Fall 2024 EC504 Midterm Report:

Hashing and Trees for Fast Approximate Nearest Neighbor *Geospatial* Search

Chris Krenz, ckrenz@bu.edu, U19402301

Table of Contents

General Description

This project is based on the Suggested Project 5: Hashing and Trees for Fast Approximate Nearest Neighbor Search. I am implementing multiple algorithms/data structures to perform approximate nearest neighbor search of geospatial data and comparing their relative speed and accuracy. Specifically, given a pair of coordinates—and ultimately additional dimensions, such as elevation, land type, and population density—the program can identify zip codes near those coordinates. I am implementing search algorithms based on:

- Multi-Table Locality-Sensitive Hashing (LSH)[1] (MVP already implemented)
- Approximate K-D Trees with Priority Search[2,3] (MVP already implemented)
- R-Trees[4] (WIP)

The datasets I have already started to use include:

- SimpleMaps[5] (US zip code and longitude/latitude data consolidated from USPS, Census Bureau, etc.)
- OpenStreetMap[6] (for additional POI and feature data)

I also plan to incorporate additional dimensions from USGS, NASA Earthdata,[7] and GADM,[8] if time permits.

A final stretch goal will be to implement a simple interactive application that will allow users to perform these searches in real time.

Data

One downside of using geospatial data instead of, say, image data, is that the data intrinsically has few dimensions. We would expect LSH to perform better than K-D Trees with high-dimensional spaces, so I anticipate the K-D trees will perform similar to, if not better than, LSH. As such, I will attempt to merge additional dimensions of data, such as elevation, land type (rural, etc.), population density, point-of-interest proximities, socio-economic data, vegetation, average temperature, etc. However, this comes with a substantial challenge in data cleaning and merging, as most of the data sources I have found provide data in different formats, and not all datasets provide coordinate information, making cross-referencing of data more difficult. I will attempt to strike a balance between feasibility and dimensionality. The upside of this lower-dimension data is that the datasets are a much more manageable size than most image datasets.

Thus far, I have been having some difficulty acquiring data from OpenStreetMap, as the downloads from the primary source (Geofabrik) have been failing (currently troubleshooting this). I am also hesitant to use excessively large datasets, as it would make it more difficult for you to run the code independently (OSM data alone would be on the order of 10s of GB). For the time being, I am primarily relying on the lower-dimensional SimpleMaps data, which provides data on US zip codes and coordinates. For the final report and demo, I plan to use a much larger (and higher dimension) dataset, but for now, while I am rapidly iterating in developing the algorithms, a smaller dataset makes sense. This will also allow me to package the data directly with the code to simplify the instructions as much as possible.

Feature List

- Primary Goals
  - Construction of searchable geospatial dataset
    - At a bare minimum, this dataset should contain data from OpenStreetMap to provide coordinates and GADM (or a comparable dataset) to provide zip codes. I am hopeful I can also integrate additional datasets listed above to increase dimensions.
  - Search based on multi-table Locality-Sensitive Hashing
    - To implement Multi-Table LSH, I'll create multiple hash tables where each table uses a different set of hash functions. For each query, I'll combine the results to find candidate nearest neighbors, improving the chance of capturing similar items. The goal is to reduce false negatives and boost recall.
  - Search based on K-D Trees with Priority Search
    - I will first construct a k-d tree, then during search, I will maintain a priority queue where nodes are prioritized by their (multi-dimensional) distance from the query coordinates. This prioritization will speed up the nearest neighbor search.
  - Algorithm to measure the speed and accuracy of these search algorithms
    - I will measure the speed by tracking the average query time per search as well as the time needed to build the data structure. For accuracy, I will compare the suggested options to the ground truth of closest zip-codes. If a method retrieves one of the top few actual nearest neighbors, that will be considered a successful search.
- Stretch Goals
  - Implement R-Trees
    - To implement R-trees, I will create a tree structure where each node represents a bounding rectangle that encloses spatial objects. As data is inserted, nodes are split dynamically to minimize overlap between bounding boxes.

- Implement Interactive App
  - Ideally I will be able to implement a GUI that displays the map that auto-focuses to the target region, places a pin at the selected coordinates, as well as highlights the zip-codes predicted by the algorithm. However, I may need to scale this back to a minimal GUI, some search boxes, drop-down lists, and checkboxes to fully specify a query and a frame to display the results in text form. Regardless, the user should be able to select one of the algorithms from a drop-down to compare performance.

Completed and Planned Tasks

- Completed
  - Topic chosen and researched: Week of Oct 14th
  - *Partial* OpenStreetMap data acquired: Week of Oct 14th
  - SimpleMaps data acquired: Week of Oct 28th
  - Working draft of the LSH: Week of Nov 4th
  - Working draft of the KD-Tree: Week of Nov 11th
  - Started experimenting with R-Tree: Week of Nov 11th
  - Basic benchmarking: Week of Nov 11th
- Planned (*see Timeline/Gantt chart for estimated timeline/schedule*)
  - Ongoing Data Collection, Merging, and Cleaning
    - This has proven more challenging than I anticipated, with
    - In addition to OpenStreetMap data, I am currently planning on using USGS, NASA Earthdata, and GADM data. I am figuring how to get the best cross-sections of USGS and Earthdata data, but GADM data has thus far proven inaccessible (download links not working; I will continue searching; ultimately this dataset may not even be necessary given other sources).
    - As I gather data, I will gauge the number of data sources I can feasibly access and merge (and therefore the number of dimensions in the final dataset). If necessary, I may decide to limit the geographic space (e.g. to North America) if doing so will enable me to have higher dimensional data.
  - Finalize Multi-Table LSH and Approximate KD Trees with Priority Search
    - I still need to implement priority search in the KD Trees algo, increase the number of dimensions (which I hope to be simple, as I designed the algorithms with this in mind), and perform more testing and bug fixing. But at least for now, I do have working prototypes.
  - Implementation of R-Trees (Stretch Goal)
    - Implementing R-Trees might be slightly more challenging as they are fundamentally different from KD-Trees and intended for range queries to produce exact matches, rather than approximate matches.
    - I welcome feedback on the importance of implementing this algorithm in a solo project, given the other objectives.
  - Creation of Interactive App (Stretch Goal)
    - I am hopeful I will be able to implement some sort of interactive application to make testing and running program easier. However, more sophisticated features, like a map display with real-time highlighting of or zooming in on target regions may not be feasible in the allotted time. As such, I may end up implementing a simpler version as described above.
  - Demo Preparation
    - The demo will hopefully consist of a live demo, with GUI, showing the program's search feature. Alternatively, I could perform the search from CLI or else prepare slides that highlight results—and metrics—of prospective searches.

Prototype – What it does

This prototype has scripts to perform the following:

- Importing from either a .pbf (from OSM, using the osmium package) or .csv file.
- Running either LSH-based or KD-Tree-based search for nearby zip-codes given coordinate inputs.
    - e.g. coordinate inputs of latitude=18.34, longitude=-64.92 would yield 5 results like the following:
      `Zip Code: 00985, Location: (18.40628, -65.94767)`
- Performing a benchmarking of the two algos, reporting their accuracy (compared to brute-force search) and run times.

The primary point of entry is main.py. Running this algorithm will result in an output similar to the following:

```
(venv) $ python main.py
Multi-Table LSH - Time: 0.04073s, Accuracy: 0.58
Approximate KD Tree - Time: 0.00011s, Accuracy: 0.06
```

This result makes sense, as there is a clear tradeoff between accuracy and speed. LSH provides a reasonably high accuracy result but takes a ~couple order of magnitudes longer to run. Approximate KD Tree, however, is much less accurate. This is a good initial benchmark, but I will be very interested to see how these results change when I increase both the amount of data and number of dimensions (and potentially compare with R-Trees).

Prototype – How to run it

The code is accessible at my public GitHub repo: https://github.com/chris-krenz/504-geospatial-project. This is a Python project. Instructions for installing Python can be found here: https://wiki.python.org/moin/BeginnersGuide/Download.

You will want to perform the following steps:

1. Download the code (either from GitHub or the assignment)
2. cd to the root directory
3. Run the following commands to create a virtual environment, install the packages, and run the main script:

```
python -m venv venv

source venv/bin/activate        OR for Windows        venv\Scripts\activate.bat

pip install -r requirements.txt

python src/main.py
```

4. (Optionally 'deactivate' to exit the virtual environment)

Alternatively, you can build and run it with **Docker** using the following command:

```
docker-compose up --build
```

After a handful of seconds (about ~30 on an M2 mac), you should see a console output similar to that shown above.

Challenges

The primary challenge I have been dealing with is to do with downloading and merging datasets. Large data files (10s or 100s of GB) associated with global–or even national–regions are failing partway through the download, making the data a bit more difficult to access than I had initially anticipated. I have mostly been trying to pull from OSM, as that source has a variety of different data types that can increase dimensions substantially. I am actively troubleshooting this, and there are a variety of other sources that I am exploring.

In order to ensure I did not stall on data collection, I started with a simpler dataset from SimpleMaps that contains the basic info I needed to start defining the algorithms. However, I would still ultimately like to find larger and better data sources. (That said, I may still generally package small, sample data with the code so that you are able to easily test the program.)

While I had initially hoped to provide global searc, or at least US search, I may end up limiting this to smaller regions, such as Massachusetts. Increasing the number of dimensions and completing the R-trees algorithm seem like a higher priority.

Conclusion

Although I have faced some unexpected setbacks, I am pleased to have gotten the basic algorithms functioning and benchmarked. My focus now will be on merging more data sources to increase the number of dimensions, implementing R-trees, and then creating a simple interactive app to demo the algorithms. While completion of these stretch goals is not guaranteed, I think I am on track for their completion. If time runs out, the interactive app will likely be the first feature I drop.

Updated Timeline/Gantt Chart

	Week of...						
	Oct 21	Oct 28	Nov 4	Nov 11	Nov 18	Nov 25	Dec 2
Data Collection							
Data Merging							
Data Cleaning							
Multi-Table LSH							
Approximate KD Trees with Priority Search							
Implementation of R-Trees (Stretch Goal)							
Algorithm Testing and Bug-fixing							
Gather Metrics and Benchmark Algorithms							
Creation of Interactive App (Stretch Goal)							
App Testing and Bug-fixing (Stretch Goal)							
Demo Preparation							

References

1. Datar, M., Immorlica, N., Indyk, P. & Mirrokni, V. S. Locality-sensitive hashing scheme based on p-stable distributions. in *Proceedings of the twentieth annual symposium on Computational geometry* 253–262 (Association for Computing Machinery, New York, NY, USA, 2004). doi:10.1145/997817.997857.
2. A comparison of k-nearest neighbour algorithms with performance results on speech data - KU Leuven. https://kuleuven.limo.libis.be/discovery/fulldisplay/lirias1945818/32KUL_KUL:Lirias.
3. Arya, S., Mount, D. M., Netanyahu, N. S., Silverman, R. & Wu, A. Y. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J ACM* **45**, 891–923 (1998).
4. Hadjieleftheriou, M., Manolopoulos, Y., Theodoridis, Y. & Tsotras, V. J. R-Trees – A Dynamic Index Structure for Spatial Searching. in *Encyclopedia of GIS* (eds. Shekhar, S. & Xiong, H.) 993–1002 (Springer US, Boston, MA, 2008). doi:10.1007/978-0-387-35973-1_1151.
5. US Zip Codes Database | Simplemaps.com. *SimpleMaps* https://simplemaps.com/data/us-zips.
6. Geofabrik Download Server. *GeoFabrik: OpenStreetMap Data* https://download.geofabrik.de/ (2018).
7. Earth Science Data Systems, N. Find Data | Earthdata. https://www.earthdata.nasa.gov/learn/find-data (2021).
8. GADM. *Database of Global Administrative Areas* https://gadm.org/data.html (2022).