Project Implementation Report on US Accidents (2016-2021)

Chi Heng Jeffrey Hui, master's student, University of Queensland

I. INTRODUCTION

In the implementation stage, I picked the US Accidents (2016-2021) dataset. Based on the properties and the attributes of the dataset, I created 5 queries. For each query, I implemented two methods. In these implementations, I used 3 distinct algorithms, which are R-Tree, KD-Tree and Linear Scan. After implementing these algorithms on the queries, it generates both qualitative and quantitative information, which helps me to understand the performance and the use case of different indexing techniques. In this report, I am going to analyze the information and report the findings.

II. DEFINITION AND SCOPE

In this project, I choose the US traffic accident dataset. The dataset has different attributes which are used to describe each traffic accident. There are 47 attributes in the dataset. Most data in the dataset are not suitable for this project's purpose or they are filled with a lot of null values.

To make the project more specific, precise and avoid data integrity issue, data is being preprocessed, so it will focus on the subset of the attributes. After preprocessing, the dataset will focus on geographical attributes, timestamp attributes and other single value attributes that can help exploring the indexing algorithms.

Query tasks are not created randomly. Instead, they are focusing on exploring different factors that can affect the query results. For instance, location, time, and severity. The ultimate goals are to use the query tasks to find out correlations and patterns among data points, so that it can provide useful feedbacks and suggestions to resolve impacts brought by traffic accidents.

From the indexing aspect, the intention of this project is to compare brute force algorithms with spatial indexing algorithms. Also, analyzing algorithms performance and efficiency under different scenarios and use them wisely.

Hypothesis: Traditional brute force algorithms (e.g., Linear Scan) has lower efficiency in general, but there are scenarios that brute force algorithms outperform over indexing algorithms (e.g., R-Tree).

III. QUERY TASKS AND ITS VALUES

I have implemented 5 queries. For each query, I implemented two methods.

The first query I implemented is "Given a point, a distance range and a date, find all the traffic accidents that happened within the distance range of the given point which also meet the date requirement." For instance, (-112.074036,33.448376) is the given longitude and latitude of Phoenix, AZ, 5km is the given distance range and 2021-12-25 is the given date. When this information is substituted to query 1, it means find all the traffic accident within 5km of Phoenix, AZ that happened on 2021's Christmas day. Query 1 has practical application values in the industry, especially in the data analysis aspect. We can tune the point, distance range and the date, which allows us to inspect different patterns, for instance, if point location and distance range don't change, and we only change the date. We can identify how different date affect the traffic accident frequencies in a specific region. Alternatively, if date and distance range don't change, only change the point location, we can compare the traffic accident frequencies between different areas. These are some of the practical usages of query 1 that can bring values to the industry.

Query 2 is '' Given an accident point, find its KNN that has the same severity value as the given accident point.'' For instance, if accident A-1 is given, its severity is 3, and the K value is 3. Then you need to find the 3 closest neighbors to A-1 that has a severity of 3. Query 2 also has practical application value in the industry. For instance, if we target A-1, which has severity of 4 and the k value is 3. When we get the three closest neighbors to A-1 that has severity of 4, we can calculate how close are A-1 to these neighbor accident points. If they are close to each other, we can conclude there are multiple serious accidents within short distance of range, which may imply the specific local area may have some issues that we need to resolve, for instance, urban planning issues. Query 2 can help industry professionals to find out hidden issues that are not obvious to the public and solve the problem before causing more troubles.

Query three is "Given a bounding region, find the accidents within the region that last the longest and the accident that last the shortest amount of time." We can utilize query three by plugging in different bounding regions and compare their results. The result of Query 3 gives us the accident that last the longest and the shortest, so we can understand the time range of the accidents in the region. If accidents happen in bounding region A range from 10 mins to 25 mins while accidents happen in bounding region B range from 60 mins to 120 mins. It can tell us bounding region B usually has worse traffic than region A if traffic accidents happen, so this can give industry professionals indications that they need to treat region B serious than region A, for instance, better resources management. The purpose of query 3 tends to compare traffic situations between different regions.

Query 4 is "given a region, find all accidents within the region and find out the number of accidents in each severity category and the overall percentages". Query 4 is different from query 3, query 4 focus on individual analysis, which means we use the detailed information from the query output to analysis the

safety or traffic conditions of the given region. It can bring practical application values to the industry as well, for instance, if the query 4 output on a region tells us severity 4 accident has the highest percentage in the entire searching region, then we know accidents happen in this region are more dangerous or fatal than other regions that have a lower severity value. Detailed analysis on a region is important, since it can tell us more information about the region itself. The messages behind the information can be more specific. Query 5 is "Given a location point, and a distance range. Dissect the distance range to subregions, explore the traffic accident frequencies and pattern within the subregions of a given distance range." For instance, if given point A, and distance range is 6 km. Then you need to find accidents happen between 0-2km distance range to point A, between 2-4 km distance range to point A and 4-6 km distance range to point A. Query 5 can bring practical application value to the industry as well. For instance, we usually think if a location is closer to downtown, there are more traffic accident. But it may not always the case, since downtown usually have more traffic lights and lower speed limit, so the accident frequencies can be lower than other regions that are further from downtown. Subregion analysis can help us analyze subregions, which can help us to find out the truths or real stories behind such misconceptions. These can help industry professionals to understand the traffic around the regions more precisely, so they can apply the right strategies to improve the traffic in the local area.

IV. IMPLEMENTATION STRATEGIES

I used 3 algorithms to solve the queries.

For query 1, I used R-Tree and linear scan to solve the query. Query one involves a lot of filtering criteria, for instance, limit result by distance range and date. R-Tree using MBR techniques, so it can filer points that is far away from the target point. I use R-Tree to index the start longitude and start latitude from the dataset. Afterwards, I use a while loop to increment the k value of nearest neighbor function one by a time. Inside the loop, it will extract the last item (furthest point) at the current k and calculate if the furthest item exceeds the distance range. If previous k doesn't exceed the distance range, but current k exceeds the distance range, which means the previous k value is the maximum possible k that doesn't exceed the distance range. By using this technique, I can take advantage from the efficiency of R-Tree and find the within range accident points in the most efficient manner. Filtering within distance range points is done prior the date filtering step, since filtering within range points can eliminate most points in the dataset, which leave small number of points to the date filtering stage, so it requires less checks, which can improve the overall efficiency substantially.

In query 2 and query 5, KD Tree is used. KD Tree avoids checking every single point in the dataset, so it can get the nearest neighbor results and get the within range points in a more efficient manner.

In query 3 and query 4, R-Tree is used as well, R-Tree intersection function helps finding intersecting points without checking all the values so it can speed up the process, and subsequent filtering can continue as soon as possible, and it also reduces the required checking points significantly.

Linear scan is used widely to solve the queries. As linear scan visits every data point in the dataset, so it takes longer than using indexing algorithms. But there are ways to improve linear scan performance during the querying process. For instance, some steps in some queries require looping through the dataset to find a specific data point. When the data point is found, I utilize the break keyword in Python to end the loop, so that it won't continue the iterations for no meaning. Linear scan requires looping through the dataset or perform operations on every datapoints, but we can use our domain knowledge to avoid unnecessary operations so efficiency can be improved.

V. TECHNICAL INTRICACIES

During the implementation process, I encountered coordinate system issues. R-Tree and KD-Tree are algorithms that based on Cartesian coordinate system, so both algorithms use Euclidean distance as the standard. In the project, most of the queries are related to longitude and latitude pairs, and some queries require calculating distance between longitude and latitude pairs, which use great circle distance as a standard. I tried to understand the class documentation of both algorithms, but there are no ways to change the distance calculation standard in both algorithms. Also, I tried to convert the longitude, and latitude pairs to x, y values on cartesian coordinate system. But it is not feasible, since altitude is not provided in the dataset, so I don't have sufficient information to do the conversion. After consulting with the tutors and Dr. Luo directly, they recommend keeping using Euclidean distance on R-Tree and KD-Tree algorithms and use great circle distance on liner scan implementations. Under this situation, queries that require distance calculations on longitude and latitude will have two sets of SQL statements, one for Euclidean distance and one for great circle distance, so that it can verify my Python implementations results accurately.

VI. RESULTS

Correctness:

In terms of correctness, all implementations are being tested repeatedly. When there are differences between Python implementations and database outputs, codes will be revised detailed until both sides have the same outputs. Among all 5 queries, python implementations and database outputs generate the same results with no errors, so all test cases among the queries obtained a perfect F1 score, which is 1.

Time and Memory cost:

Query 1 Case 1

	Index time (In seconds)	Index memory cost (in bytes)	Query time (In seconds)	Query memory cost (in bytes)
R-Tree	90.771211	892239872	24.646412	232914944
Linear Scan	9.060428	713752576	8.455063	216793088

Query 1 Case 2

	Index time (In seconds)	Index memory cost (in bytes)	Query time (In seconds)	Query memory cost (in bytes)
R-Tree	91.429948	888963072	13.239949	424542208
Linear Scan	9.020343	713408512	8.385083999999999	393216

Query 1 Case 3

	Index time (In seconds)	Index memory cost (in bytes)	Query time (In seconds)	Query memory cost (in bytes)
R-Tree	91.008923	836321280	23.500118	655179776
Linear Scan	9.044628	712884224	8.554835	98304

Query 1 Case 4

	Index time (In seconds)	Index memory cost (in bytes)	Query time (In seconds)	Query memory cost (in bytes)
R-Tree	90.880888	901955584	80.958263	423133184
Linear Scan	9.032562	712540160	8.463758	101531648

Query 2 Case 1

	Index time (in seconds)	Index memory cost (bytes)	Query time (in seconds)	Query memory Cost (in bytes)
KD-Tree	0.094505	43466752	0.001523	435421184
Linear Scan	None	None	7.613861	109150208

Query 2 Case 2

	Index time (in seconds)	Index memory cost (bytes)	Query time (in seconds)	Query memory Cost (in bytes)
KD-Tree	1.652133	1141669888	0.000276	1142472704
Linear Scan	none	none	10.477545	996065280

Query 2 Case 3

	Index time (in seconds)	Index memory cost (bytes)	Query time (in seconds)	Query memory Cost (in bytes)
KD-Tree	1.660319	1508245504	0.000269	1509015552
Linear Scan	none	none	10.434614	999227392

Query 3 Case 1:

	Index Time (in seconds)	Index memory cost(bytes)	Query Time (in seconds)	Query memory cost (in bytes)
R-Tree	89.177596	344064000	0.304879	898138112
Linear Scan	none	none	8.887827	1277378560

Query 3 Case 2:

	Index Time (in seconds)	Index memory cost(bytes)	Query Time (in seconds)	Query memory cost (in bytes)
R-Tree	88.53356	42318848	0.314029	913211392
Linear Scan	None	None	8.809211	1274445824

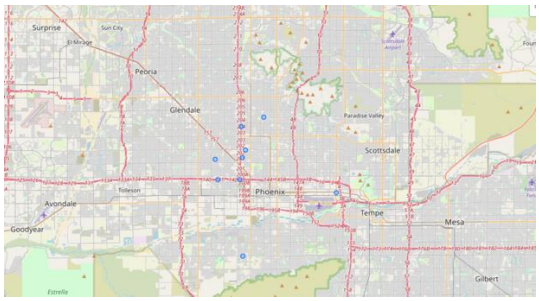
Query 4 case1

	Index Time (in seconds)	Index memory cost (bytes)	Query time (seconds)	Query memory cost (bytes)
R-Tree	89.374658	213975040	0.005738	226263040
Linear Scan	None	None	9.908522	946962432

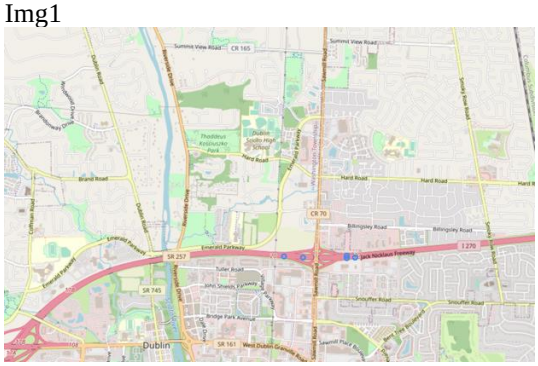
Query 5 Case1

	Index time(second)	Index memory cost(bytes)	Query time(seconds)	Query memory cost(bytes)
KD-Tree	10.336843	381566976	0.001691	383713280
Linear Scan	None	None	11.67908	659636224

Visualizations:



Img1



Img2

Both images demonstrate the application of using ST_Transform in postgis. It not only shows us the results geographically, but it also allows us to validate the results and decide the correctness.

VII. DISCUSSIONS AND FINDINGS

After implementing different algorithms on different queries, it gives me sufficient qualitative and quantitative information to draw more meaning insights. The following are some important takeaways from this project.

Firstly, I realized the importance of using the right standards and using same standards when doing a comparison. Using wrong standards can give you different results on different methods, even both methods are working on the same query and providing the same data inputs.

Secondly, using KD-Tree for indexing is faster than using R-Tree. R-Tree usually takes longer during the indexing process, since it need to find out the best MBR that can avoid the overlapped situation. For KD-Tree, it only needs to assign values base on each dimension and decide if it is larger or smaller than the node values, so the procedure is less complicated than R-Tree constructions, so KD-Tree takes less time when comparing with R-Tree indexing.

Thirdly, R-Tree and KD-Tree do not always give you the best result or performance and it depends on the dataset size, the data points that you are focusing on in the dataset(position) and the frequencies on working with the dataset. In example, the US accident dataset contains more than 2 million rows of data, so we can see it takes a lot of time for indexing (in both R-Tree and KD-Tree). If the dataset is larger, it is going to take more time to index the dataset. In the project, we run test cases separately. According to the table results, we can see the indexing time is way longer than the query time in some cases. If a program has many operations that will access the dataset, indexing will save a lot of time in the long run and long indexing time is just a one-time cost which can be disregarded. But for small applications that do not access the dataset frequently, or even a one-time search, indexing algorithms may not bring the actual improvements to its users, and it may slower or affect the overall efficiency, which makes indexing the dataset is not the smartest decision.

We can also observe the query time has improvements after indexing the dataset when comparing with no indexing. But the improvements may not noticeable sometimes, so users/clients may not even realize the improvements.

According to these important insights, I would say spatial indexing techniques are more efficient in general cases, but there are exceptional cases that linear scan could do better, and it really depends on the context and your usage of the search processes. These important insights and findings prove the hypothesis I made in section II is valid and correct.

References

[1] “Tutorial¶,” Tutorial - Rtree 1.0.1 documentation, https://rtree.readthedocs.io/en/latest/tutorial.html (accessed May 24, 2023).

[2] “Scipy.spatial.KDTree#,” scipy.spatial.KDTree - SciPy v1.10.1 Manual, https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.KDTree.html (accessed May 24, 2023).

[3] “Program for distance between two points on Earth,” GeeksforGeeks, https://www.geeksforgeeks.org/program-distance-two-points-earth/ (accessed May 24, 2023).