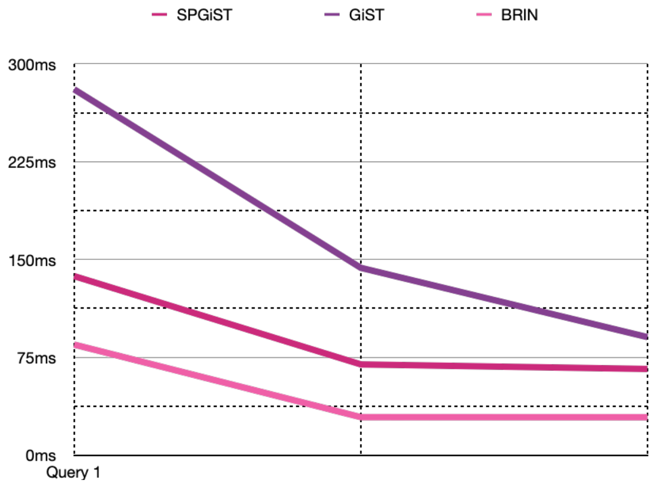

Fig. 5. Comparison of execution time for testcases for query 1

In query 2 as can be seen from table 2 and figure 6, for the first testcase BRIN works better as compared to the other indexes. The reason for this was found in the EXPLAIN part. In the Bitmap Heap Scan it indicates a smaller number of heap blocks and a more efficient bitmap index scan, with few rows removed by filtering, thus making the data for January (testcase 1) more suitable for BRIN. However, in testcase 2 and 3 the Bitmap Heap Scan shows a larger number of heap blocks and more rows removed by filtering, which is less optimal for BRIN and better suited for GiST.

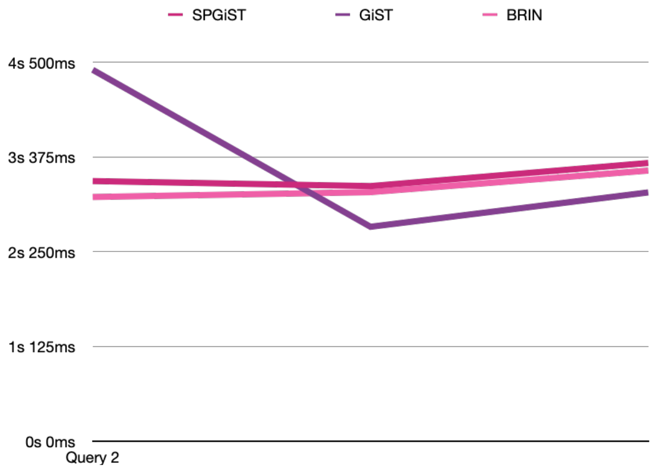

Fig. 6. Comparison of execution time for testcases for query 2

For query 3 BRIN performs worse than GiST in testcase 3, this might be because of the *lossy compression* in BRIN indexes. BRIN indexes use lossy compression to get the summary of the data within blocks, to reduce the storage overhead. In the case where BRIN performs good (testcase 1 and 2), the dataset might involve more predictable patterns, but as the query requires precise comparison, the imprecisions introduced by lossy compression may have very little impact on query performance.

CONCLUSION

In this project multiple queries were built and were then executed using different testcases. Each of these testcases were run using indexes on one or more of the table's columns. The queries used had practical applications and provided with good and expected results. Thorough analysis also showcased the importance of indices in handling spatiotemporal data. By using PostgreSQL was very useful in running these queries and applying indexes to the columns of the data. PgAdmin was useful in running the queries and visualizing the maps and geometries used in this project.

REFERENCES

[1] "SP-gist indexes," PostgreSQL Documentation, https://www.postgresql.org/docs/current/spgist.html (accessed May 9, 2024).

[2] "Gist and gin index types," PostgreSQL Documentation, https://www.postgresql.org/docs/9.1/textsearch-indexes.html#:~:text=A%20GiST%20index%20is%20lossy,by%20a%20fixed%2Dlength%20signature. (accessed May 9, 2024).

[3] "BRIN Indexes," PostgreSQL Documentation, https://www.postgresql.org/docs/current/brin-intro.html (accessed May 9, 2024).

[4] D. Papadias, Y. Tao, G. Fu, B. S. P.-U. Marburg, and B. Seeger, "An optimal and progressive algorithm for Skyline Queries: Proceedings of the 2003 ACM SIGMOD international conference on management of data," ACM Conferences, https://dl.acm.org/doi/abs/10.1145/872757.872814 (accessed May 9, 2024).

[5] N. Grégoire and M. B. Bouillot, "Hausdorff Distance Between Convex Polygons," Hausdorff distance, https://cgm.cs.mcgill.ca/~godfried/teaching/cg-projects/98/normand/main.html (accessed May 9, 2024).