48679097

Efficient Spatial Queries on Taxi Trajectories: A Comparative Study of Indexing Methods

X.Ren

Abstract— **This project focuses on evaluating the efficiency of spatial-temporal queries using taxi trajectory data and comparing the performance of different indexing strategies. This project chose the Porto taxi dataset and designed three practical query tasks: identifying trajectories similar, retrieving trajectories within a certain distance of target path, and detecting shortest trajectory between the certain start and end point. For each query, we tested three indexing methods: sequential scan, R-tree, and SP-GiST. The results showed that different indexes work better depending on the type of tasks. This project helps us understand how to choose suitable indexes to improve query performance on real-world movement data. The results highlight the importance of index-task alignment in spatial query, offering practical guidance for large-scale dataset management such as smart transportation system.**

Index Terms—**Spatial-temporal data, PostgreSQL, trajectory analysis, spatial indexing, R-tree, SP-GiST**

I. INTRODUCTION

Nowadays, with the rapid development of location-based technology such as GPS, vast amounts of spatial data with timestamps are generated on a daily basis. As a result, analyzing such data efficiently has become a critical challenge in the field of country management and intelligent transportation systems.

This project chose the example dataset—Taxi Trajectory dataset, which records one year of taxi trip trajectories in Porto, Portugal, including spatial-temporal data from 442 taxis categorized by dispatch type. In addition, the dataset contains more than 1.7 million records with timestamps, which helps us test different query types and indexing methods effectively.

The project aims to complete database queries by combining different query types with different indexing methods, and to find which indexing method yields higher efficiency and accuracy for each query. These outcomes of could support real-world applications, such as smart transportation systems, route optimization and traffic flow analysis, which are essential for making timely and intelligent decisions.

The objective is to design and implement these query tasks using PostgreSQL/PostGIS, and to evaluate their performance under different access methods, including R-trees, SP-GiST, and sequential scans. Prior studies have shown that different spatial indexing structures yield varying levels of performance depending on the query type and spatial data distribution [1].

In addition, this project focuses on three core spatio-temporal query tasks, which reflect real-world applications in intelligent transportation systems:

(1) Trajectory similarity search, which finds the top-k trajectories most similar to a given target trajectory within the same trip date. This can be particularly useful in emergency situations which need to identify alternative paths immediately.

(2) Spatial range join, which retrieves trajectories within a certain distance of a target trajectory on the same day. This method helps detect areas where many vehicles are concentrated, which could support to deal with traffic congestion.

(3) Shortest trajectory query, which computes the most spatially efficient path between two locations. This query can also be applied in assessing the most energy-efficient paths taken by vehicles across urban areas.

To sum up, these tasks could provide a practical approach for using spatial-temporal query techniques in the real world.

II. METHODOLOGY

This section outlines the implementation of this project and the evaluation of query tasks, covering data preprocessing, query task design, and indexing method implementation.

Fig. 1 Workflow of the query task implementation process.

A. Data preprocessing

The original dataset was imported into PostgreSQL in CSV format and stored in a temporary table named taxi_trip_temp due to existing repeated trip_id. Then, invalid records such as missing or empty trip_id, timestamp, or polyline fields were removed. To prepare the convenience of queries, the polyline field was converted from a JSON array into a LINESTRING geometry using PostGIS functions. Because the dataset is massive, the trajectory_geom column was updated in batches at a time to avoid system overload. Reference [2] shows such batch update methods are recommended in both industry and official documentation when handling large PostGIS datasets. In that case, it enables subsequent spatial queries based on location and latterly analysis. The geometry was saved with SRID 4326, a new created column ‘trip_date’ was added by converting the UNIX timestamp using to_timestamp().

Initially, the dataset contained 1710670 records. After cleaning and filtering, approximately 1704681 valid records remained for further analysis.

B. Query Task Design

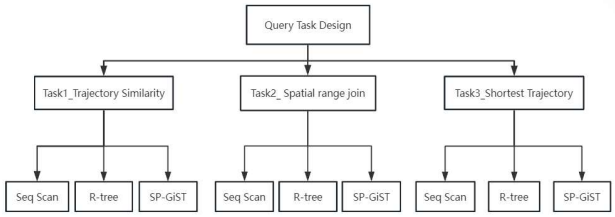

Fig. 2 Query Task Design

(1) Task1_Trajectory Similarity

The query aims to retrieve the top-k paths that are the most similar in shape to the given target track on a specific date. The process consists of two stages: spatial pre-filtering and Frechet distance computing. First, filtering trajectories reduced the number of candidate trajectories before using different indexing methods. Then, the Frechet distance was computed between remaining candidate and the target path which is regarded as trajectory similarity. This query implemented and comparerd the following three indexing methods:

a. Sequential Scan (No Index):

This baseline method used 'ST_DWithin' function as a spatial filter and compute Frechet Distance between the target path and remaining paths on the same date.

b. R-tree (Gist Index):

Firstly, a GiST index was created on the 'trajectory_geom' column to enable R-tree-style access. This indexing method adopted the same filtering strategy and Frechet computation as the sequential query.

c. SP-GiST(Start Point):

Using SP-GiST, the data was filtered based on start point proximity through function ST_DWithin(ST_StartPoint(...)) alternate ways to access the same functionalities noted here).

Query 1: Trajectory similarity search.

Indexing Methods:1) sequential scan, 2) R-tree, 3) SP-GiST.

Test Cases:

1) K=5, target trajectory trip_id = '1403863405620000542',
2) K=10, target trajectory trip_id = '1388517151620000080',
3) K=20, target trajectory trip_id = '1372929767620000304'.

(2) Task2_ Spatial range join

This query aims to retrieve trajectories that fall within a certain distance threshold of the given target on a specific date, without considering path similarity. Inspired by non-parametric density estimation methods [3], we counted the number of nearby trajectories to quantify local density differences more accurately. Three indexing methods were applied and evaluated for this query:

a. Sequential Scan (No Index):

The query performs a Spatial Range Join by using 'ST_DWithin' function as a spatial filter which could exclude trajectories that are too far from the target, on a different date, or with null geometries.

b. R-tree (Gist Index):

This query follows the same logic as the sequential scan version. It filters trajectories based on spatial proximity (ST_DWithin) without ordering the results, and returns the matches.

c. SP-GiST Index (Start & End Points):

Firstly, SP-GiST indexes were created on both start_geom and end_geom which could guarantee the precision of the results. And then, we used two-step spatial filtering strategy to complete the query task.

Query 2: Spatial range join.

Indexing Methods:1) sequential scan, 2) R-tree, 3) SP-GiST.

Test Cases:

1) K=5, target trajectory trip_id = '1403863405620000542',
2) K=10, target trajectory trip_id = '1388517151620000080',
3) K=20, target trajectory trip_id = '1372929767620000304'.

(3) Task3_ Shortest Trajectory

This query aims to find the most spatially efficient trajectory between two locations. To make it more practical, we set the range of the starting and ending point within 100 meters to of the given starting and ending point. To ensure the availability of data, the path length are greater than 0 and the nodes in the trajectory are greater than 1. The results are ordered by total trajectory length, and the top-k shortest ones are returned.

a. Sequential Scan (No Index):

This method uses ST_DWithin() to check whether a trajectory starts and ends near the target points. Then it applies ST_Length() to compute the path length and sorts the results accordingly.

b. R-tree (Gist Index):

This method follows the same logic as the sequential scan, but adds spatial GiST indexes on start_geom and end_geom to optimize spatial filtering.

c. SP-GiST Index (Start & End Points):

SP-GiST indexes were created SP-GiST on start_point and end_point columns. The same filtering logic using ST_DWithin was applied, and the results were sorted by trajectory length.

Query 3: Shortest Trajectory.

Indexing Methods: 1) sequential scan, 2) R-tree, 3) SP-GiST.

Test Cases:

1) K=5 and trajectory with start point (-8.585946,41.148585) and end point (-8.618283,41.165721),
2) K=10 and trajectory with start point (-8.661843,41.176728) and end point (-8.656965,41.177529),
3) K=20 and trajectory with start point (-8.628264,41.157495) and end point (-8.615061,41.152806).

III.) Experimental Results & Analysis

A. Overview of Experimental Setup

To evaluate different indexing strategies for spatial queries, Taxi Trajectory dataset was chosen, which contains around 1.7 million trajectory records with timestamps. The experiments

were run on PostgreSQL 15.12 with PostGIS 3.4.2, using EXPLAIN (ANALYZE, BUFFERS) to examine query plans and collect performance metrics.

B. Query Performance Comparison

(1) Task 1_Trajectory Similarity Search

A target trajectory is compared against others in the dataset using the ST_FrechetDistance function as a similarity measure. Since the calculation is slow due to the massive dataset, adding an index doesn't help much. Therefore, the most important thing is narrow down the results by picking only nearby trajectories.

a. Sequential scan:

Execution Time: The sequential scan performed without any index, the total time ranged from 18,289ms to 18,727ms, with minor variation between cases. The high query time means that sequential scan is slow and not suitable for large datasets.

Buffers Hit: The number of buffers hit stayed at a similar level in all tests.

Buffers Read: Each query incurred significant I/O loads with over 2.31 million buffers read from disk, demonstrating the inefficiency of using brute-force scanning, especially when similarity computation (ST_FrechetDistance) is applied to a large number of trajectories.

b. R-tree:

To better visualize the density of trajectories around the target path, we used Python to generate plots like **Fig.2.**

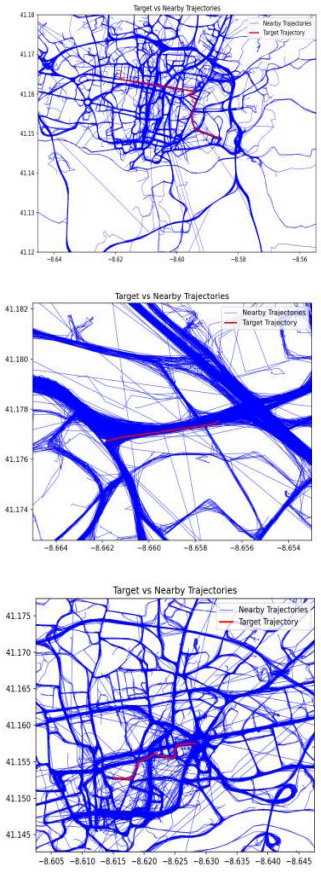

Fig. 3 Target & Local Trajectory Distribution in cases

However, we realized that it's difficult to compare density just by looking at the graphs, so we also counted the number of nearby paths to provide a clearer data-based comparison, since relying solely on visual impressions can be misleading [4], with 565919, 111483, 295756 respectively.

Execution Time: The execution time varied significantly due to the location of the target path. Case 2 completed in 26,250ms, which is nearly twice as fast as case 1(52,943ms) and Case 3 (52,715ms). This shows that the efficiency might be influenced by the density of tracks, even though we used the indexes.

Buffers Hit: The number of buffers hit ranged from 43,717 to 167,716. Like the performance of execution time, case 2 had the fewest buffer hits, which means the query used less memory. This suggests filtering work playing an essential role in preventing to spend waste time scanning unnecessary rows.

Buffers Read: The range of physical I/O was between 576,255 and 1,431,690. R-tree indexing reduced buffers read compared to sequential scans, but only case 2 showed an obvious reduction, which highlights the important of pruning.

Filter: To find the reason for efficiency of case 2, we count the number of the other trajectories within 200 meters. Interestingly, there were only 11483 paths in case 2, which was much less than case 3 with 295756 and case 1 with 565919. This means trip in case 2 was located in an edge or spatially sparse region.

The performance of SP-GiST is illustrated in **Fig. 3.**

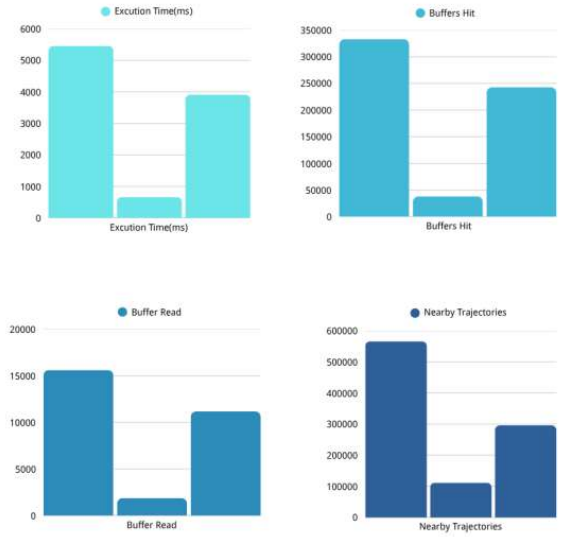

Fig. 4 SP-GiST Index Performance on Task 1 (Case 1–3)

While R-tree indexing provides better I/O efficiency than sequential scans, its effectiveness is highly dependent on the spatial distribution of the target trajectory. So, we need to combine indexing with spatial information when optimizing similarity queries on large datasets.

c. SP-GiST:

Execution Time: There is much difference among the cases on execution times. Case 2 finished in 664ms, which was much lower than that of Case 1 (5459ms) and Case 3 (3911ms), which means the location of the target path, especially the high density of area, plays a major role in impacting on performance.
Buffers Hit: The buffers hit of Case 2 was only 1880, significantly fewer than Case 1 (15605) and Case 3 (11170) which means Case 2 accessed fewer cached pages and memory.
Buffers Read: In terms of physical I/O, Case 2 performed best as well, with only 38,123 read, compared to 332,884 in Case 1 and 242,571 in Case 3. This indicates that spatial index pruning was most effective in Case 2.

Overall, the performance of SP-GiST is better than others, reflecting well-suited for coordinate-based indexing, applying it to the start_geom column significantly reduced buffer usage and improved query speed.

(2) Task 2_ Spatial range join
Efficiency was assessed using Execution Time, Buffers Hit, and Buffers Read in order to evaluate different indexing methods for the Spatial Range Join task.

TABLE I
COMPARISON OF SEQ SCAN AND SP-GIST INDEX

	Seq Scan		SP-GiST	
	Case 1	Case 2	Case 1	Case 2
Time(ms)	25073	24397	2281	916
Hit	2294264	508	146334	59196
Read	31680	268	6840	2763

a. Sequential Scan:
This approach executed the query in 25,073 and 24,397 ms, which is significantly faster than the R-tree method. However, it resulted in a very high number of buffers read (2,279,963), reflecting heavy I/O usage due to the lack of spatial filtering. Despite its relatively low number of buffer hits (45,981), the absence of indexing led to full table scans and large memory access costs and the trajectory was located in a dense area, causing even heavier scan workload.

b. R-tree (GiST):
R-tree indexing produced the longest execution time (49,413 ms), which is likely because of the use of the ST_DWithin function with high burden of calculation. Although it achieved a moderate reduction in disk reads (105,428 compared to 2.2 million in Sequential Scan), the buffer hit count was the highest (1,103,790), suggesting excessive computation. This may be because a large number of filtered trajectories need to be precisely compared.

c. SP-GiST:
SP-GiST outperformed others, completing the query in just 8,433ms. It also achieved the low number of buffer hits (2281) and buffer reads (146,334), highlighting its efficiency in handling point-based spatial joins. In case 2, the nearby paths were relatively fewer than other cases, so the sharp reduction in buffer usage indicates early filtering, which plays an essential role in avoiding unnecessary computation and I/O overhead.

In conclusion, for these three methods, SP-GiST performed the best with the shortest execution time, lowest buffer usage, and lowest I/O cost than both R-tree and sequential scan. This advantage is attributed to the use of point-based coordination. Besides that, this task focuses on identifying spatially nearby trajectories rather than the shape of these, so point-based queries are more suitable than LineString-based comparisons.

(3) Task 3_ Shortest Trajectory
This query aims to identify the top-k shortest trajectories where both the start and end points fall within a specified spatial region. To compare the performance of difference indexing strategies that are Sequential Scan, R-tree (GiST), and SP-GiST. **Fig. 4** summarizes the performance comparison for Task 3.

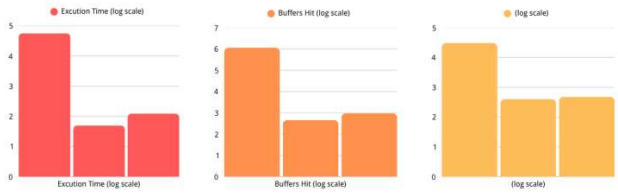

Fig. 5 Task 3: Index Performance Comparison

(Values are log-transformed to improve visual clarity due to large variance.)

a. Sequential Scan:
The Sequential Scan recorded the longest execution time (56,754ms), with over 115 million buffer hits and 330,738 buffers read. Since the entire table must be scanned, the length of every trajectory is computed, leading to high processing overhead, particularly on large datasets. As shown in Fig.4, its bars far exceed others in every metric, confirming its inefficiency.

b. R-tree（GiST）:
The R-tree method completed in 49.805ms averagely, with 455 buffer hits and only 403 reads. This suggests that performing bounding-box filtering before precise distance computation significantly improves query efficiency. The general trend of query efficiency is similar to that of R-tree. From the chart, it can be observed that a dramatic drop in all three bars compared to Seq Scan clearly.

c. SP-GiST:
SP-GiST query performed better than Seq Scan obviously, spending 122ms, with 950 buffer hits and 479 buffer reads. Similar to R-tree method, it created indexes on start and end

point, but its pruning capability was slightly weaker than R-tree. While it was slightly slower than R-tree and showed marginally higher buffer usage.

Overall, the I/O cost of Seq Scan showed the biggest, which means this method is not suitable to be used in big dataset. Although the R-tree indexing structure showed the best performance in terms of speed and efficiency, its effectiveness largely depends on the spatial density of the data. In terms of buffer performance, SP-GiST showed a more stable trend which ensures its performance.

C. Impact of Path Distribution on Indexing Performance

To further explore how indexing strategies respond to spatial distribution, we visualize buffer performance including buffer hits and reads for each indexing methods and cases in Task 2, as shown in Fig.6.

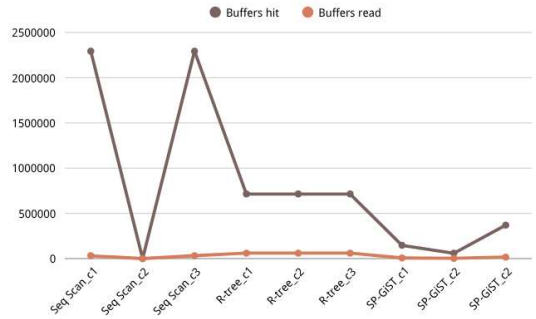

Fig. 6 Buffer Hit and Read Comparison across Index Types and Cases

According to the chart, Sequential Scan is highly sensitive to path density. When the path is located in spatially sparse region, both buffer hits and reads decrease sharply. On the contrary, when the target path located in dense area, like case 1 and case 3, the number of both are higher extremely.
SP-GiST also demonstrates moderate sensitivity to path distribution. It has best performance in Case 2, while its buffer usage increases in case 3 and case 1
R-tree, by comparison, maintains relatively stable buffer usage across all these cases due to the bounding box pruning strategy. It highlights that the uneven spatial and temporal distribution of trajectory data necessitates tailored indexing strategies to maintain efficient buffer utilization [5].

AI ACKNOWLEDGEMENT

ChatGPT-4o (OpenAI, accessed on 12 May 2025) was used to write the construction of the Methodology part of this report. The tool assisted in sorting out my draft. In addition, this tool was used to correct grammatical issues in this report.

REFERENCES

[1] M. Schoemans, W. G. Aref, E. Zimányi, and M. Sakr, *Multi-entry generalized search trees for indexing trajectories*, arXiv preprint arXiv:2406.05327, Jun. 2024. [Online]. Available: https://arxiv.org/abs/2406.05327

[2] devgem.io, *Effective strategies for updating large tables in PostGIS without running out of space*, Jan. 2025. [Online]. Available: https://www.devgem.io/posts/effective-strategies-for-updating-large-tables-in-postgis-without-running-out-of-space

[3] M. Rizk, M. A. Youssef, and A. Elgohary, *Mining spatial trajectories using non-parametric density functions*, in *Advances in Databases: Concepts, Systems and Applications*, Berlin, Germany: Springer, 2011, pp. 483–497. doi: 10.1007/978-3-642-23199-5_37

[4] Z. Bao, H. Cao, R. Cheng, and B. C. Ooi, *Exploring trajectories for efficient top-k spatial keyword search*, in *Proc. ACM SIGMOD Int. Conf. Management of Data*, Melbourne, Australia, 2015, pp. 333–344. doi: 10.1145/2723372.2749433

[5] Y. Yang, X. Zuo, K. Zhao, and Y. Li, *Non-uniform spatial partitions and optimized trajectory segments for storage and indexing of massive GPS trajectory data*, *ISPRS Int. J. Geo-Inf.*, vol. 13, no. 6, p. 197, 2024. doi: 10.3390/ijgi13060197