PROJECT REPORT

Spatial Data Science

CSE 594
Fall 2022

Group 11 :

Name	ASU ID	ASURITE ID
Fenil Bharat Madlani	1222149747	fmadlani
Ananth Rangarajan	1225285633	arangar9
Priyanka Prabhu	1225344731	pprabhu5

1. Dataset:

For this project, we have used simulated trajectories dataset. The dataset is in json format. Each trajectory consisted of:

- **trajectory_id :** Each trajectory had a unique trajectory id.
- **Vehicle_id :** Vehicle Id is the id of vehicle in the particular trajectory
- **Trajectory :** Trajectory consisted of a tuple which had a list of location and a timestamp**.**
    - **Location:** The location was a list which consisted of a latitude and longitude.
    - **Timestamp:** The timestamp denoted the time at which the vehicle was present at that particular location.

The first task in the project was to load and store dataset in Apache Sedona Dataframe in a meaningful format in order to optimize the query processing. In order to do that we did following steps:

- We first exploded the trajectory column to get location and timestamp column
- The we dropped trajectory column
- After that we extracted the latitude and longitude for each location
- Lastly we formed the final dataframe which consisted of trajectory_id, vehicle_id, location, timestamp, latitude and longitude as a POINT

```
+-------------+----------+--------------------+--------------------+----------+
|trajectory_id|vehicle_id|            location|               point| timestamp|
+-------------+----------+--------------------+--------------------+----------+
|            0|         0|[33.4142915026357...|POINT (33.4142915...|1664511371|
|            0|         0|[33.4142645230006...|POINT (33.4142645...|1664511386|
|            0|         0|[33.4142375789892...|POINT (33.4142375...|1664511401|
|            0|         0|[33.4142105993541...|POINT (33.4142105...|1664511416|
|            0|         0|[33.4141836553427...|POINT (33.4141836...|1664511431|
|            0|         0|[33.4141566757076...|POINT (33.4141566...|1664511446|
|            0|         0|[33.4141297316962...|POINT (33.4141297...|1664511461|
|            0|         0|[33.4141027520611...|POINT (33.4141027...|1664511476|
|            0|         0|[33.4140758080497...|POINT (33.4140758...|1664511491|
|            0|         0|[33.4140488284147...|POINT (33.4140488...|1664511506|
|            0|         0|[33.4140218844032...|POINT (33.4140218...|1664511521|
|            0|         0|[33.4139949047682...|POINT (33.4139949...|1664511536|
|            0|         0|[33.4139679607567...|POINT (33.4139679...|1664511551|
|            0|         0|[33.4139409811217...|POINT (33.4139409...|1664511566|
|            0|         0|[33.4139140371102...|POINT (33.4139140...|1664511581|
|            0|         0|[33.4138870574752...|POINT (33.4138870...|1664511596|
|            0|         0|[33.4138601134637...|POINT (33.4138601...|1664511611|
|            0|         0|[33.4138331338287...|POINT (33.4138331...|1664511626|
|            0|         0|[33.4138061898173...|POINT (33.4138061...|1664511641|
|            0|         0|[33.4137792101822...|POINT (33.4137792...|1664511656|
+-------------+----------+--------------------+--------------------+----------+
only showing top 20 rows
```

Fig 1: Top 20 rows of dataframe in Apache Sedona

2. Algorithms:

Spatial Range Query - Spatial range query takes as input a range query window and a SpatialRDD and returns all geometries that have a specified relationship with the query window.

Algorithm:-

1) The spatial range query first creates an envelope of points(minimum and maximum latitude and longitude) given to the query.
2) This range query window can be specified in the following way:
    - val rangeQueryWindow = new Envelope(-90.01, -80.01, 30.01, 40.01)
    - The range query window can also be:
        - Point - val pointObject = geometryFactory.createPoint(new Coordinate(-84.01, 34.01))
        - Polygon - val polygonObject = geometryFactory.createPolygon(coordinates)
        - LineString - val linestringObject = geometryFactory.createLineString(coordinates)
3) We can specify different types of spatial predicates such as :
    - SpatialPredicate.INTERSECTS - geometry have at least one point in common with the query window
    - SpatialPredicate.COVERED BY - geometry has no point outside of the query window
    - SpatialPredicate.WITHIN - geometry is completely within the query window (no touching edges)
4) All the geometries which are covered by the above geometry or within it are returned from the query.
5) The output returned is another SpatialRDD object.
6) In terms of spark sql query, we can use the below query to return the geometries falling within the range window(Query from the assigment).
    - spark.sql(s " SELECT trajectory_id, vehicle_id, collect_list(timestamp) as timestamp, collect_list(location) as location FROM Range_Table WHERE ST_Within(Range_Table.point, ST_PolygonFromEnvelope($latMin,$lonMin,$latMax,$lonMax)) group by trajectory_d, vehicle_id")

Spatial Temporal Range Query - Spatial temporal range query takes as input a range query window along with minimum and maximum timestamp and a SpatialRDD and returns all geometries that have a specified relationship with the query window.

Algorithm:-

1. The spatial range query first creates an envelope of points(minimum and maximum latitude and longitude) given to the query.
2. This range query window can be specified in the following way:
3. val rangeQueryWindow = new Envelope(-90.01, -80.01, 30.01, 40.01)
4. The range query window can also be:
    a. Point - val pointObject = geometryFactory.createPoint(new Coordinate(-84.01, 34.01))
    b. Polygon - val polygonObject = geometryFactory.createPolygon(coordinates)
    c. LineString - val linestringObject = geometryFactory.createLineString(coordinates)
5. We can specify different types of spatial predicates such as :
6. SpatialPredicate.INTERSECTS - geometry have at least one point in common with the query window
7. SpatialPredicate.COVERED BY - geometry has no point outside of the query window

8. SpatialPredicate.WITHIN - geometry is completely within the query window (no touching edges)
9. All the geometries which are covered by the above geometry or within it are returned from the query.
10. The output returned is another SpatialRDD object.

7) Once the output is returned we can apply minimum and maximum timestamp clause to the returnrf SpatialRDD object.

8) In terms of spark sql query, we can use below query to return the geometries falling within the range window(Query from the assigment).

- spark.sql(s"SELECT trajectory_id,vehicle_id,collect_list(timestamp) as timestamp,collect_list(location) as location FROM RangeTemporal_Table WHERE ST_Within(RangeTemporal_Table.point, ST_PolygonFromEnvelope($latMin,$lonMin,$latMax,$lonMax)) and RangeTemporal_Table.timestamp between $timeMin and $timeMax group by trajectory_id, vehicle_id ")

KNN Query -A spatial K Nearest Neighbor query takes as input a K, a query point and an SpatialRDD and finds the K geometries in the RDD which are the closest to the query point.
Algorithm:-

1. Calculate the distance between the given query point and all the other query points in the space.
2. In this case, we calculate the Euclidean distance between two points.
3. Once the distances are computed, sort the distances in ascending order.
4. Return the top K records from the result above 3rd)
5. For example, If K = 5, we will return the top 5 records from above query.
6. The following has been shown in terms of spark sql query calculated in the project-phase-1:-
   - var df = spark.sql(s"SELECT b.trajectory_id,ST_Distance(a.point, b.point) as distance FROM KNN_Table a, KNN_Table b where a.trajectory_id = $trajectoryId and b.trajectory_id <> $trajectoryId")
   - df.createOrReplaceTempView("table")
   - df = spark.sql(s"SELECT trajectory_id, min(distance) as finalDistance FROM table group by trajectory_id order by finalDistance ASC limit $neighbors")
   - df.createOrReplaceTempView("finalTable")
   - df = spark.sql(s"SELECT trajectory_id FROM finalTable")

3. API and Front End

Front End -

Front-end for phase 2 of the project was built using html5 and javascript. We had 3 range queries to implement and hence we split the entire website into 3 different pages where each webpage implements a single query. We implemented buttons on the top of the main webpage where we displayed our team details and allows the user to choose which particular query he would like to view. Clicking on the buttons will then take the user to that specific page where we have implemented a simple form which would take in the inputs from the user and then clicking on submit button would send the form data to the backend where we are running the spatial queries and then the result returned from the backend is displayed in the iframe for the user to view the desired output.

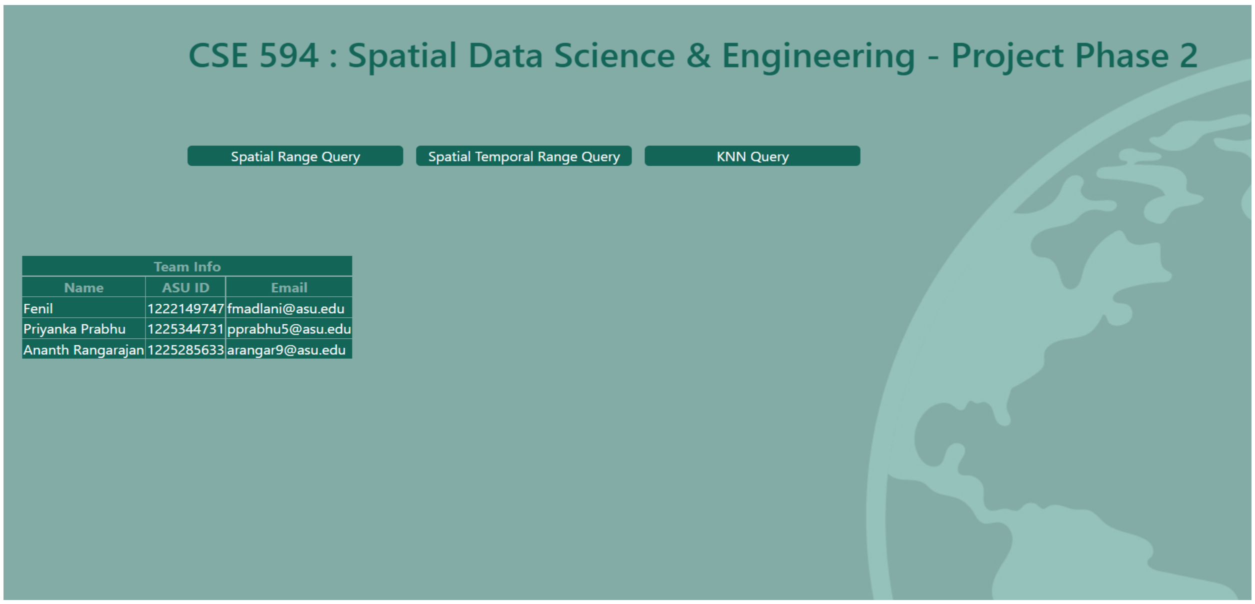

Fig 2: Front End UI for Phase 2

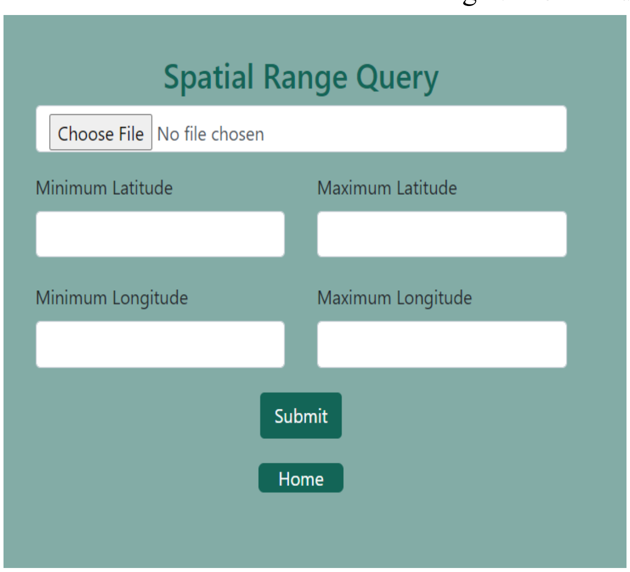

Fig 3: Front End UI for Spatial Range Query

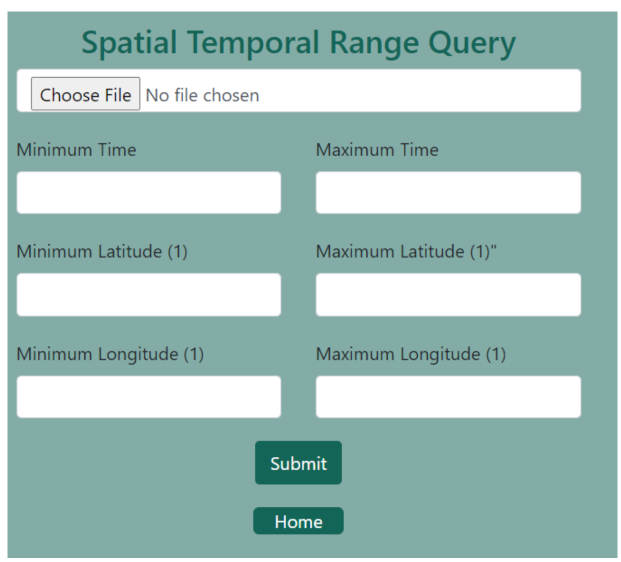

Fig 4: Front End for Spatial Temporal Range Query

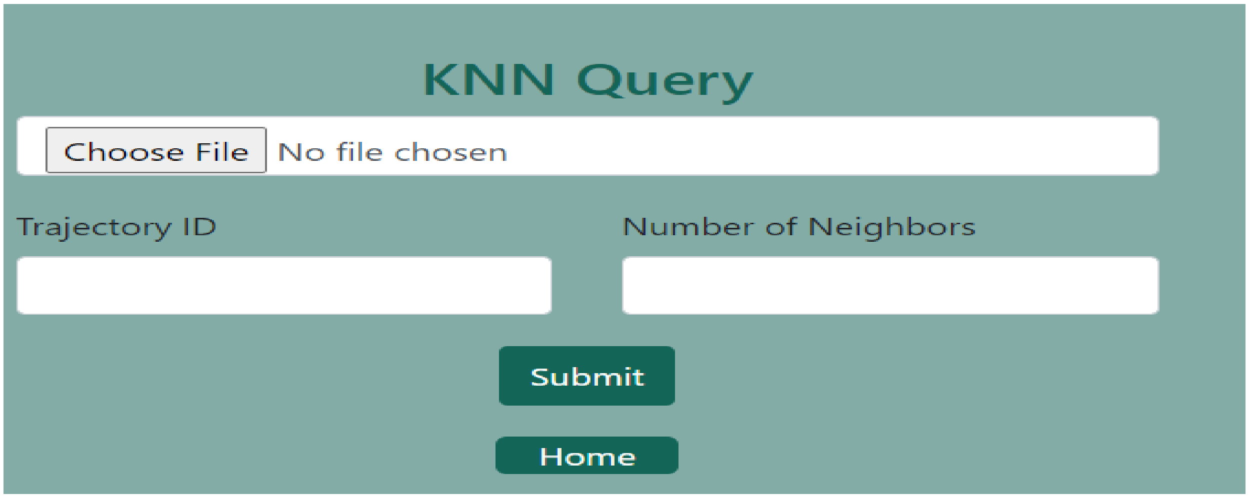

Fig 5: Front End UI for KNN Query

Back End -

Back-end for phase 2 was implemented using Python programming language along with Flask to incorporate the function calls and communicate with the front-end. Once the form data is received from the front-end to the back-end we call the specific scala function with the user inputs as the parameters and allow the scala to run the spatial command in the background. Once the scala command is successfully executed it is going to store the results in a specific folder which is used for creating the visualizations.

Visualization Layer -

We have used pydeck library to generate the Trips Layer for the data received as output from the scala command. We consider only the coordinates and timestamps data from the output generated from scala and pass it as parameter to the Trips Layer function. We are also making use of the other parameters such as rounded, opacity, trail length, current time and colors to distinguish between the different trips layer output generated for different spatial queries.

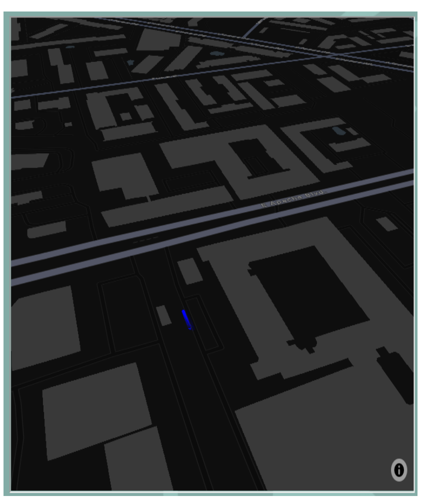

Fig 6: Spatial Range

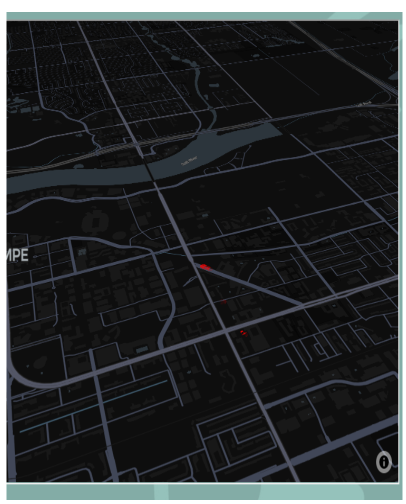

Fig 7: Spatial Temporal Range

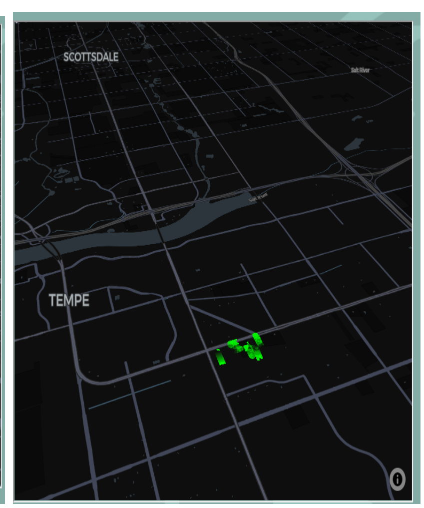

Fig 8: KNN