EC 504 Final Project Report

“Comparing KD-Trees and R-Trees for a location-based Food Recommendation system”

Team Information:
Mohnish Sandi, U14653579, mohnishs@bu.edu
Joshua Bone, U22742355, jbone@bu.edu
Pujan Paudel,U37789655, ppaudel@bu.edu
Muzammil Hussain, U52894967, muz@bu.edu
Tianhe Lei, U95904715, tianhel@bu.edu

Abstract:

In this project, we tested the performance of R-Trees and KD-Trees on storing and retrieving location-based data. By performing a multi-stage rigorous performance analysis, we observe that KD-Trees are faster than R-Trees in the building as well as the searching process. We then utilize the algorithms to build a food recommendation system. Our system runs a web-based application on the front end. Users can input their custom location, or they can allow the location services to extract their current location for them. They can then adjust other parameters such as cuisine type and the radius of the search for refining their search experience. We implemented the underlying algorithms in a local server, and the web application communicated the input and output with the server. We implemented the server as an Express application and the underlying algorithms on C++ and Python.

Introduction:

Discovering nearby restaurants is a very useful application of modern information systems. With rapidly increasing volumes of geospatial information being added to the internet daily, efficient algorithms for searching through this data have grown exponentially in importance. Nearest neighbor search algorithms are a natural tool to use when searching for restaurants in one’s area. In this project, we implement different data structures to enable K-nearest neighbor search based on parameters selected by the user. The goal is to quickly provide the user with recommendations for nearby restaurants, by leveraging these K-nearest neighbor algorithms. We will also compare the speed of each algorithm, in order to discern which data structure is more appropriate for this task.

R-Trees Background:

R-Trees are a data structure inspired by B-Trees. The main idea of this data structure is to construct bounding boxes around data points. These bounding boxes are constructed in a hierarchical nested manner, where at each level, you move to a smaller bounding box. Searching for a data point (in our case, a restaurant) in the R-Tree corresponds to traversing towards the leaves.

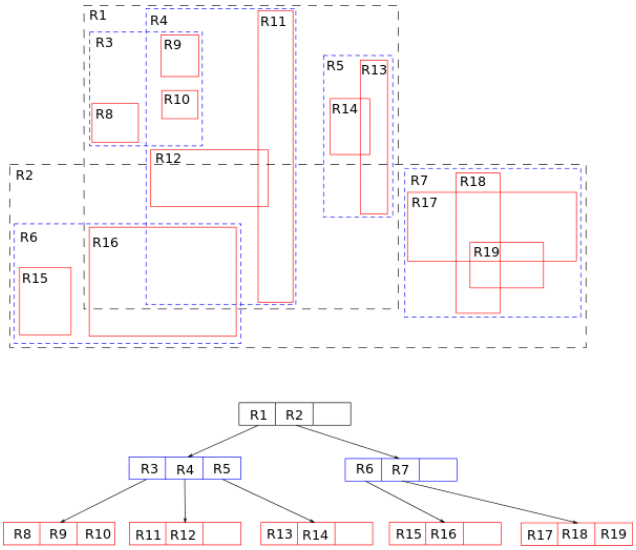


Figure 1. A visual representation of an R-Tree (public domain).

We used the implementation of R-Tree, provided by the libspatialindex library [1].

K-d Trees Background:

A K-d tree is a binary tree in which the leaf nodes are k-dimensional points (in our case, restaurants). The non-leaf nodes each represent a hyperplane, which separates all of the leaf nodes into one of two halfspaces. Search in a K-d tree corresponds to visiting these hyperplanes, and deciding which half space to exclude until the desired leaf node is reached.

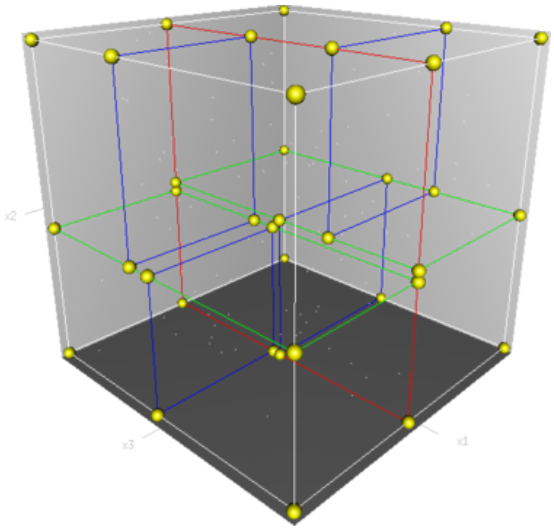


Figure 2. A visual depiction of a k-d Tree, with k=3 (GNU General Public License).

We used the implementation of K-d tree, provided by Scipy.org [2].

Performance comparison of algorithms:

Since the dataset we have in hand has only around 10000 data points, we cannot perform a true evaluation of the performance of the algorithms. Hence we select a large-scale dataset, from [3] which contains 424205 Points of Interest. We then study how the performance of the algorithms change as the size of the tree (number of points) increases.

i) Task: Tree Building

The below results demonstrate the time it takes to build the given data structure, for varying sizes of data structure.

# of Points	**R-Tree (time in microseconds)**	**KD-Tree (time in microseconds)**
10,000	306626	3069.87
100,000	3797016	31435.96
200,000	8473758	70704.69
300,000	12210408	118255.13
424,205	18114805	195240.97

From this table, it is clear that the KD-Tree takes a much shorter amount of time to construct than the R-Tree. The discrepancy becomes much more significant as the number of points increases.

ii) Task: Searching Points by N-Nearest Neighbors

Results averaged over 10 random points on each case.

Tree Size	**N**	**R-Tree (time in microseconds)**	**KD-Tree (time in microseconds)**

10,000	10	205	51.99
	100	490	68.80
	1000	2709	181.79
100,000	10	264	64.01
	100	606	71.69
	1000	2812	299.04
	10000	24199	2023.43
200,000	10	362	60.98
	100	674	74.41
	1000	2963	220.89
	10000	24944	1871.68
300,000	10	363	60.20
	100	670	115.10
	1000	2908	259.92
	10000	23567	2105.21
424,205	10	501	60.00
	100	817	89.09
	1000	3076	243.66
	10000	23448	2168.488

The KD-tree data structure consistently outperforms the R-Tree when it comes to searching by N-nearest neighbors, by a factor of approximately 10 across the tree sizes and number of neighbors.

iii) Task: Searching Points by Radius

Tree Size	**Radius (km)**	**R-Tree (time in microseconds)**	**KD-Tree (time in microseconds)**
10,000	1	652	184.24
	10	721	332.40

	100	3115	307.79
	1000	6491	303.74
100,000	1	1187	1492.73
	10	4836	4041.45
	100	27975	4161.57
	1000	66575	5273.22
200,000	1	1628	3073.07
	10	9369	8885.59
	100	58314	8369.99
	1000	135551	8534.09
300,000	1	2204	5201.98
	10	13353	13353.00
	100	84278	13280.63
	1000	205294	12660.67
424,205	1	2013	6750.13
	10	17002	18898.86
	100	116790	18770.86
	1000	279546	18466.61

Once again, the KD-tree data structure consistently outperforms the R-Tree when it comes to searching by radius.

Discussion of Results:

Based on our results, KD-trees are noticeably faster than R-trees. This remained true across all of our experiments. The building, searching for nearest neighbors, and radius searches of the KD-trees are much more efficient than R-trees. As we increase the size of the data sets, search and radius parameters, the computing time is longer, but the amount of improvement gained by switching from R-Trees to KD-Trees also increases.

For future projects building off of these results, we recommend using the KD-Tree data structure for food recommendation systems.

Instructions:

Our project will be deployed as a web-based application so both the R-Tree and the KDtree implementations do not have to be individually compiled.

We can run the benchmarking tests on SCC and observe the time taken for tree building, searching by radius, and searching by nearest neighbors. For algorithm testing purposes, we can also execute the individual algorithms provided by the web-based application on SCC.

The remainder of the section describes the commands which can be executed on SCC to perform the tests.

Navigate to /projectnb/ec504/ppaudel/FinalProject
Since we make use of libspatialindex shared library, we should set the environment variables such that it can load the required libraries. Hence before running any of the tests / programs, we should run the following command :

. ./envvarsetup.sh which lies in /projectnb/ec504/ppaudel/FinalProject
Or , Alternatively *export LD_LIBRARY_PATH=/projectnb/ec504/ppaudel/FinalProject/usr/lib* should also work

The additional code resources (for the Web demo) are kept under directory /projectnb/ec504/ppaudel/FinalProject/Additional

1) **Benchmark tests** :

    a) Tree Building Timing on R-Tree
    Command to be run : ./src/testbed_buildtree
    Command line argument : input file for the list of points
    Example : **./src/testbed_buildtree input/poi_data_10000.txt**
    Other invocations might be with command-line arguments
    input/poi_data_100000.txt,
    input/poi_data_200000.txt,
    input/poi_data_300000.txt,
    input/poi_data_all.txt

Output :

```
[ppaudel@scc1 FinalProject]$ ./src/testbed_buildtree input/poi_data_10000.txt
Time taken by function: 275229 microseconds
[ppaudel@scc1 FinalProject]$
```

    b) Tree Building Timing on Kd-Tree
    Command to be run : python3 src/benchmark_buildtree.py

Command line arguments : input file for the list of points

Example : python3 src/benchmark_buildtree.py input/poi_data_100000.csv

Other invocations:
input/poi_data_100000.csv
input/poi_data_200000.csv,input/poi_data_300000.csv
input/poi_data_all.csv

Output :

```
[ppaudel@scc1 FinalProject]$ python3 src/benchmark_buildtree.py input/poi_data_100000.csv
--- Tree building took 38224.220275878906 seconds ---
```

c) Search by Radius Timing on R-Tree
Command to be run : ./src/testbed_radius
Command line argument : [input file for the list of points]

Example : **./src/testbed_radius input/poi_data_20000.txt**

Other invocations:
input/poi_data_100000.txt,
input/poi_data_300000.txt,
input/poi_data_all.txt

Output :

```
[ppaudel@scc1 FinalProject]$ ./src/testbed_radius input/poi_data_200000.txt
Time taken by radius 1: 1781 microseconds
Time taken by radius 10: 6201 microseconds
Time taken by radius 100: 27440 microseconds
Time taken by radius 1000: 58569 microseconds
[ppaudel@scc1 FinalProject]$
```

d) Search by Radius Timing on KdTree
Command to be run : python3 src/benchmark_searchradius.py
Command line argument : [input file , targetradius]
Example : **python3 src/benchmark_searchradius.py input/poi_data_300000.csv 10**

Output :

```
[ppaudel@scc1 FinalProject]$ python3 src/benchmark_searchradius.py input/poi_data_300000.csv 10
--- Average execution took 21020.817756652832 microseconds ---
```

e) Search *n* Nearest Neighbor Timing on R-Tree
Command to be run : ./src/testbed_searchpoints
Command line argument : [input file for the list of points]

Example : **./src/testbed_searchpoints input/poi_data_300000.txt**

Other invocations:
input/poi_data_100000.txt,
input/poi_data_200000.txt,
input/poi_data_all.txt

Output :

```
[ppaudel@scc1 FinalProject]$ ./src/testbed_searchpoints input/poi_data_300000.txt
Time taken by NN 10 : 171 microseconds
Time taken by NN 100 : 411 microseconds
Time taken by NN 1000 : 1699 microseconds
Time taken by NN 10000 : 13632 microseconds
[ppaudel@scc1 FinalProject]$
```

f) Search n Nearest Neighbor Time on Kd-Tree
Command to be run : python3 src/benchmark_neighbor.py
Command line argument : [input file , targetneighbors]
Example : **python3 src/benchmark_neighbor.py input/poi_data_300000.csv 5**

```
[ppaudel@scc1 FinalProject]$ python3 src/benchmark_neighbor.py input/poi_data_300000.csv 5
--- 100.1119613647461 microseconds ---
```

2) **Algorithm tests**:
   a) R-Tree
      i) Search by Nearest Neighbor
      Command to be run : ./src/RTree_NN
      Command line argument : [lat, lng, number of results]
      Example : **./src/RTree_NN 28.4581066 77.5281286 15**
      Output :

```
[ppaudel@scc1 FinalProject]$ ./src/RTree_NN 28.4581066 77.5281286 15
310063,28.4581,77.5281, Ali Baba Caves
310067,28.4581,77.5281, Golden Chaat
18373737,28.4581,77.5282, Get Set Go
18254400,28.458,77.528, The Anna
18124366,28.4579,77.5282, Sip & Snacks
18364351,28.458,77.5283, Chakna
312278,28.4581,77.5279, Baker's Zone
309545,28.4579,77.5281, Kathi Junction
302139,28.4583,77.5279, Domino's Pizza
18382359,28.4634,77.5297, Hunger's Hut
4505,28.4642,77.5215, Cooks Cafe - Jaypee Greens
18317498,28.4642,77.5215, The Butler & The Chef - Jaypee Greens
18377927,28.4645,77.518, Night Food Service
8441,28.4694,77.5184, Tea Lounge - Jaypee Greens
8422,28.4697,77.5181, Baker's Studio - Jaypee Greens
8344,28.4697,77.5181, Matrix - Jaypee Greens
[ppaudel@scc1 FinalProject]$
```

      ii) Search by Radius

Command to be run : ./src/RTree_radius
Command line argument : [lat, lng, radius]
Example : **./src/RTree_radius 28.4248315 77.0393103 1**

Output :

```
[ppaudel@scc1 FinalProject]$  ./src/RTree_radius 28.4248315 77.0393103 1
18306533,28.4159,77.0418, The Brewhouse
5019,28.4165,77.0412, Fortune Deli - Fortune Select Excalibur
4719,28.4165,77.0412, India on my Plate - Fortune Select Excalibur
6847,28.4167,77.0413, Cafe Coffee Day
8147,28.4167,77.0414, FOA - The Flavours of Arabia
18277212,28.4184,77.0384, Cyber Adda 24
306883,28.4184,77.0382, Subway
18303696,28.4186,77.0383, Tandoori Nation
18306548,28.42,77.0382, Nukkadwala
303003,28.4201,77.0407, The Golden Dragon
301310,28.4209,77.0384, Sagar Ratna
3909,28.4234,77.0395, Viva Hyderabad
309030,28.4234,77.0396, Hong Kong Express
18398606,28.4237,77.0392, Burger King
313085,28.4245,77.0389, Burger Singh
18396451,28.4248,77.0393, K Lab
6877,28.4248,77.0393, Domino's Pizza
18237941,28.4249,77.0392, Pind Balluchi
309466,28.4249,77.0392, Dunkin' Donuts
18365865,28.4256,77.033, Fat Monk
18161567,28.4257,77.0322, Tea Halt
```

b) Kd-Tree
   i) Search by Nearest Neighbor
      Command to be run: .python3 src/KDTree_test.py
      Command line argument : [lat, lng, number of results]
      Example : **python3 src/KDTree_test.py nn 28.4581066 77.5281286 15**

   Output :

```
[ppaudel@scc1 FinalProject]$  python3 src/KDTree_test.py nn 28.4581066 77.5281286 15
{'name': 'Golden Chaat', 'latitude': 28.4581066, 'longitude': 77.5281286, 'Address': 'Ground Floor, MSX Mall, Greater Noida, Noida', 'distance'
: 0.0}
{'name': 'Ali Baba Caves', 'latitude': 28.4581066, 'longitude': 77.5281286, 'Address': '1st Floor, Ansal Plaza Mall, Pari Chowk, Greater Noida,
 Noida', 'distance': 0.0}
{'name': 'Get Set Go', 'latitude': 28.4580767, 'longitude': 77.5282389, 'Address': '2nd Floor, MSX Mall, Swarn Nagari, Greater Noida, Noida', '
distance': 0.011283725818980468}
{'name': 'The Anna', 'latitude': 28.458049300000003, 'longitude': 77.52799420000001, 'Address': 'Shop 135, MSX Mall, Surajpur Site IV, Delhi NC
R, Greater Noida, Noida', 'distance': 0.014602165246341154}
{'name': 'Chakna', 'latitude': 28.458033399999998, 'longitude': 77.5283078, 'Address': 'Shop 1 & 2, Food Court, 3rd Floor, MSX Mall, Greater No
ida, Noida', 'distance': 0.019316949103825105}
{'name': 'Sip & Snacks', 'latitude': 28.4579354, 'longitude': 77.5282182, 'Address': 'MSX Mall, Greater Noida, Noida', 'distance': 0.0209550594
51989864}
{'name': "Baker's Zone", 'latitude': 28.4580751, 'longitude': 77.5278832, 'Address': 'MSX Mall, Site 4, Greater Noida, Noida', 'distance': 0.02
4244358032909315}
{'name': 'Kathi Junction', 'latitude': 28.4578535, 'longitude': 77.52814740000001, 'Address': 'Shop 132, Ground Floor, MSX Mall, Greater Noida,
 Noida', 'distance': 0.02820338174275576}
{'name': "Domino's Pizza", 'latitude': 28.458344699999998, 'longitude': 77.527913, 'Address': 'Shop 114, Ground Floor, MSX Mall, Surajpur Indus
trial Area, Greater Noida, Noida', 'distance': 0.03384055072542462}
```

ii) Search by Radius

Command to be run : python3 src/KDTree_test.py

Command line argument : [type, lat,lng, radius]

Example : **python3 src/KDTree_test.py radius 28.42 77.0393103 0.5**

Output :

```
[ppaudel@scc1 FinalProject]$ python3 src/KDTree_test.py radius 28.42 77.0393103 0.5
{'name': 'Sadabahar Hotel', 'latitude': 28.4157403, 'longitude': 77.04034490000001, 'Address': '3 Km Milestone, Opposite 9X Mall, Sohna Road, G
urgaon'}
{'name': 'India on my Plate - Fortune Select Excalibur', 'latitude': 28.4164952, 'longitude': 77.0411995, 'Address': 'Fortune Select Excalibur,
 Sohna Road, Gurgaon'}
{'name': 'Fortune Deli - Fortune Select Excalibur', 'latitude': 28.4164952, 'longitude': 77.0411995, 'Address': 'Fortune Select Excalibur, Sohn
a Road, Gurgaon'}
{'name': 'The Golden Dragon', 'latitude': 28.420119500000002, 'longitude': 77.0406597, 'Address': 'Ground Floor, Spazedge Mall, Sohna Road, Gur
gaon'}
{'name': 'FOA - The Flavours of Arabia', 'latitude': 28.4167026, 'longitude': 77.0413756, 'Address': '3rd Floor, Ninex City Mart Mall, Sohna Ro
ad, Gurgaon'}
{'name': 'Cyber Adda 24', 'latitude': 28.418352399999996, 'longitude': 77.0383803, 'Address': 'Unit 1-A, Ground Floor, Weldone Tech Park, Near,
 Sohna Road, Gurgaon'}
{'name': 'Subway', 'latitude': 28.4184165, 'longitude': 77.03824350000001, 'Address': '7, Ground Floor, Welldone Tech Park, Sohna Road, Gurgaon
'}
{'name': 'Cafe Coffee Day', 'latitude': 28.416683199999998, 'longitude': 77.04128940000001, 'Address': '23, Ground Floor, Ninex City Mart Mall,
 Sohna Road, Gurgaon'}
```

iii) Filter by Cuisine

Command to be run :

python3 src/KDTree_test.py radius 28.4581066 77.5281286 1 "Chinese"

Or

python3 src/KDTree_test.py nn 28.4581066 77.5281286 1 "North Indian"

Last argument is the cuisine type , ranging from “Chinese”, “Cafe”, “North Indian”,”South Indian”

Output :

```
[ppaudel@scc1 FinalProject]$ python3 src/KDTree_test.py radius 28.4581066 77.5281286 1 "Chinese"
Some sort of cuisine selection
{'name': 'The Anna', 'latitude': 28.458049300000003, 'longitude': 77.52799420000001, 'Address': 'Shop 135, MSX Mall, Surajpur Site IV, Delhi NC
R, Greater Noida, Noida'}
{'name': 'Ali Baba Caves', 'latitude': 28.4581066, 'longitude': 77.5281286, 'Address': '1st Floor, Ansal Plaza Mall, Pari Chowk, Greater Noida,
 Noida'}
{'name': 'Sip & Snacks', 'latitude': 28.4579354, 'longitude': 77.5282182, 'Address': 'MSX Mall, Greater Noida, Noida'}
{'name': 'Chakna', 'latitude': 28.458033399999998, 'longitude': 77.5283078, 'Address': 'Shop 1 & 2, Food Court, 3rd Floor, MSX Mall, Greater No
ida, Noida'}
{'name': 'Get Set Go', 'latitude': 28.4580767, 'longitude': 77.5282389, 'Address': '2nd Floor, MSX Mall, Swarn Nagari, Greater Noida, Noida'}
```

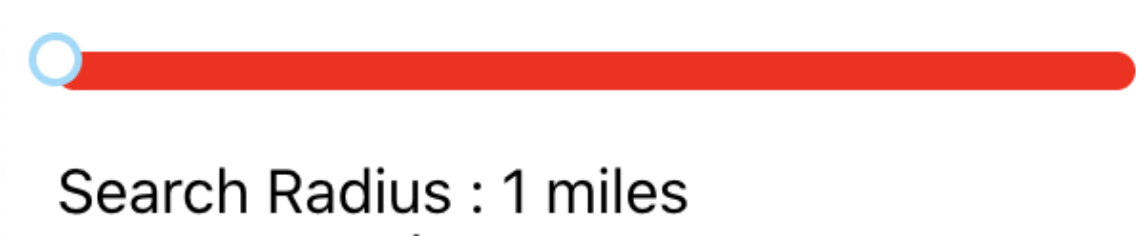
3) Web-based visualization

Our web-based application is hosted on the Heroku app and accessible via http://ec504project.herokuapp.com/. Due to the lack of an SSL certificate, the web application doesn't open with HTTPs, hence we need to manually enter HTTP for now, as a limitation. The browser opens https by default, hence we need to enter the http:// URL to run the demo properly.

Web application UI to run the algorithms :

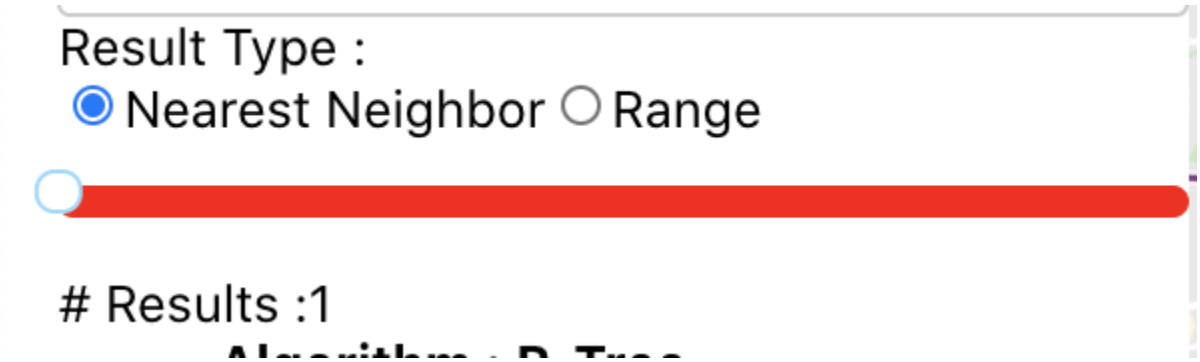
a) Algorithm selection
We can select which of the underlying algorithms we want to run (KD Tree / R-Tree) by selecting this dropdown

b) Search Radius Adjustment
The search radius can be adjusted by using this slider



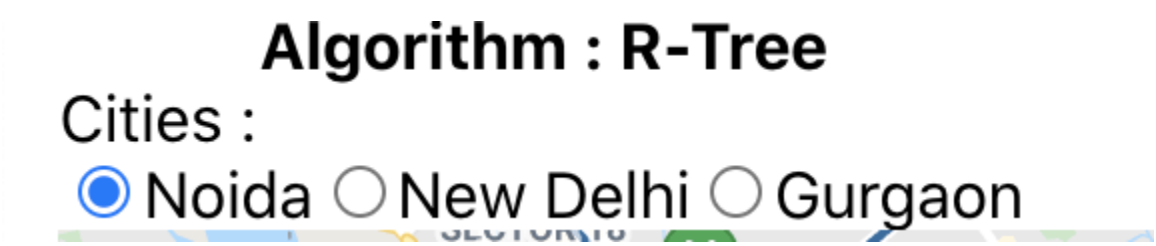
c) Result type selection

Users can select between two types of results: range-based , or number of nearest neighbors based.

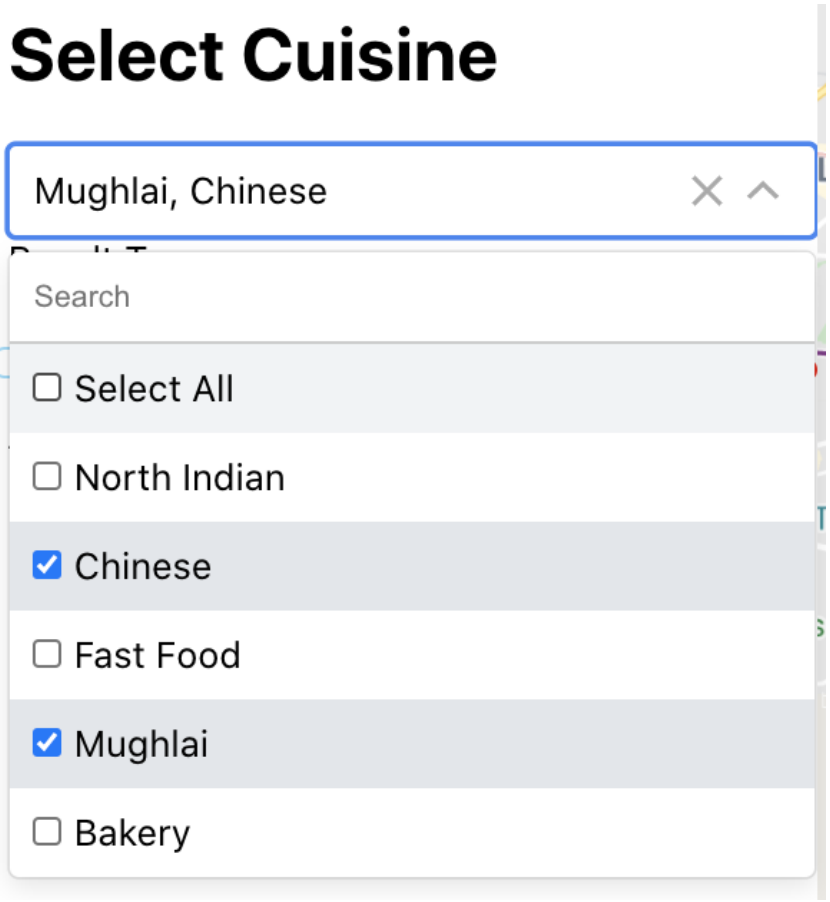


d) Cities selection

Our restaurant dataset consists of 9552 unique restaurants, but more than 3000 of them are located in the three major cities of India: Noida, New Delhi, and Gurgaon. Hence, for this demo we allow the users to center the target location in one of these cities and observe the results of the algorithm working.

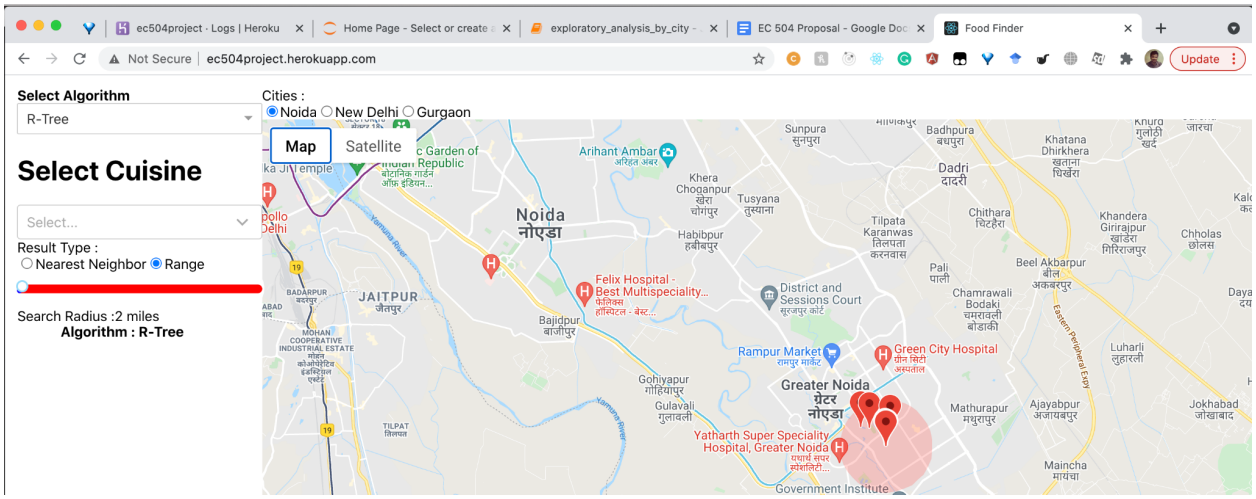


e) Cuisine selection

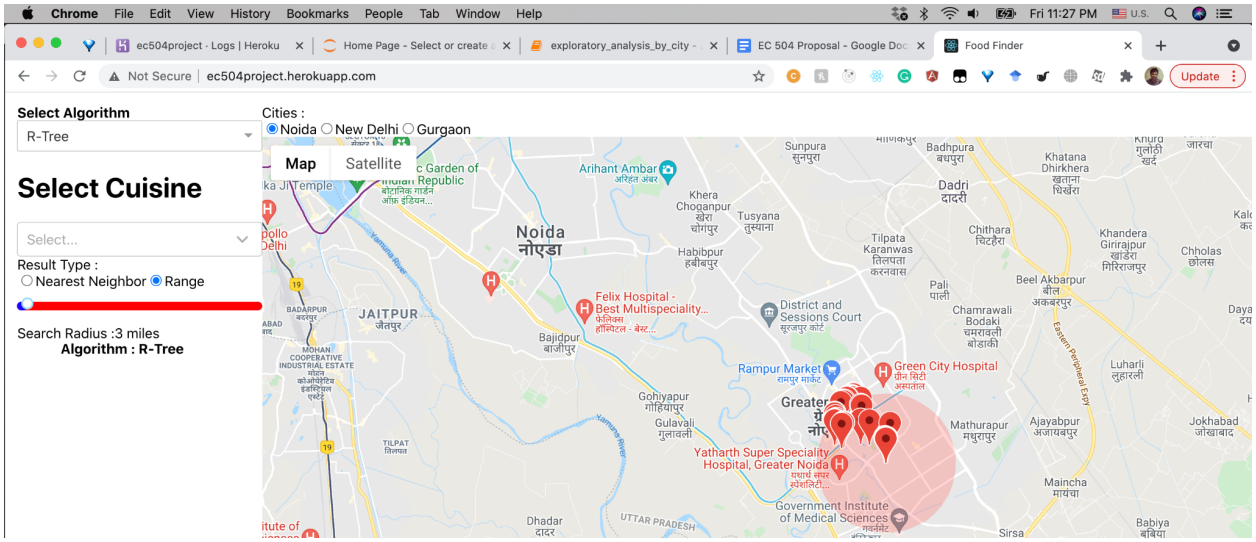


f) Visualization of Results

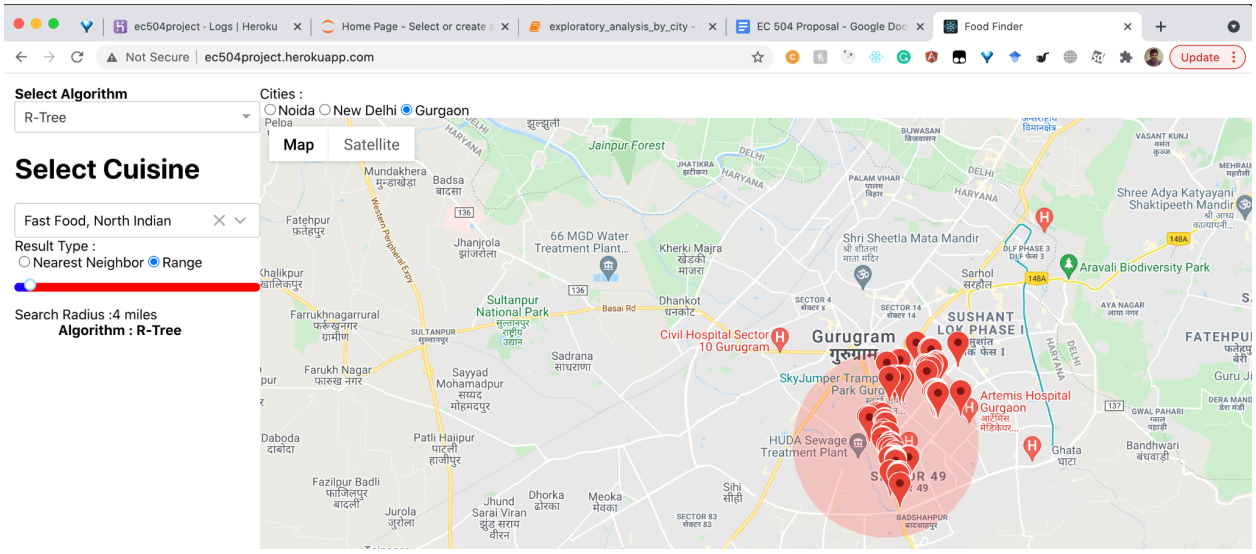
ii) Results with a smaller radius of search



iii) Results with an increased radius of search



iii) Results with different city and change of cuisines



References

[1] https://github.com/libspatialindex/libspatialindex
[2] https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.cKDTree.html
[3] https://www.kaggle.com/ehallmar/points-of-interest-poi-database